High-Performance Numerical Software for Control Systems Analysis and Design, and Subspace-Based System Identification

WGS-report 97-2*

Fellow: Dr. Vasile Sima
Research Institute for Informatics
Computer Process Control Laboratory
Bd. Mareşal Averescu, Nr. 8–10,
71316 Bucharest 1, ROMANIA
Telex: 06500+ 11891 icpci r
Phone: (40) 1 665 33 90, Fax: (40) 1 312 85 39,
Email: vsima@roearn.ici.ro
Vasile.Sima@esat.kuleuven.ac.be

Scientific Coordinators:
Prof. Dr. Ir. Sabine Van Huffel
Prof. Dr. Ir. Bart De Moor
Katholieke Universiteit Leuven
Department Electrotechn.—Afd. ESAT/SISTA
Kardinaal Mercierlaan 94
B–3001 Heverlee
Belgium
Phone : +32-(0)16-32 17 09, Fax : +32-(0)16-32 19 70
Email : Sabine.VanHuffel@esat.kuleuven.ac.be
Email : Bart.DeMoor@esat.kuleuven.ac.be

March 25, 1997

*This research was supported by the Belgian Office for Scientific, Technical and Cultural Affairs.

Abstract

New results on performance of some recently developed algorithms and associated software for control system analysis and design, as well as for multivariable state space system identification based on subspace techniques are presented.

The main results achieved can be summarized as:

1. Improvements of the efficiency, reliability, and accuracy of the new SLICOT library routines.
2. Improvements of the efficiency of software for multivariable state space system identification based on subspace techniques.

The new codes are usually at least two times (and frequently more) faster than equivalent computations performed in Matlab. The accuracy is generally the same, or better, except for the subspace techniques for system identification, where, sometimes, it is worse than that achieved by the best, robust Matlab implementation. Further analysis is needed to find the source of this potential inaccuracy.

Keywords

Control systems analysis and design, dynamical systems, identification techniques, Matlab toolbox, numerical linear algebra, numerical methods, subroutine library.

1 Introduction: Summary of the research activity and results

An important current research topic is the investigation of fundamental algorithms for high-performance computing, as well as of practical approaches of using high-performance algorithms and software, efficient on modern computer architectures. The special scientific and technical interest in this topic is illustrated by the well-funded U.S.A. program entitled *High Performance Computing and Communications*, and the similar program initiated by the European Community countries, as well as by the large number of scientific symposia organized in recent years, devoted to this topic.

This report reflects the author's efforts to bring high-performance computing in the areas of control systems analysis and design, as well as subspace-based system identification.

My scientific research activity during the research stay at Katholieke Universiteit Leuven (October 1, 1996—March 31, 1997) can be summarized as follows:

1. Development of a new version of the SLICOT library for computer-aided control system analysis and design, based on the high level BLAS (Basic Linear Algebra Subprograms) routines, and the newly available numerical linear algebra package LAPACK.

2. Investigation of algorithms and Fortran software for multivariable state space system identification based on subspace techniques.

3. Presentation of new algorithmic techniques and related software for subspace-based system identification at several seminars:

    - Department of Mathematical Engineering, Université Catholique de Louvain, Belgium, October 31, 1996;
    - Department of Electrical Engineering, Katholieke Universiteit Leuven, Belgium, December 12, 1996;

- Department of Mechanical and Marine Engineering, Technical University Delft, The Nederlands, March 12, 1997.

4. Participation to the seminars organized by the research group ESAT-SISTA, Department of Electrical Engineering, Katholieke Universiteit Leuven, and to the ICCOS Study day on nonlinear identification, nonlinear time series analysis and neural identification, January 30, 1997, Leuven.

5. International cooperation with the members of the **Working Group on Software (WGS)** (of which I am member from 1996), and with many other recherchers from Belgium and abroad.

The **main results** achieved so far are:

1. Improving the efficiency, reliability, and accuracy of the SLICOT library routines, as shown in Section 2.

2. Improving the efficiency of the software for multivariable state space system identification based on subspace techniques, as shown in Section 3.

This work fits nicely into the work of the SISTA research unit. Indeed, substantial research has been conducted on the design of new subspace identification algorithms, both for the time and frequency identification problem. This work resulted in a Matlab implementation of the algorithms and their application to many practical and industrial examples. However, a more efficient implementation of the algorithms, where the full structure of the problem is exploited, is certainly needed for solving industrial large-scale problems. The author's work significantly contributes to this implementation project. In particular, high performance implementatios open doors to new applications, such as the identification of high-dimensional mechanical structures and complicated chemical plants, and the use of the algorithms in fastly sampled model predictive based control schemes.

All of these topics have been heavily explored within the context of the interuniversity pole of attraction IUAP/PAI 17 (Modeling and Control of Dynamic Systems) and its successor IUAP/PAI IV/2 (Modeling, Identification, Simulation and Control of Complex Systems).

2 Improving the SLICOT library for computer-aided control systems analysis and design

The research and development of reliable tools for computer-aided control system analysis and design (CACSD) is heavily dependent on the availability of high quality basic software [1, 2, 3]. Such a software is well represented by the SLICOT Fortran 77 library [4, 5], a product of WGS, commercialized till now by the Numerical Algorithms Group (NAG), Oxford, UK. Recently, WGS and NAG agreed to make SLICOT freely available, by replacing the NAG mathematical library-based computations by equivalent calculations performed by BLAS [7] and LAPACK [8] routines, which belong to the *public-domain* software, and represent the state-of-the-art in numerical linear algebra. Research in this area still continues and new numerical algorithms are being developed that will fill in the gaps of the SLICOT library. A thematic network for numerics in control, called NICONET, was set up, for extending the cooperation to a European level and provide a wider basis of users and software developers for the SLICOT library.

It was anticipated that an improved version of the SLICOT library can have several advantages over other tools, such as the Matlab toolboxes. The reasons for this are, for instance:

- Matlab calculations cannot easily exploit the structure of the problem, while this can be achieved by SLICOT routines, which are designed to be *structure preserving* or *structure exploiting* and make full use of the flexibility offered by BLAS and LAPACK components.
- By ignoring the problem structure, Matlab cannot solve some complex industrial large-scale problems, due to memory limitations and execution time required.
- Matlab is actually an interpretative language, and it cannot beat C or Fortran codes in efficiency.
- Matlab kernel computations are still based on the EISPACK and LINPACK libraries, developed in 1976–1979, while the new SLICOT library is based on newer, high level BLAS and LAPACK software, taking advantage of the recent developments in numerical linear algebra and scientific computations. In particular, the same routines can be easily modified to run on parallel processors by an appropriate tuning of the software and the use of the recently developed ScaLAPACK [9]. Parallelisation clearly opens new perspectives for solving large scale problems which frequently arise in industrial practice.

