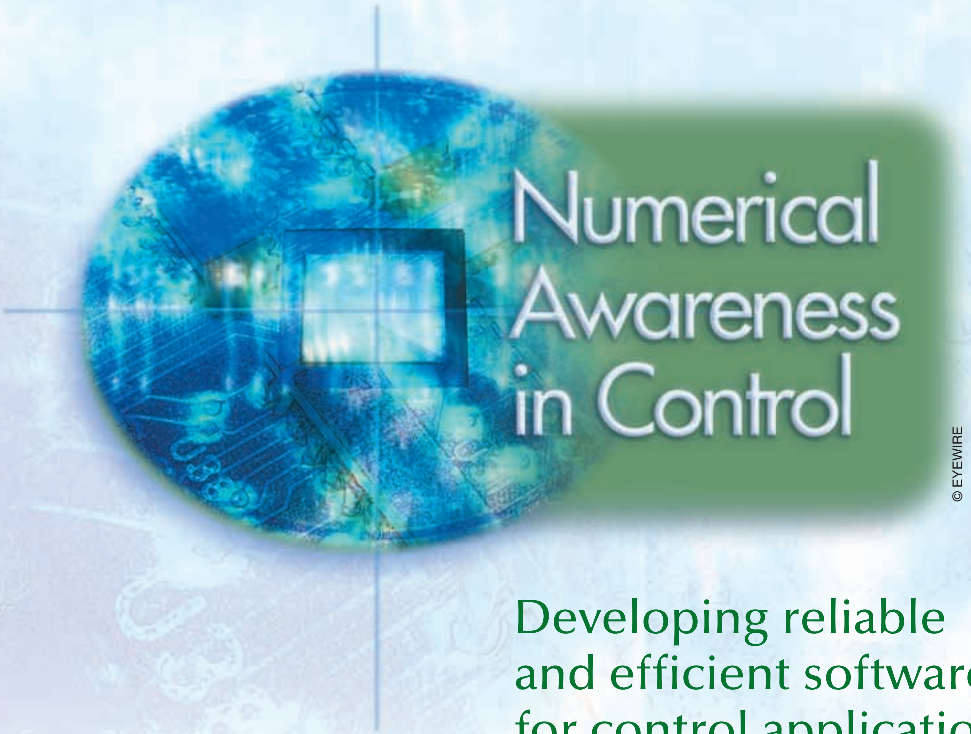
FEATURE

High-Performance Numerical Software for Control



Developing reliable and efficient software for control applications.

By Sabine Van Huffel, Vasile Sima, Andras Varga, Sven Hammarling, and François Delebecque

This article is concerned with the development of quality control-application software that performs efficiently and reliably on modern computing machines. In particular, we describe the subroutine library SLICOT (Subroutine Library In COntrol Theory) [1], give the background and motivation for the development of this library, and describe its most important features.

The need to solve the increasingly challenging large-scale problems that arise in control systems analysis and design has led to a need for more powerful algorithms able to solve these problems. Most users of standard control-oriented commercial software tools tend to have a blind confidence in the "quality" of the computed solutions and in the "efficiency" of the underlying methods. In many cases, however, it can be difficult to judge the numerical accuracy of the solutions as well as the efficiency of the implemented methods. It is a sad and mostly ignored fact that in many cases such tools are of dubious numerical quality. Therefore, to judge the numerical performance of algorithms,

0272-1708/04/$20.00©2004IEEE

we need to understand the requirements for satisfactory numerical algorithms. Some of the employed algorithms are inappropriate for solving large-scale problems and regularly fail even for relatively low-order problems. However, such failures are not always caused by the algorithms but rather their naive implementation. It is rarely the case that estimates of the accuracy of the solution or of the sensitivity of the problem being solved are also returned. The latter can be vital to help judge whether or not the modeling process has led to a well-posed problem.