The truth of such expected results has been proven by the numerical evidence acquired during this research stay, partly illustrated in the following subsections, which show that SLICOT can outperform equivalent Matlab calculations in terms of efficiency, numerical accuracy and memory, up to several orders of magnitude.

2.1 New features of the SLICOT library

The first step of the approach for improving the SLICOT library was to identify the NAG routine calls and replace them by equivalent BLAS/LAPACK routine calls. However, this could lead to a poor efficiency which does not make use of the peak performance on some modern computer architectures, because the previous implementation would be translated to one based on Level 1 or Level 2 BLAS. Therefore, we tried to upgrade the low-level computational structure of the old SLICOT, and use Level 3 BLAS and LAPACK block algorithms as much as possible. In addition, the codes have been carefully scrutinized, in order to detect the code portions that can be equivalently formulated in terms of BLAS calls, and replace them accordingly. This offers not only better efficiency, but also improved clarity of the source code, and, of course, better compactness. The robustness was improved, whenever possible, by replacing less stable calculations by mathematically equivalent, numerically stable ones.

Many routines have been significantly modified, and generally employ the LAPACK block algorithms and BLAS 3. Some algorithms have been rearranged, and the new codes are shorter and clearer than the previous ones. Several new routines for basic mathematical computations, which have no LAPACK correspondents, have been added. For instance, there are three routines which compute the LU factorization, estimate the reciprocal condition number, and solve a set of linear systems, respectively, with A a complex upper Hessenberg matrix, taking advantage of the Hessenberg structure.

Table 1: Size of the working array in the previous and new SLICOT versions.

Routine	Workspace	
	Previous version	New version
FB01QD	$(n+p)(m+n+p)$	$\max(n(p+n)+2p, n(n+m+2), 3p)$
FB01RD	$(n+p)(m+n+p)$	$\max(n(p+n+1), n(p+n)+2p, n(n+m+2), 3p)$
FB01SD	$(m+n+p)(m+n+p)$	$\max(n(n+2m)+3m, (n+1)(p+n)+2n, 3n)$
FB01TD	$(m+n+p)(m+n+2)$	$\max(n(n+2m)+3m, (n+1)(p+n)+n+\max(n-1, m+1), 3n)$
FB01VD	$n(m+n+\ell)+2\ell$	$\max(\ell n+3\ell, n^2, nm)$

For enhancing the collection, all routines without a clear modularity or functionality role have been removed, by including the calculations they performed in the corresponding calling subroutines. Note that many such routines have been written with the only purpose to partition the unique working array (of each type), allowed by the implementation standards, into several working arrays (not seen by the user), and thus make programming easier. The new implementation achieves the requirement of having a unique working array by using the LAPACK-style of partitioning. Namely, an array is divided into several parts, if needed, by introducing mnemonic index variables that are used as pointers to its parts. Chapter T (Transformation Routines) now contains 30 routines, which cover (and sligthly extend) the functionality of the previous 41 routines. So, 11 routines have been removed for reasons mentioned above. Similarly, Chapter F (Filtering) now includes only five routines, compared to nine routines in the previous implementation. But four of the old routines had a very close functionality to other four newer routines. In the new implementation, the functionality of the four older routines is completely covered by the new ones.

About two thirds of the SLICOT library have been re-written, to improve the codes and adapt them to the new implementation standards. The simplest approach to update the routines would lead to BLAS 2-based routines. The approach taken was to use as much as possible BLAS 3 calls and LAPACK block algorithms, while exploiting the special structure whenever possible (e.g., the structure of the pre-arrays associated to Kalman filter problems, for the Chapter F routines). The computations have been reorganized in this sense. We will illustrate the advantages of this approach for the routines in Chapter F. The main results are:

1. Improving the efficiency and modularity. Six new mathematical routines have been written (mentioned below), which could also be useful in other calculations.

2. Reducing the size of the working array, as shown in Table 1.

The new mathematical routines and their functions are:

`MB01RD` : Compute the matrix formula

$$\bar{R} = \alpha R + \beta \mathtt{op(A)} X \mathtt{op(A)}^T,$$

where R, X, and $\bar{R}$ are symmetric matrices, A is a general matrix, and $\mathtt{op(A)} = A$, or $\mathtt{op(A)} = A^T$. The symmetry is fully exploited.

`MB02OD` : Solve one of the matrix equations

$$\mathtt{op(A)} X = \alpha B, \qquad \text{or} \qquad X \mathtt{op(A)} = \alpha B,$$

where X and B are m-by-n matrices, A is a triangular matrix, and also estimate the reciprocal of the condition number of A. The triangular structure is exploited.

`MB04ID` : Compute a QR factorization of an n-by-m matrix A ($A = QR$), having a p-by-$\min(p, m)$ zero triangle in the lower left-hand side corner, and optionally apply the transformations to an n-by-ℓ matrix B (from the left). The special structure is exploited.

`MB04JD` : Compute an LQ factorization of an n-by-m matrix A ($A = LQ$), having a $\min(n, p)$-by-p zero triangle in the upper right-hand side corner, and optionally apply the transformations to an ℓ-by-m matrix B (from the right). The special structure is exploited.

`MB04KD` : Calculate a QR factorization of the first block column and apply the orthogonal transformations (from the left) also to the second block column of a structured matrix, as follows

$$Q^T \begin{bmatrix} R & 0 \\ A & B \end{bmatrix} = \begin{bmatrix} \bar{R} & C \\ 0 & D \end{bmatrix},$$

where R and $\bar{R}$ are upper triangular. The matrix A can be full or upper trapezoidal/triangular. The special structure is exploited.

`MB04LD` : Calculate an LQ factorization of the first block row and apply the orthogonal transformations (from the right) also to the second block row of a structured matrix, as follows

$$\begin{bmatrix} L & A \\ 0 & B \end{bmatrix} Q = \begin{bmatrix} \bar{L} & 0 \\ C & D \end{bmatrix},$$

where L and $\bar{L}$ are lower triangular. The matrix A can be full or lower trapezoidal/triangular. The special structure is exploited.

Each of these routines is called several times from the Chapter F codes.

The routines have been checked using adaptations of the previous test programs, data and results. Part of the routines (those from Chapters D, F, and partially, M) have also been tested exhaustively, by using the NAGWare gateway generator [6], which facilitates the integration of SLICOT in Matlab. New Matlab tests programs have been written, which call the Fortran routines. Equivalent computations have been performed in Matlab, so that it was possible to compare the efficiency and accuracy of Fortran and Matlab implementations. The routines already finished have been posted on the WGS ftp site,

`wgs.esat.kuleuven.ac.be`

(directory `pub/WGS/SLICOT/` and its subdirectories) in compressed (gzipped) tar files. On line `.html` documentation files are also provided there. It is possible to browse through the documentation using, e.g. Netscape, from the WGS www address

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

after clicking on the `ftp site` link). The SLICOT index (for the SLICOT part already implemented) is operational. Each functional "module" can be copied to user's current directory, by clicking on an appropriate location in the `.html` image (near the end). A "module" is a compressed (gzipped) tar file, which includes the following files: source code

Table 2: Relative residual errors for first version of MB02OD and Matlab results.

m	n	MB02OD	Matlab	MB02OD - Matlab
16	4	1.0496e-16	2.6562e-16	2.0310e-16
32	8	3.7981e-16	2.7822e-15	9.9620e-16
64	16	6.0237e-16	8.4907e-15	1.1551e-14
128	32	7.6825e-14	1.6183e-12	1.1396e-12
256	64	7.3687e+01	2.8041e-09	2.1225e+06

Table 3: Relative residual errors for the last example, with tolerance `EPS*norm(A)`.

m	n	MB02OD	Matlab	MB02OD - Matlab
256	64	2.5318e-10	2.8041e-09	3.2251e-08

for the main routine and its test program, test data, execution results, the associated `.html` file, as well as the source code for all called SLICOT routines. This involves wasting some disk space on the WGS ftp site (by certain duplications), but can be convenient for a user needing an isolate function. There is also a file, called `slicot.tar.gz`, in the directory `/pub/WGS/SLICOT/`, which contains all files currently adapted to the new standards. The tree structure created after applying `gzip -d slicot.tar` and `tar xvf slicot.tar` is:

```
./slicot/        - for routine source files;
./slicot/doc/    - for html files;
./slicot/tests/  - for test programs/data/results files.
```

2.2 Performance results

The tests revealed that the default tolerances provided by the previous SLICOT release to be used for singularity decisions are sometimes **not appropriate** for large problems. For instance, trying to solve a triangular set of linear systems, defined by $AX = B$, with A m-by-m, and B m-by-n matrices, and

```
A = triu(rand(M,M)) + eye(M)/2,
```

the relative residual errors in Table 2 have been obtained.

While the first four examples show that SLICOT accuracy was better, the last one shows no accuracy at all. The problem for this last example is ill-conditioned, and, due to the default tolerance `M*M*EPS*norm(A)`, where `EPS` is the relative machine precision, the error indicator `info` was set to a non-zero value, and the solution X was actually not computed. This tolerance is too large, since m (alias `M`) is large. But matrix A in the last example was not so ill-conditioned that a meaningful solution couldn't be obtained. Actually, the estimated reciprocal condition number was `rcond = 4.6346e-11`, which is not too small, but, however, less than the default tolerance `M*M*EPS*norm(A)`.

Using instead the tolerance `EPS*norm(A)`, the last row of the table becomes as given in Table 3, showing again improved accuracy compared to Matlab.

The impossibility to solve enough well-conditioned problems was indicating that the previously used test is not good in general. Removing the factor `M*M`, but preserving `norm(A)` was enough for the problem discussed above, but the difficulty reappeared for a larger problem (even if `norm(A)` was not too large). This suggested to remove `norm(A)` too. Of course, it is necessary to include `norm(A)`, or other estimate of the largest singular value of the matrix, for rank decision problems.

The numerical evidence from additional tests performed with the triangular solver has led to the following **main conclusions**:

1. It could be wrong to use a default tolerance larger than EPS, because this could prevent a "reasonable" solution being computed.

2. Meaningful results can be obtained even if `RCOND < EPS`; indeed, the relative errors for a problem with $m = 512$ have been of the order 10^{-2}, so at least the first digit of the result was good. This could be better than having no solution of all. Of course, the case `RCOND < EPS` should be detected by the error/warning indicator `info`, to warn the user about the possible loss of precision.

3. The SLICOT gateway can be more than 3–5 times faster than Matlab.

4. LAPACK condition estimator for triangular matrices proved to be a little bit less accurate in these tests than the LINPACK one, used by the Matlab function rcond(.).

5. The use of the Matlab `inv(.)` function in expressions like `inv(A)*B` should be definitely avoided, because serious loss of accuracy could appear; the use of the expression `A\B` gave the same results as the SLICOT routine. Moreover, `inv` is also more expensive than `A\B`. It is worth mentioning that many Matlab codes call `inv(.)`, especially if one has to compute the solution of $XA = B$.

Concerning the condition estimators, the results were surprizing, since it could be expected that LAPACK estimators are more accurate than LINPACK ones. The difference is not due to the fact that LINPACK uses the 1-norm, while the infinity-norm was used in the tests, to get error bounds, as suggested in the LAPACK manual [8]. Indeed, repeating the calculations for the 1-norm, the results were about the same. It should be mentioned, however, that both estimators gave results of the same order of magnitude; it is only the difference to the "true" condition number, computed in the tests by the Matlab `cond(.)` function, that indicated LINPACK as a more accurate estimator.

The conclusions based on the tests performed led to the following proposals:

1. Use `EPS` as default tolerance for solving linear systems instead of `M*M*EPS` (or `M*M*EPS*norm(A)`). LAPACK expert drivers for linear systems also use `EPS`, and no tolerance can be set by the user.

2. Compute the solution even if the condition estimate clearly indicates nearly singularity, but warn the user (setting `INFO` to a non-zero value) that the results could be highly inaccurate. The kernel Matlab codes also do so. Warnings are produced by `inv(A)*B` if A is too ill-conditioned, but no warnings are given when using `A\B`.

The tables below contain a summary of the test results for some SLICOT routines, in comparison with equivalent Matlab calculations. These tables illustrate the possible benefits of using SLICOT, and the importance of improving it, as proposed in the undergoing NICONET project.

The results reported have been obtained by calling from Matlab the gateways produced by the NAGWare Gateway generator for the corresponding SLICOT codes. These results show that SLICOT routines can outperform Matlab calculations. While the accuracy is practically the same, or better, the gain in efficiency by calling SLICOT routines can be significant for large problems. Note that the figures have been obtained by timing strictly the equivalent computations. Even better efficiency is to be expected by calling the SLICOT Fortran routines only (not through the gateway), and similar accuracy/efficiency improvements are possible for other SLICOT computations, especially for large problems, due to the incorporated calls to upper level BLAS routines, and performant LAPACK blocked (and unblocked) algorithms.

The following tables indicate the accuracy (measured by either the relative errors or relative residuals, when available), and the time spent exclusively for equivalent SLICOT and Matlab computations, for some routines in the Chapters D, F, and M of the library.

The **main conclusions** of the tests are:

1. The SLICOT gateways can usually be several times faster than Matlab.
2. The accuracy of SLICOT routines is at least as good as, or better than, that for Matlab calculations.
3. The memory is much better used, and less memory is required by SLICOT routines for equivalent calculations, because the problem structure is fully exploited.