The purpose of the SLICOT collection of computer-aided control system design (CACSD) algorithms is to have a strong basis for a new generation of control systems design packages by establishing a comprehensive standard set of numerically robust routines with known performance in terms of reliability and efficiency. To achieve this goal, it was necessary to develop new control-relevant algorithms by taking into account the latest developments in the numerical analysis field and relying on the expertise of qualified implementors.

The approach used in building SLICOT was to utilize the standard numerical linear algebra packages BLAS (Basic Linear Algebra Subprograms) [2]–[4] and LAPACK (Linear Algebra Package) [5], which form the basic linear algebra layer of SLICOT. Additionally, routines for the numerical solution of mathematical problems often encountered in CACSD are provided, such as linear and quadratic matrix equations, rational matrix factorization, and the computation of condensed forms. The applicability of the library is facilitated when its routines are embedded, by means of appropriate interfaces, in widely accepted user-friendly environments, such as MATLAB or Scilab [6]. Such a framework has all the advantages of open-source software and provides the user with a flexible and easy way to combine and experiment with the routines of the library and the tools of MATLAB and Scilab. The use of Fortran 77 as the implementation language has a considerable advantage with respect to execution speed, compared to similar genuine MATLAB or Scilab functions. Besides guaranteed numerical quality of algorithms, performance gains of 600% or more in speedup can occasionally be achieved. Moreover, the low-level nature of Fortran 77 makes it easy to call SLICOT from other languages such as C or C++, as well as to build interfaces to MATLAB and Scilab.

The outline of this article is as follows. First, we discuss the requirements for high-quality control software, giving some illustrative examples and indicating the features of high-quality software in the context of numerical computation. We then give an overview of linear algebra and control libraries that have been important influences in the development of SLICOT, describe the recent history of SLICOT, and follow with a section on the functional capabilities of SLICOT. Next we discuss and illustrate the performance of SLICOT and then describe some user-friendly interfaces to SLICOT. Finally, we indicate some future directions.

To judge the numerical performance of algorithms, we need to understand the requirements for satisfactory numerical algorithms.

Requirements on High-Quality Control Software

In developing a library of high-quality subroutines, several criteria must be fulfilled, such as reliability and accuracy of underlying algorithms, high computational efficiency, robustness of implementations, ease-of-use interfaces, rich functionality, and portability across a wide range of machines [7]. In what follows, we discuss these aspects in the context of developing the copyrighted freeware subroutine library SLICOT to solve control-related computational problems. The implementation of these requirements in the software is not only time consuming but also requires skilled experts in numerical analysis, control, and computational programming. Commercially, these requirements are not necessarily attractive. Instead, the implementation of these requirements in software should be performed in a standardized way and coordinated by an academic consortium. This group can then join with experts worldwide to promote the dissemination of the software by means of integration into the existing commercial and freeware packages, thereby improving the overall software quality.

Reliability

The reliability of numerical software is strongly related to the guaranteed accuracy of the computed results and can only be ensured by a proper selection of the underlying algorithms. Many basic linear algebra algorithms, as well as some control-related algorithms, are proven to be numerically stable. Numerical stability (or more exactly numerical backward stability) of an algorithm means that the results computed by that algorithm are exact for slightly perturbed original data. As a consequence, a numerically stable algorithm applied to a well-conditioned problem will produce guaranteed accurate results. If a problem is intrinsically ill-conditioned, no algorithm is guaranteed to deliver an accurate result. However, numerically stable algorithms are guaranteed to solve a problem close to the exact problem and can thus provide accurate solutions for acceptably well-conditioned problems. On the other hand, unstable algorithms could produce wrong results even for well-conditioned problems.