Summary of results

(i) Tables 4 and 5 display comparative results for SLICOT `fft/ifft` routines (`DG01MD`, for complex sequences, and `DG01ND`, for real sequences), and Matlab `fft/ifft` functions, for randomly generated sequences `X` of length n. Besides better efficiency, the accuracy of SLICOT routines shows an improvement which can be two orders of magnitude. The accuracy was measured by computing

```
norm(X - ifft( fft( X ) ))/norm(X);
```

(which should theoretically be zero), and similarly for the SLICOT gateways. Note that `DG01ND` is 2–4 times faster than Matlab.

(ii) Tables 6–8 display comparative results for SLICOT linear systems solver `MB02OD` for triangular matrices, which includes a condition number estimator, and Matlab functions `A\B` and `rcond(.)`. Triangular sets of linear systems, defined by $AX = B$, with A m-by-m, and B m-by-n matrices were solved, where

```
A = triu(rand(M,M)) + eye(M)/2;   X = rand(M,N);   B = A*X;
```

Timing results, and relative error and residual norms of `MB02OD` and Matlab results are given. Note that the relative errors are the "truly" errors, because A and X have first been chosen and then used to compute $B = AX$, so the true results were known. For large matrices, SLICOT gateway can be more than 5 times faster than Matlab. Note also that the LINPACK reciprocal condition estimator (used by Matlab function `rcond(.)`) is somewhat more accurate than the LAPACK estimator.

Tables 9 and 10 give, for comparison purposes, some results obtained using Matlab `inv` function. Note that the SLICOT gateway can be more than 10 times faster than Matlab (for large matrices), and it obtains better relative error and residual norms.

Table 4: Comparison between DG01MD and Matlab results.

n	Time		Relative error	
	DG01MD	Matlab	DG01MD	Matlab
256	0	0	3.2138e-16	1.5922e-15
512	0	0	5.0420e-16	2.8591e-15
1024	0	0	7.5435e-16	8.1396e-15
2048	0.01	0.02	1.4554e-15	6.7684e-15
4096	0.02	0.03	1.3566e-15	2.4919e-14
8192	0.03	0.06	2.3314e-15	2.0229e-14
16384	0.08	0.13	2.3231e-15	1.0993e-13
32768	0.21	0.34	3.3539e-15	2.9552e-13
65536	0.80	1.22	5.0354e-15	4.7660e-13
131072	2.58	2.82	1.1206e-14	1.1097e-12
262144	5.74	6.78	1.7484e-14	1.5873e-12

Table 5: Comparison between DG01ND and Matlab results.

n	Time		Relative error	
	DG01ND	Matlab	DG01ND	Matlab
256	0	0	3.8330e-16	1.3000e-15
512	0	0	3.8718e-16	2.2666e-15
1024	0	0.01	6.4400e-16	6.5623e-15
2048	0	0.01	1.0566e-15	6.5603e-15
4096	0.01	0.02	1.4364e-15	1.9996e-14
8192	0.02	0.05	1.9524e-15	2.1254e-14
16384	0.03	0.11	2.4556e-15	8.2107e-14
32768	0.07	0.27	2.8141e-15	2.4582e-13
65536	0.23	0.88	4.2154e-15	4.2554e-13
131072	0.99	2.39	8.2977e-15	9.4844e-13

Table 6: Comparison between MB02OD and Matlab results (timing).

			Time	
m	n	info	MB02OD	Matlab
16	4	0	0	0
32	8	0	0.01	0
64	16	0	0.01	0.02
128	32	0	0.03	0.12
256	64	0	0.17	0.99
512	128	1	1.69	9.42

Table 7: Comparison between `MB02OD` and Matlab results (condition estimate).

`rcond`		`1/cond(A)`	`1/cond(A) - rcond`		Error bound
`MB02OD`	Matlab		`MB02OD`	Matlab	
6.3752e-03	1.6058e-02	2.3266e-02	1.6891e-02	7.2075e-03	3.4829e-14
2.3475e-03	5.7554e-03	7.8234e-03	5.4759e-03	2.0680e-03	9.4587e-14
8.2433e-05	1.8197e-04	2.6663e-04	1.8420e-04	8.4655e-05	2.6936e-12
1.6725e-06	5.1484e-06	7.5425e-06	5.8701e-06	2.3941e-06	1.3276e-10
5.9760e-11	1.2778e-10	2.3305e-10	1.7329e-10	1.0527e-10	3.7156e-06
6.7864e-17	3.3953e-16	3.6639e-16	2.9853e-16	2.6864e-17	1.0000e+00

Table 8: Comparison between `MB02OD` and Matlab results. Relative error and residual norms and relative error norms `MB02OD`/Matlab.

Relative error		Relative residual		`MB02OD - Matlab`
`MB02OD`	Matlab	`MB02OD`	Matlab	
1.8085e-15	1.8085e-15	4.5182e-17	4.5182e-17	0
2.5745e-15	2.5745e-15	6.4082e-17	6.4082e-17	0
4.7463e-14	4.7463e-14	6.7368e-17	6.7368e-17	0
1.3926e-12	1.3926e-12	5.8022e-17	5.8022e-17	0
1.9740e-08	1.9740e-08	6.5680e-17	6.5680e-17	0
1.1370e-02	1.1370e-02	6.6017e-17	6.6017e-17	0

Many Matlab codes call `inv(.)` instead `A\B`, especially if one has to compute the solution of $XA = B$. Most Matlab users are not aware about such numerical details, so the chances for doing unreliable calculations can be quite high. This is not the case when using SLICOT routines.

(iii) Tables 11–15 display comparative results for SLICOT Kalman filter routines (Chapter F), and equivalent Matlab calculations. The Chapter F routines update some covariance/information square root factors after one iteration of the Kalman filter algorithm; special cases of system matrices in lower observer or upper controller Hessenberg forms are dealt with by separate routines. The updates are obtained by applying either LQ or QR factorizations to some structured "pre-arrays". For the routines updating co-

Table 9: Comparison between `MB02OD` and Matlab `inv(.)` results (timing).

			Time	
m	n	`info`	`MB02OD`	Matlab
16	4	0	0	0
32	8	0	0	0.01
64	16	0	0	0.01
128	32	0	0.02	0.11
256	64	0	0.20	1.60
512	128	1	1.79	20.07

Table 10: Comparison between `MB02OD` and Matlab `inv(.)` results. Relative error and residual norms and relative error norms `MB02OD`/Matlab.