Example 1

The poles of a minimal-order linear system with $n \times n$ state matrix A are the roots of the characteristic polynomial of A, $\det(A - \lambda I_n)$, where I_n is the identity matrix of order n, and λ is a complex variable; equivalently, the poles are the eigenvalues of A. For

$$A = \begin{pmatrix} 1 & \varepsilon \\ \varepsilon & 1 \end{pmatrix},$$

with $\varepsilon \in \mathbb{R}$, the characteristic polynomial is $\lambda^2 - 2\lambda + (1 - \varepsilon^2)$, so the poles are $\lambda_1 = 1 + \varepsilon$, and $\lambda_2 = 1 - \varepsilon$. Let ε_u be the round-off unit of the computer used for computations, that is, the smallest positive representable number so that the floating-point representation of $1 + \varepsilon_u$, $\mathrm{fl}(1 + \varepsilon_u)$, is greater than 1 (but $\mathrm{fl}(1 + \nu) = 1$, for $0 \le \nu < \varepsilon_u$). If $\varepsilon^2 < \varepsilon_u < \varepsilon < 1$, then the roots computed using the MATLAB commands `roots(poly(A))` are $\hat{\lambda}_1 = \hat{\lambda}_2 = 1$, since $\mathrm{fl}(1 - \varepsilon^2) = 1$. Hence, roughly half of the accuracy could be lost by this simple computation. The interested reader can try this example in MATLAB on an Intel Pentium machine for $\varepsilon = 5 \cdot 10^{-9}$. On the contrary, if the poles are computed using the MATLAB function `eig(A)`, full accuracy will be retained. This example shows that intuitive algorithms can be unreliable in finite-precision computation even on well-conditioned problems.

The difficulties shown by the above example will be even more severe for larger problems or sophisticated algorithms, which can require the solution of many subproblems. A basic ingredient for achieving numerical stability is the use of orthogonal transformations wherever possible. Their usage often makes it possible to find bounds for perturbations of the initial data that are equivalent to the cumulative effect of round-off errors during the computations therefore verifying the numerical stability of an algorithm. One approach to developing numerically reliable algorithms for solving control problems is to reduce the original problem, using orthogonal transformations, to an equivalent one that is easier to solve. The reduced problem is solved by specialized algorithms, and the solution of the original problem is recovered by back transformation.

A key aspect for enhancing the reliability of algorithms is to exploit structural information about the underlying computational problem. The following examples show that, without exploiting the problem structure, numerical reliability can be lost even when using numerically stable algorithms.

Example 2: Eigenvalues of Hamiltonian Matrices

Hamiltonian matrices play an important role in solving Riccati equations, where the main numerical problem is the computation of eigenvalues and orthogonal bases of special invariant subspaces. Consider the matrices

$$H = \begin{pmatrix} A & F \\ Q & -A^T \end{pmatrix}, \qquad J = \begin{pmatrix} 0 & I_n \\ -I_n & 0 \end{pmatrix},$$

where A, F, $Q \in \mathbb{R}^{n\times n}$ and F and Q are symmetric matrices. The matrix H has the property that $JH = (JH)^T$. Such matrices are called Hamiltonian. Since $J^T = -J = J^{-1}$, it follows that $H^T = JHJ$, or $J^T H^T J = -H$. Hence, $-H$ has the same eigenvalues as H^T and H, and so, if $\lambda \in \Lambda(H)$, then $-\lambda \in \Lambda(H)$, with the same (algebraic) multiplicity. (The notation $\Lambda(M)$ denotes the set of eigenvalues of the matrix M.) However, if the eigenvalues are computed numerically with a standard eigensolver (for instance, with the function `eig` from MATLAB), this pairing property can no longer be guaranteed. For instance, by taking

$$A = \begin{pmatrix} 10 & 0 \\ 10 & 0 \end{pmatrix}, \quad F = \begin{pmatrix} 10 & 10 \\ 10 & 0 \end{pmatrix}, \quad Q = \begin{pmatrix} 0 & 10 \\ 10 & 10 \end{pmatrix},$$

the MATLAB function eig(H) gives

```
± 17.32050807568877,
−1.094633850125614e−15
                    ± 1.003289648705741e−7i.
```

The last two eigenvalues should be zero, and hence more than half the machine accuracy was lost. Moreover, these eigenvalues form a complex conjugate pair, and thus are qualitatively wrong, since they should have opposite values (if nonzero). An algorithm relying on the above-mentioned pairing property would therefore fail.