Relative error		Relative residual		`MB02OD - Matlab`
`MB02OD`	Matlab	`MB02OD`	Matlab	
7.5518e-16	1.5336e-15	4.9095e-17	1.4728e-16	1.6378e-16
4.0113e-15	4.2807e-15	4.7629e-17	3.0959e-16	4.0112e-16
9.1881e-14	1.0558e-13	5.2511e-17	1.7722e-15	5.4398e-15
1.0142e-11	1.0171e-11	5.9074e-17	9.9504e-14	1.7766e-13
1.7096e-08	1.8931e-08	6.9761e-17	1.0952e-10	1.9217e-10
1.6445e-02	1.9956e-02	6.2213e-17	1.1722e-04	8.8566e-05

Table 11: Comparison between `FB01QD` and Matlab results.

Dimensions				Time		Relative error norms	
n	m	p	`info`	`FB01QD`	Matlab	`FB01QD`	Matlab
16	4	8	0	0	0	2.0017e-16	2.4851e-16
32	8	16	0	0.01	0.01	2.4928e-16	2.7110e-16
64	16	32	0	0.05	0.05	3.9896e-16	1.3642e-15
128	32	64	0	0.38	0.34	5.2037e-16	4.8178e-16
256	64	128	0	4.01	2.87	8.9377e-16	1.5683e-15

variance square roots, we have checked that the input data and output results are related by an orthogonal transformation. As this was not possible for the routines updating information square roots, for those routines we have just compared SLICOT and Matlab results. The same has been done for the routine `FB01VD`, which directly updates the Riccati difference equation and the Kalman filter gain matrices. Except for `FB01QD`, SLICOT codes were about 2–4 times faster than Matlab calculations, at comparable or better accuracy.

For large dimensions, `FB01QD` gateway proved to be about 1.4 times slower than Matlab code. This can be explained by the fact that `FB01QD` uses LQ factorization, which operates row-wise; in the Matlab code, we have transposed the pre-array and used (column-wise) QR factorization. Of course, if the same approach would be used in `FB01QD`, it would be faster than Matlab (this is actually the case for `FB01SD`), but such

Table 12: Comparison between `FB01RD` and Matlab results.

Dimensions				Time		Relative error norms	
n	m	p	`info`	`FB01RD`	Matlab	`FB01RD`	Matlab
16	4	8	0	0	0	1.5792e-16	2.2532e-16
32	8	16	0	0.01	0.01	3.6389e-16	4.0294e-16
64	16	32	0	0.02	0.05	3.4772e-16	5.3653e-16
128	32	64	0	0.20	0.34	3.0651e-16	9.4382e-16
256	64	128	0	1.73	2.87	3.9917e-16	1.0853e-15

Table 13: Comparison between `FB01SD` and Matlab results.

Dimensions				Time		Relative error norms
n	m	p	`info`	`FB01SD`	Matlab	`FB01SD - Matlab`
16	4	8	0	0.01	0	4.7443e-16
32	8	16	0	0.01	0.01	9.9886e-16
64	16	32	0	0.05	0.11	1.5488e-15
128	32	64	0	0.36	0.76	2.1013e-15
256	64	128	0	3.40	7.16	1.8436e-15

Table 14: Comparison between `FB01TD` and Matlab results.

Dimensions				Time		Relative error norms
n	m	p	`info`	`FB01TD`	Matlab	`FB01TD - Matlab`
16	4	8	0	0	0.01	1.8073e-16
32	8	16	0	0.01	0.01	3.0954e-16
64	16	32	0	0.03	0.11	2.2506e-16
128	32	64	0	0.17	0.76	1.8628e-15
256	64	128	0	1.48	6.48	1.6709e-15

Table 15: Comparison between `FB01VD` and Matlab results.

Dimensions				Time		Relative error norms
n	m	p	`info`	`FB01VD`	Matlab	`FB01VD - Matlab`
16	4	8	0	0	0	1.1639e-17
32	8	16	0	0.01	0	4.7562e-17
64	16	32	0	0.05	0.05	1.8170e-16
128	32	64	0	0.41	0.38	1.0548e-15
256	64	128	0	3.37	8.09	8.9321e-15

an approach would double the working space (for storing the transpose). We propose to include an alternative routine in the library, to work efficiently on the transpose problem, provided enough working space is available.

3 Improving the software for subspace-based multivariable state space system identification

Computer-aided control system design is usually based on state space models. Surprisingly enough, multivariable state space system identification has only recently become a topic of intense research. In contrast with classical input-output identification approaches, the newly proposed subspace techniques essentially find a state sequence, and then determine the system matrices by solving some (total) least squares problems [20]. These techniques have promising advantages over the classical identification approaches. One advantage is that there is no need for parameterizations, which are notoriously difficult to choose or analyze, or could lead to ill-conditioned problems. Another advantage is that robust numerical linear algebra techniques, like QR decomposition and singular value decomposition, can be largely used.

Matlab codes for state space identification algorithms have been recently developed. For efficiency and accuracy reasons, it is necessary to implement some algorithms in Fortran, using the state-of-the-art "public domain" linear algebra package LAPACK (the first public release in 1992, and the second release in 1994). Moreover, this will allow to exploit the potential parallelism of many modern computer architectures.

3.1 The approach followed

The **main objective** of our research was to analyze and develop high-performance algorithms and associated software for subspace-based system identification. Some **derived objectives** were the following ones:

- efficient and flexible implementation of the recently proposed subspace-based algorithms for system identification;
- exploiting the problem structure and the capabilities of the newly developed high-performance numerical linear algebra algorithms;
- numerical investigation of subspace-based algorithms.

Recent algorithmic developments and advances in numerical linear algebra have been taken into account. The impact of the new, high-performance computer architectures on the identification techniques has been considered.

The investigated identification algorithms and software compute discrete, linear, and time-invariant state space models from input-output (I/O) data sequences. Both the MOESP (Multivariable Output Error state SPace) approach [15, 16, 17, 18] and the N4SID (Numerical algorithms for Subspace State Space System IDentification) approach [12, 11, 13, 14], are considered. All routines allow to handle multiple I/O data sequences, each sequence being collected by a possibly independent identification experiment with the system to be identified. Furthermore, care has been taken to minimize the memory use so that processing of large I/O sequences and the identification of systems of large order becomes possible. The key numerical step in the algorithms belonging to the MOESP class is the approximation of an extended observability matrix, while that in the algorithms belonging to the N4SID class is the approximation of the state sequences by "intersecting" past and future I/O data matrices. The main algorithms considered are:

- The ordinary MOESP scheme, producing consistent and statistically efficient estimates of the state space quadruple of the deterministic part, for v—the additive disturbance to the output variable—zero mean white noise, independent from the input.
- The ordinary MOESP scheme extended with instrumental variables based on past input quantities, producing consistent estimates when v is zero mean, but of arbitrary statistical color.
- The ordinary MOESP scheme extended with instrumental variables based on past input and past output quantities, producing consistent and statistically efficient estimates when v is generated by an innovation model. Optionally, a Kalman predictor is also computed.
- The ordinary MOESP scheme extended with instrumental variables based on reconstructed state and/or past input quantities, producing consistent estimates when v is zero mean, but of arbitrary statistical color.
- The N4SID scheme, producing consistent and statistically efficient estimates when v is generated by an innovation model. Optionally, a Kalman predictor is also computed.