On the other hand, using the combination of the SLICOT routines `MB04ZD` and `MB03SD` to implement a Hamiltonian structure-preserving algorithm [8], yields the eigenvalues ±17.32050807568877, 0, 0.

Example 3: Computation of Hankel Singular Values

The Hankel singular values are input–output invariants of stable linear systems and play a fundamental role in finding balanced realizations and in model reduction. They are defined as the nonnegative square roots of the eigenvalues of the product $P_c P_o$, where P_c and P_o are the nonnegative definite controllability and observability Gramians, respectively, and $P_c P_o$ has nonnegative eigenvalues. For a stable state-space realization (A, B, C), $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, $C \in \mathbb{R}^{p\times n}$, of a discrete-time linear time-invariant system, the Gramians are given by the solutions of the stable discrete-time Lyapunov equations, also called Stein equations

$$A P_c A^T - P_c = -BB^T \quad \text{and} \quad A^T P_o A - P_o = -C^T C.$$

By solving these equations without taking into account the symmetry and semidefiniteness of the solutions, round-off errors can cause the computed Gramians to become nonsymmetric or indefinite. This error can result in negative or even complex Hankel singular values, which contradicts the system-theoretic properties. For instance, considering

$$A = \begin{pmatrix} 0.0101 & -0.0030 & 0.0167 \\ -0.0117 & 0.0055 & -0.0334 \\ 0.0566 & 0.0451 & -0.0342 \end{pmatrix},$$
$$B = \begin{pmatrix} 0.4678 \\ -0.3276 \\ -0.7043 \end{pmatrix},$$
$$C = \begin{pmatrix} -0.8113 & -0.6185 & -0.0309 \end{pmatrix},$$

the MATLAB functions `gram` and `eig`(P_cP_o) give (with five significant digits)

$$\texttt{2.4147e}{-}\texttt{2},\ \texttt{2.7170e}{-}\texttt{7},\ {-}\texttt{5.7231e}{-}\texttt{19}.$$

Hence, the computed Hankel "singular values" are the nonnegative square roots of the above eigenvalues, i.e.,

$$\texttt{1.5539e}{-}\texttt{1},\ \texttt{5.2125e}{-}\texttt{4},\ \texttt{7.5651e}{-}\texttt{10i},$$

the last value being purely imaginary! On the other hand, the SLICOT function `AB13AD` computes the real Hankel singular values

$$\texttt{1.5539e}{-}\texttt{1},\ \texttt{5.2125e}{-}\texttt{4},\ \texttt{9.0630e}{-}\texttt{10},$$

by exploiting the fact that the Hankel singular values can be equivalently computed as the singular values of the product R_cR_o, where the upper triangular matrices R_c and R_o are the Cholesky factors of the Gramians satisfying $P_c = R_cR_c^T$ and $P_o = R_o^TR_o$. It is essential to compute R_c and R_o by solving the Lyapunov equations directly for these factors using the algorithm in [9]. Also, the condition numbers (in the two-norm) of the Gramians are the squares of (and hence larger than) those for their factors, and, therefore, the SLICOT approach is to be preferred.

Structure-preserving algorithms have been developed for many control domains, and robust software implementations are available in SLICOT. Some of the most recent algorithmic developments are: balancing-free square-root methods for model reduction [10] (and the references therein), periodic Schur methods for solving periodic Lyapunov equations [11], descriptor systems analysis procedures [12], symplectic methods for solving Riccati equations [13], [14], and subspace identification methods [15] (and the references therein).

A key ingredient for assessing the quality of computed results is the use of condition estimators. Since many real-life problems include hidden ill-conditioned subproblems, it is useful to have techniques for tracking the conditioning of the subproblems that can significantly affect the accuracy of the final results. Cheaply computable condition number estimators are available for most of standard linear algebra problems and are implemented in LAPACK [5]. Condition number estimates and error bounds are also available in SLICOT for some control related computations, as the solution of linear matrix equations (Lyapunov, Stein, and Sylvester).

Efficiency

The availability of efficient algorithms and codes is important for solving large-scale engineering design problems, where extensive computations have to be performed repeatedly, for various design parameters. Efficient algorithms are also essential for some *real-time* control problems, where the results must be available in time slots smaller than the sampling period. There are several factors that affect the computational efficiency. At the algorithmic level, it is essential to take advantage of structure that the underlying problem might possess, such as symmetry, or definiteness. Therefore, we need to design specialized algorithms for each important type of data processed by these algorithms. At the implementation level, it is important to ensure flexibility, portability, and adaptability to various computing platforms, including the ability to use the potential of modern high-performance computer architectures (vector, parallel, with memory hierarchies). For medium-size and large-scale problems, exploiting the memory hierarchies is a key issue, since the problem data cannot be fully stored in the fastest (cache) memory level, and frequent data moving between the fast and slow memory levels will strongly degrade the computational performance. A similar argument is valid for parallel computing architectures, where the communication between processors is usually the bottleneck in terms of efficiency.

In view of these needs, it is necessary to make use of any existing tools for performance improvement. These tools include high-performance compilers, such as (high-performance) Fortran, message-passing paradigms, such as MPI (for inter-processor communication), and preexisting efficient computational building blocks such as LAPACK and optimized BLAS and their parallel counterparts, ScaLAPACK [16] and PBLAS. The use of suitable data structures is also essential; for instance, the solution of large-scale problems typically involves operations with sparse matrices. A sparse matrix has most of its entries zero. The essential idea is to store and operate with the nonzero elements only. Besides the values of the nonzero elements, information about their row/column indices must be stored.

To illustrate the benefits of exploiting sparsity, consider a 2000-by-2000 tridiagonal matrix A, with nonzero elements only on the main diagonal and the first sub- and superdiagonal. Such a matrix has 5,998 nonzero elements at most. Using MATLAB sparse matrix computations, each solution of $Ax = b$, for several vectors b, was obtained in no more than 0.05 s on a Pentium 3 machine at 500 MHz. But when A was considered as a full matrix, the calculations needed about 1.43 s. This speed-up results from using the general sparse matrix representation. If the tridiagonal structure is exploited by a solver, even greater efficiency is possible. Then the number of floating-point operations is $O(n)$, compared to $O(n^3)$ for a general linear system solver, where n is the order of A.

Algorithm-blocking and vectorization techniques have contributed decisively to the high performance of many LAPACK codes, and this feature has been partly inherited by the implementations of many SLICOT subroutines, where many linear algebra subproblems are handled near the peak achievable performance. Moreover, in all SLICOT routines matrix–matrix operations are performed by means of the corresponding BLAS routines [4]. Still, generally, the algorithms implemented in SLICOT are not block oriented but can be essentially considered as highly vectorized. Thus, we expect that the SLICOT routines will provide near-peak performance on many vector processors, but the performance for large-order systems could be limited on some machines by the rate of data movement.

Robustness

Robustness of software implementations means avoiding overflows, harmful underflows, and unacceptable accumulation of round-off errors. To achieve robustness, the computational routines must be able to determine the floating-point properties, such as the overflow and underflow thresholds OVFL and UNFL, at run time, without overflowing. OVFL and UNFL can be used for scaling to prevent overflow and harmful underflow during subsequent calculations. By careful implementation, it is frequently possible to avoid overflow in situations where the numerical results lie within the range of representable values, but where partial results can overflow. A typical example is the Euclidean length of a real vector $x : ||x||_2 := (\sum_i x_i^2)^{1/2}$. If $|x_i| > \mathrm{OVFL}^{1/2}$, then x_i^2 will overflow, although $||x||_2$ may be well below OVFL. A careful implementation of this norm computation is a nontrivial task, and a special routine is available in BLAS [2] for this purpose. Another aspect of implementation robustness is the use of default values for tolerances. These values are used, for instance, for rank determination or in the stopping criteria for iterative processes, so that meaningful results are returned for well-scaled problems. In this context, the round-off unit ε_u plays an important role. To determine OVFL, UNFL, ε_u, and other double precision floating-point representation related quantities, the LAPACK routine DLAMCH is called by SLICOT routines.