In addition, there are several auxiliary routines which implement some kernel linear algebra computations, or solve related identification problems: identification of a limited set of Markov parameters, shifting the I/O data sequences to reduce the state dimension when there are deadtimes in the different input channels, estimation of the initial conditions, identification of systems operating in closed loop.

There is a serious difference between a theoretical statement of an algorithm and an efficient and reliable implementation. For instance, the N4SID algorithm makes in theory use of the inverse of a part of the triangular factor of an RQ factorization of the Hankel-like matrix constructed from the input and output sequences. This is needed before computing the singular value decomposition that gives the system order and then the quadruple of system matrices. The Matlab implementations solve several standard least squares problems to determine the associated, so-called *oblique projection*. On the other hand, the new, LAPACK-based implementation only involves some orthogonal transformations for obtaining the various projections and residuals. Algorithmic details are given in [22, 23].

The **approach** followed was to use the components of LAPACK and BLAS (Basic Linear Algebra Subprograms) as much as possible. Preference has generally been given to the block variants and Level 3 BLAS codes, which are responsible for efficient use of parallelism. LAPACK is used in a constructive way. Some very large routines, like DGESVD, are not called, for reducing the object code volume. New LAPACK-style codes for special QR or singular value decompositions are used. Whenever possible, advantage is taken of the particular problem structure, in order to increase efficiency, and reduce the memory requirements. For instance, the calculations are organized such that singular value decompositions are required for triangular matrices only, and a dedicated, very compact and efficient routine is used for that purpose. Moreover, both in theory and in the Matlab implementations, the computation of the system matrices involves a very large matrix. Fortunately, this matrix can be brought to a special block-triangular structure. A structure-exploiting QR row compression scheme has been devised, which significantly reduces the storage requirements and computational effort.

Flexibility means that various algorithmic paths and options are offered. As already shown, the main algorithms considered belong to either the MOESP scheme (with variants) or to the N4SID scheme. Optionally, a Kalman estimator can also be computed, when appropriate. An integration of the past input and past output MOESP scheme and N4SID approach (for covariance calculations) was considered too.

Flexibility also allows to perform the computations with minimal memory requirements, but without sacrificing the speed, when enough memory is available. For instance, there is an option for sequential calculation of the triangular factor in the QR factorization of the Hankel-like matrix. The input-output data can be entered in batches (related or not). Moreover, when there is not enough working memory for processing a given data batch, the codes automatically perform an inner sequential processing of each batch, to accommodate with the available memory space. (A reasonable lower limit of the memory space is however required.) This way, processing of large input/output sequences, and the identification of systems of large order becomes possible, even on computers with limited resources, like personal computers. No provisions for using the displacement rank techniques (based on the block-Hankel structure) are provided in the current implementations.

The performances of the investigated algorithms in solving some simulated or real-world problems have been analyzed, and the results obtained using various algorithms (including their Matlab versions) extensively compared. (The comparisons refer primarily to efficiency, accuracy and reliability.) The associated software is able to efficiently exploit the capabilities of several high-performance computer architectures.

Tables 16–27 give some performance measures for several subspace identification, as well classical identification algorithms, for some industrial data sets. Both Matlab and Fortran codes are included in the comparison. The codes (Matlab functions) referred to as `subid`, `com_alt`, and `com_stat` are essentially those provided on the diskette accompanying the book [14]. The difference is that here each computational function is timed individually, so that comparison of efficiency is readily obtained. The codes referred to as `pem` and `oe` are functions included in the Matlab identification toolbox [10], based on classical identification techniques. The codes referred to as `IMN4S` and `IMOEKG` are two of the Fortran implemented algorithms; the first one is based on the N4SID technique [12, 13, 14], while the second uses the past input/past output MOESP scheme [15].

Table 16 gives a summary description of the applications considered in our comparison, indicating the number of inputs m, number of outputs l, estimated system order n, number of data points taken for identification t_{ID}, and number of data points taken for validation t_{VAL}.

Table 17 shows the computation time in seconds spent by each code. It is clear that Fortran codes outperform the other ones. The fastest code is `IMOEKG`, closely followed by `IMN4S`, and then by other subspace codes. Generally, `IMOEKG` and `IMN4S` are about two times (or sometimes more) faster than the Matlab competitors. It is worth mentioning that the results for the classical identification algorithms have been obtained using the output of the `subid` code as initial guess. Nevertheless, `pem` and `oe` could sometimes be 20 or even 30 times less efficient as the best subspace algorithms.

The raw data have been preprocessed when needed, to ensure the input and output mean values close to zero. The time horizon has been divided into two parts: the identification part, and the validation part. Both prediction and simulation relative errors are included. The simulation errors have been computed using the estimated system matrices A, B, C,

Table 16: Summary description of applications.

#	Application	m	l	n	t_{ID}	t_{VAL}
1	Glass tubes	2	2	8	900	426
2	Labo Dryer	1	1	4	500	500
3	Glass Oven	3	6	5	900	347
4	Mechanical flutter	1	1	6	1024	1024
5	Flexible robot arm	1	1	4	800	224
6	Evaporator	3	3	4	3300	3005
7	CD player arm	2	2	8	1024	1024
8	Ball and beam	1	1	2	999	999
9	Wall temperature	2	1	3	1200	480

Table 17: Computation time in seconds.

#	`subid`	`com_alt`	`com_stat`	`IMN4S`	`IMOEKG`	`pem`	`oe`
1	2.89	2.36	2.56	1.52	1.39	24.65	16.85
2	0.52	0.29	0.28	0.17	0.16	0.42	0.17
3	3.34	2.99	3.27	1.91	1.80	32.92	36.82
4	0.97	0.81	0.80	0.48	0.46	0.66	4.39
5	0.68	0.60	0.61	0.37	0.36	1.40	0.86
6	5.76	5.31	5.39	3.00	2.97	51.13	53.06
7	1.94	1.57	1.65	0.99	0.95	30.26	8.24
8	0.78	0.73	0.75	0.45	0.43	0.20	0.23
9	2.00	1.83	1.92	1.11	1.08	4.39	3.48

Table 18: Percentual prediction and simulation relative errors for Glass tubes application.

#	Function	Prediction errors		Simulation errors	
		Identification	Validation	Identification	Validation
1	`subid`	9.9	7.2	35.6	15.8
2	`com_alt`	11.2	8.3	57.1	36.1
3	`com_stat`	9.9	7.2	35.8	17.0
4	`IMN4S`	15.1	10.8	97.8	71.2
5	`IMOEKG`	11.6	8.6	53.0	32.8
6	`pem`	9.6	7.1	35.3	17.2
7	`oe`	35.4	15.5	35.4	15.5

Table 19: Percentual prediction and simulation relative errors for Labo Dryer application.