A control specific aspect of computational robustness is the scaling of initial data. The entries of the state space matrices (A, B, C) resulting from physical modeling or system identification often have a wide range of magnitudes. An example is a state-space realization of a Butterworth filter, obtained, for instance, by using the MATLAB function `butter`. Scaling a linear state-space system model (A, B, C) implies determining an input-output equivalent representation $(\widetilde{A}, \widetilde{B}, \widetilde{C})$ by means of a similarity transformation of the form

$$\left(\widetilde{A}, \widetilde{B}, \widetilde{C}\right) := \left(T^{-1}AT, T^{-1}B, CT\right),$$

where T is a diagonal matrix chosen such that the range of entries of the triple $(\widetilde{A}, \widetilde{B}, \widetilde{C})$ is the least possible. Scaling a state-space model often contributes to improving the accuracy of results and in extreme cases can even prevent algorithmic failures because of false rank determinations, or extreme inaccuracies. Scaling can be optionally done in almost all SLICOT routines where state-space models are manipulated (for example, minimal realization, model reduction, and conversions). Scaling of individual matrices is also implicitly performed in many linear algebra routines of LAPACK, as for example in computing eigenvalues or solving Sylvester-like matrix equations.

Ease of Use

Ease of use of numerical software can be achieved if implementation details are largely hidden from the user and thus not visible through the user interface. In this regard, SLICOT provides user-callable routines, which cover a typical system theoretic or mathematical functionality. These routines are fully documented, and for most of them, accompanying simple test examples are also available to allow a quick check of the installation correctness. Many user-callable routines rely on supporting routines, which implement either computational steps if a more complex algorithm is considered, or solve special subproblems (for example, after reducing the original problem to a simpler form).

Another aspect of ease-of-use is to provide meaningful default settings for all algorithm parameters, such as tolerances for rank determination, and tolerances for stopping iterative processes. In most SLICOT routines, method-related parameters have standard settings, which are satisfactory in most cases.

Wide Scope

Wide scope refers to the range of control problems and system representations that the algorithms and software can address. This scope implies rich functionality. To illustrate, the system representations handled in SLICOT are

- continuous-time and discrete-time standard state-space models of the form

$$\begin{aligned}\dot{x}(t) &= Ax(t) + Bu(t)\\ y(t) &= Cx(t) + Du(t)\end{aligned}$$

and

$$\begin{aligned} x(t+1) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) + Du(t) \end{aligned}$$

- continuous-time and discrete-time generalized (or descriptor) state-space models of the form

$$\begin{aligned} E\dot{x}(t) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) + Du(t) \end{aligned}$$

and

$$\begin{aligned} Ex(t+1) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) + Du(t), \end{aligned}$$

where E is a square matrix, and $A - \lambda E$ is regular
- input–output models given by rational Laplace-transform or Z-transform transfer-function matrices $G(s)$ or $G(z)$, respectively
- polynomial fraction input–output transfer-function matrix models represented as $G(\lambda) = N(\lambda)M(\lambda)^{-1}$ or $G(\lambda) = M(\lambda)^{-1}N(\lambda)$, where $N(\lambda)$ and $M(\lambda)$ are polynomial matrices in the complex variable $\lambda = s$ for a continuous-time system and $\lambda = z$ for a discrete-time system
- time-series of input and output vectors for state-space system identification. Several routines are provided to perform conversions between these representations.

Although SLICOT can handle all of these system representations, SLICOT has been restricted to state-space analysis and design methods because of the intrinsic ill-conditioning present in manipulating polynomial and rational matrices.

Portability

Since code developed on one machine is often embedded (and hidden) in an application on another machine, and possibly used on a third machine, it would be unreasonable to expect a user acquiring code to modify all of its subparts to ensure that they run correctly. Hence, portability of programs has always been an important consideration in implementing numerical libraries like LAPACK and SLICOT. We use here the term portability in the restricted sense of portability of functional correctness, which means that a code written in a standard language such as Fortran will run correctly on an arbitrary machine with an arbitrary Fortran compiler. The more general concept of portability of performance is more difficult to achieve (see [7]).

One way to reduce code-porting difficulties is to isolate non-portable features such as floating-point properties in a few routines and modify only these routines when porting is necessary. Alternatively, the developers of LAPACK implemented special routines (`SLAMCH`/`DLAMCH`) to return floating-point properties at run time. By doing so, a tradeoff between portability, on the one hand, and efficiency, accuracy, and robustness, on the other hand, automatically results. For example, assuming the availability of IEEE arithmetic for the target machine and standard high level language access to its exception handling, this approach allows significantly faster, more accurate, and more robust code in many LAPACK routines [7].

The key to portability is to rely on standards. In the case of SLICOT, the implementation language is Fortran 77, with only one non-standard extension. This exception is the double precision `COMPLEX*16` data structure, which is supported by a majority of compiler vendors and is now standard in Fortran 90. Another aspect of achieving the high portability of SLICOT is the use of the standard libraries BLAS and LAPACK as underlying linear algebra tools.

The BLAS can be seen as a key to the portability of performance. Its use in LAPACK and also in SLICOT can provide portable high performance on many platforms, for which assembly-coded BLAS tuned for particular architectures is available. The freely available standard implementation of BLAS serves not only for code development and testing, but may substantially improve the efficiency of programs when they run with nonoptimizing compilers.

Reusability

Many sophisticated CACSD platforms, such as ANDECS [17], EASY5 of the Boeing Company, MATLAB, or Scilab [6], rely on robust implementations of numerically reliable and computationally efficient algorithms. In the architecture of such CACSD platforms we can identify and usually access a basic computational layer consisting of subroutine libraries, such as RASP (a product of the Deutsches Zentrum für Luft- und Raumfahrt e.V. (DLR), Oberpfaffenhofen, Germany) in ANDECS [17], SLICOT in EASY5, or built-in functions in MATLAB and Scilab. This layer includes all computational routines for specific mathematical and control computations, simulation, and optimization. An important advantage of developing control libraries such as RASP and SLICOT is that the process is not restricted by specific requirements of the CACSD platform operation, by the languages used for its implementation, or by the employed data structures. Moreover, such control libraries can serve the development of several platforms, or can be used within other dedicated engineering software systems. This low-level reusability can only be achieved by using general purpose programming languages such as C or Fortran 77 as implementation languages.

In the case of RASP and SLICOT libraries, Fortran 77 has been chosen as the implementation language since routines from the high-performance linear algebra packages

BLAS, EISPACK, LINPACK, and recently LAPACK, can be directly and efficiently called.

Overview of Linear Algebra and Control Libraries

For the past 20–25 years, there has been a great deal of activity in the area of algorithms and software for solving both linear algebra problems and control-relevant computational problems. High performance on codes that are portable across platforms have been achieved by identifying the computational hierarchy of the main linear algebra packages (for example, BLAS, EISPACK, LINPACK, LAPACK) and of several control libraries based on them. It is important to emphasize the influence of the underlying linear algebra tools on the development of control-oriented software. It seems appropriate, in this historical section, to give a brief review of the linear algebra packages that represent milestones in developing high quality portable numerical software.

The need to solve the increasingly challenging large-scale problems that arise in control systems analysis and design has led to a need for more powerful algorithms.

Linear Algebra Libraries

The Handbook

During the 1960s a number of Algol procedures were developed and published in the journal *Numerische Mathematik*, then collected into a volume [18], generally referred to simply as "The Handbook."