#	Function	Prediction errors		Simulation errors	
		Identification	Validation	Identification	Validation
1	`subid`	3.3	3.2	8.2	7.5
2	`com_alt`	3.3	3.2	8.2	7.4
3	`com_stat`	3.3	3.2	8.2	7.4
4	`IMN4S`	3.3	3.2	8.4	7.7
5	`IMOEKG`	3.3	3.1	8.3	7.5
6	`pem`	3.3	3.1	8.8	8.2
7	`oe`	8.3	7.5	8.3	7.5

and D, as well as a least squares estimate of the initial state of the system, determined based on the given input and output data on a limited time horizon (not larger than $3n$). The prediction errors have been computed using the estimated system matrices A, B, C, D, and K (the Kalman filter gain matrix), and the estimate of the initial state, determined as specified above. It can be seen that the simulation errors are always worse than the prediction errors, which proves that the Kalman filter matrix was well estimated.

Tables 18–27 show the percentual prediction and simulation relative errors for the applications considered. The most accurate algorithm proven to be `subid`. The classical identification algorithms (especially `pem`) have usually achieved the same (or even slightly better) accuracy, but sometimes the convergence was not attained in a reasonable number of iterations. (It should be noted that generally `pem` and `oe` spent significantly more time than the subspace-based algorithms.) The other two Matlab codes were not enough robust. The Fortran codes, and especially `IMOEKG`, have frequently obtained a similar accuracy as `subid`, with some exceptions, which deserve a more detailed numerical investigation. It is our firm belief that Fortran codes could outperform the Matlab codes also regarding accuracy, but this assertion should be supported by numerical evidence.

4 Conclusions

New results on performance of some recently developed algorithms and associated software for control system analysis and design, as well as for multivariable state space system identification based on subspace techniques have been presented in this report.

Table 20: Percentual prediction and simulation relative errors for Glass Oven application.

#	Function	Prediction errors		Simulation errors	
		Identification	Validation	Identification	Validation
1	`subid`	10.3	7.7	39.6	32.9
2	`com_alt`	10.2	7.7	39.7	32.7
3	`com_stat`	10.3	7.6	39.8	34.0
4	`IMN4S`	28.4	17.8	87.0	53.7
5	`IMOEKG`	51.5	32.8	90.6	68.7
6	`pem`	100.0	100.0	100.0	100.0
7	`oe`	100.0	100.0	100.0	100.0

Table 21: Percentual prediction and simulation relative errors for Mechanical flutter application.

#	Function	Prediction errors		Simulation errors	
		Identification	Validation	Identification	Validation
1	`subid`	1.4	1.4	26.6	26.6
2	`com_alt`	100.0	100.0	100.0	100.0
3	`com_stat`	100.0	100.0	100.0	100.0
4	`IMN4S`	100.0	100.0	100.0	100.0
5	`IMOEKG`	100.0	100.0	100.0	100.0
6	`pem`	1.2	1.2	27.1	27.1
7	`oe`	16.0	16.0	16.0	16.0

Table 22: Percentual prediction and simulation relative errors for Flexible robot arm application.

#	Function	Prediction errors		Simulation errors	
		Identification	Validation	Identification	Validation
1	`subid`	1.2	0.7	4.5	2.4
2	`com_alt`	7.8	4.3	100.0	100.0
3	`com_stat`	100.0	100.0	100.0	100.0
4	`IMN4S`	100.0	100.0	100.0	100.0
5	`IMOEKG`	100.0	100.0	100.0	100.0
6	`pem`	0.9	0.5	6.1	2.6
7	`oe`	6.4	3.6	6.4	3.6

Table 23: Percentual prediction and simulation relative errors for Flexible robot arm application ($n = 5$).

#	Function	Prediction errors		Simulation errors	
		Identification	Validation	Identification	Validation
1	subid	1.2	0.7	4.2	2.2
2	com_alt	10.5	4.9	100.0	100.0
3	com_stat	100.0	100.0	100.0	100.0
4	IMOEKG	73.7	25.7	93.6	33.7
6	pem	1.1	0.6	5.7	3.4
7	oe	6.4	3.7	6.4	3.7

Table 24: Percentual prediction and simulation relative errors for Evaporator application.

#	Function	Prediction errors		Simulation errors	
		Identification	Validation	Identification	Validation
1	subid	20.4	15.9	37.6	37.1
2	com_alt	20.4	15.8	37.6	36.9
3	com_stat	20.4	15.9	37.5	36.9
4	IMN4S	22.3	18.7	40.4	39.9
5	IMOEKG	20.4	15.9	37.6	37.0
6	pem	19.9	15.6	39.3	37.5
7	oe	33.5	36.2	33.5	36.2

Table 25: Percentual prediction and simulation relative errors for CD player arm application.

#	Function	Prediction errors		Simulation errors	
		Identification	Validation	Identification	Validation
1	subid	10.7	10.7	16.3	17.2
2	com_alt	100.0	100.0	100.0	100.0
3	com_stat	100.0	100.0	100.0	100.0
4	IMN4S	100.0	100.0	100.0	100.0
5	IMOEKG	100.0	100.0	100.0	100.0
6	pem	10.7	10.7	16.3	17.1
7	oe	15.8	16.6	15.8	16.6

Table 26: Percentual prediction and simulation relative errors for Ball and beam application.

#	Function	Prediction errors		Simulation errors	
		Identification	Validation	Identification	Validation
1	`subid`	36.5	36.5	53.9	53.9
2	`com_alt`	100.0	100.0	100.0	100.0
3	`com_stat`	44.0	44.0	100.0	100.0
4	`IMN4S`	100.0	100.0	100.0	100.0
5	`IMOEKG`	100.0	100.0	100.0	100.0
6	`pem`	36.2	36.2	62.9	62.9
7	`oe`	53.9	53.9	53.9	53.9

Table 27: Percentual prediction and simulation relative errors for Wall temperature application.

#	Function	Prediction errors		Simulation errors	
		Identification	Validation	Identification	Validation
1	`subid`	11.8	7.3	11.8	7.4
2	`com_alt`	37.0	22.6	40.5	25.0
3	`com_stat`	12.0	7.4	12.1	7.5
4	`IMN4S`	100.0	100.0	100.0	100.0
5	`IMOEKG`	35.9	21.9	40.0	24.6
6	`pem`	11.7	7.4	11.8	7.4
7	`oe`	11.7	7.4	11.7	7.4

The main results achieved can be summarized as:

1. Improvements of the efficiency, reliability, and accuracy of the new SLICOT library routines, as shown in Section 2.

2. Improvements of the efficiency of software for multivariable state space system identification based on subspace techniques, as shown in Section 3.

The new codes are usually at least two times (and frequently more) faster than equivalent computations performed in Matlab. The accuracy is generally the same, or better, except for the subspace techniques for system identification, where, sometimes, it is worse than that achieved by the best, robust Matlab implementation. A potential reason for the lower accuracy of the Fortran codes `IMN4S` and `IMOEKG` could be the use of total least squares instead of least squares for the calculation of the matrices B and D. (The matrices A and C obtained by `subid` and Fortran routines agreed well.) Further analysis is needed to find the source of this potential inaccuracy. This will be the topic for a future work. Another topic could be the use of fast recursive identification algorithms via exploitation of the "displacement structure" [19].

Acknowledgment

The author expresses his gratitude to Professor Dr. Ir. Sabine Van Huffel, from Katholieke Universiteit Leuven, Belgium, for her generous intellectual and human support during the stay at Leuven, for many suggestions, as well as encouragements.

The author is also grateful to the staff of Signals, Identification, System Theory and Automation (SISTA) department, for making my research stay at Leuven so pleasant and instructive. Special thanks are due to Professor Dr. Ir. Joos Vandewalle, Professor Dr. Ir. Bart De Moor, and Ir. Bart Motmans. The author has also benefited by many discussions with, and suggestions, and help from Dr. Ir. Peter Van Overschee, Dr. Ir. Bart De Schutter, Ir. Wouter Favoreel, Ir. Leen Vanhamme, and many others. The support from the system managers of ESAT is also much acknowledged.

Last, but not least, the author acknowledges the financial support from the Belgian Office for Scientific, Technical and Cultural Affairs, which made this research stay possible.

References

[1] A. van den Boom, *CADCS developments in Europe*, Invited paper, 4th IFAC Symposium CADCS '88, Bejing, China, pp. 65–73, 1988.

[2] A. Laub, R. Patel, and P. Van Dooren, *Numerical linear algebra aspects of systems and control algorithms*, Numerical Linear Algebra Techniques for Systems and Control, pp. 1-35, (Eds. R. Patel, A. Laub, P. Van Dooren), IEEE Press, Piscataway NJ, 1993.

[3] P. Van Dooren, *Some numerical challenges in Control Theory*, in IMA Volumes in Mathematics and its Applications, "Linear Algebra for Control Theory", (B. Wyman, P. Van Dooren Eds.), pp. 177–189, Springer Verlag, 1993.

[4] A. van den Boom, A. Brown, A. Geurts, S. Hammarling, R. Kool, M. Vanbegin, P. Van Dooren, and S. Van Huffel, *SLICOT, a subroutine library in control and systems theory*, In Preprints 5th IFAC/IMACS Symp. CADCS'91, Swansea, UK, Pergamon Press, Oxford, 1991, pp. 89–94.

[5] Numerical Algorithms Group (NAG), *SLICOT library, Release 2.* NAG, Wilkinson House, Jordan Hill Road, Oxford, OX2 8DR, U.K., 1993 (updates Release 1 of May 1990).

[6] Numerical Algorithms Group (NAG), *NAGWare Gateway Generator, Release 2.0*, NAG Ltd, Wilkinson House, ISBN 1-85206-104-9, 1994.

[7] J.J. Dongarra, J. Du Croz, I.S. Duff, and S. Hammarling, *A set of level 3 basic linear algebra subprograms.* ACM Trans. Math. Software, vol. 16, pp. 1–28, 1990.

[8] 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 User's Guide*, 2nd ed., SIAM, Philadelphia, 1995.

[9] L. Blackford, J. Choi, A. Cleary, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R.C. Whaley, *Scalapack: A portable linear algebra library for distributed memory computers—design issues and performance.* In Proceedings of Supercomputing '96, Sponsored by ACM SIGARCH and IEEE Computer Society, 1996. (ACM Order Number: 415962, IEEE Computer Society Press, Order Number: RS00126. http://www.supercomp.org/sc96/proceedings/).

[10] L. Ljung, *System Identification Toolbox*, The MathWorks, Inc., Natick, MA, 1992.

[11] M. Moonen, B. De Moor, L. Vandenberghe, and J. Vandewalle, *On- and Off-line Identification of Linear State-space Models*, Int. J. Control, Vol.49, No.1, 1989, pp. 219–232.

[12] P. Van Overschee, and B. De Moor, *N4SID: Two Subspace Algorithms for the Identification of Combined Deterministic-stochastic Systems*, Automatica, Vol.30, No.1, 1994, pp. 75–93.

[13] P. Van Overschee, *Subspace Identification : Theory – Implementation – Applications*, Katholieke Universiteit Leuven, Ph. D. Thesis, 1995.

[14] P. Van Overschee, and B. De Moor, *Subspace identification for linear systems: Theory, Implementation, Applications*, Kluwer Academic Publishers, Boston, 1996.

[15] M. Verhaegen, *Identification of the Deterministic Part of MIMO State Space Models Given in Innovations Form from Input-output Data*, Automatica, Vol.30, No.1, 1994, pp. 61–74.

[16] M. Verhaegen, and P. Dewilde, *Subspace Model Identification. Part 1: The Output-error State- space Model Identification Class of Algorithms*, Int. J. Control, Vol.56, No.5, 1992, pp. 1187–1210.

[17] M. Verhaegen, *Subspace Model Identification. Part 3: Analysis of the Ordinary Output-error State-space Model Identification Algorithm*, Int. J. Control, Vol.58, No.3, 1993, pp. 555–586.

[18] M. Verhaegen, *State Space Model Identification Toolbox*, Delft University of Technology, Technical Report, November 1993.

[19] Y. M. Cho, G. Xu, and T. Kailath, *Fast Recursive Identification of State Space Models via Exploitation of Displacement Structure*, Automatica, Vol.30, No.1, 1994, pp. 45–59.

[20] S. Van Huffel, and J. Vandewalle, *The Total Least Squares Problem: Computational Aspects and Analysis*, SIAM, Philadelphia, PA, 1991.

[21] V. Sima, *Algorithms for Linear-quadratic Optimization*, in E. J. Taft and Z. Nashed (Eds.) Volume 200 of "Pure and Applied Mathematics: A Series of Monographs and Textbooks", MARCEL DEKKER, Inc., New York, 1996.

[22] V. Sima, *Algorithms and LAPACK-based Software for Subspace Identification*, Proceedings of the IEEE International Symposium on Computer-Aided Control System Design—CACSD'96, Dearborn, MI, September 15–18, 1996, U.S.A., pp. 182–187.

[23] V. Sima, *Subspace-based Algorithms for Multivariable System Identification*, Studies in Informatics and Control, Informatics and Control Publications, Vol.5, No.4, 1996, pp. 335–344.