EISPACK

"The Handbook" was the basis for the EISPACK collection, an influential set of Fortran routines for solving eigenvalue problems, and the first of many "PACKs" [19], [20].

BLAS

Another fundamental development at this time was the design of the Basic Linear Algebra Subprograms (the BLAS), a set of kernel routines for basic scalar and vector operations such as generating a plane rotation, computing the Euclidean length of a vector, and computing the dot product of two vectors [2]. In view of subsequent developments, these are now usually referred to as the Level 1 BLAS. The advent of vector machines in the late 1970s with vector registers and pipelines meant that the Level 1 BLAS were at too low a level of granularity to take advantage of the speed of these machines, and this realization led to the development of specifications for a set of Level 2 BLAS for matrix-vector operations [3]. Soon afterwards the level of granularity was raised once again with the specification of a set of Level 3 BLAS for matrix–matrix operations targeted at machines with hierarchies of memory and shared memory multiprocessor machines [4]. The aim of the specification of the BLAS has been to encourage the production of efficient machine-specific versions, and this aim has been successfully realized.

LINPACK

The success of EISPACK motivated the development of a second package of high quality software for linear algebra problems, LINPACK, for solving linear equations and linear least squares problems [21]. One of the distinctive features of LINPACK was efficiency, which was achieved by the column orientation of the algorithms (storage of arrays in Fortran being by column) and the use of the Level 1 BLAS.

LAPACK

The advent of vector machines, of machines with hierarchies of memory, and of multiprocessor machines meant that the earlier software packages were no longer efficient on modern machines. This realization led to the development of LAPACK, a linear algebra package for the solution of dense and banded linear algebra problems, combining and extending the facilities of EISPACK and LINPACK. LAPACK uses the Level 3 and Level 2 BLAS wherever possible for efficiency, and also puts great emphasis on error and condition estimation.

Control Libraries

The development of efficient, reliable, and portable numerical software for CACSD requires joint expertise in control theory, numerical mathematics, programming, and software engineering. Therefore, the development of fully tested, production-quality numerical software for CACSD is a challenging and time-consuming task, which involves cooperative efforts over a lasting period of time. In what follows we present a short historical survey of control libraries that contributed to the development of the current release of the SLICOT library. From this limited perspective, our survey is biased toward tracing the roots of this library and therefore we apologize for any omissions. For a fairly complete inventory of control libraries and CACSD packages available around 1985 see [22]. A more recent presentation is in [1].

The SLICOT library is the result of international cooperation among leading numerical algebra and control numerics experts. Many routines in SLICOT originate from earlier implementations available in the control libraries BIMAS/BIMASC, LISPACK, RASP, SLICE, SYCOT.

BIMAS and BIMASC are Fortran 77 control libraries developed at the Research Institute for Informatics (ICI), Bucharest, Romania, and used to build the interactive package SIPAC [23]. BIMAS [24] extends the capabilities of the underlying EISPACK and LINPACK software for control-related computational problems, such as solving matrix equations (Lyapunov, Sylvester, Riccati), computing matrix exponentials, and reordering Schur forms. The library BIMASC [25], based on BIMAS, provides a rich collection of subroutines covering system analysis, modeling, design, and simulation.

LISPACK is another East European control library developed by a Bulgarian group from the Technical University of Sofia. LISPACK is based mainly on EISPACK and its content is similar to that of BIMAS. LISPACK served as the basis of SYSLAB [26], an interactive package extending the 1984 free initial version of MATLAB.

The development of the RASP library started in the early 1980s at the University of Bochum and was continued by the control group at the German Aerospace Center (DLR) in Oberpfaffenhofen, Germany. Based on EISPACK and LINPACK, RASP covers a broad area of control engineering computations supporting frequency- and time-domain analysis and synthesis techniques, multi-criteria parameter optimization, simulation, and graphics. Special attention is given to the numerical reliability of the implemented algorithms. The last official release, RASP'95, consisted of about 350 user-callable routines. RASP and the engineering-database and operating system RSYST together formed the software infrastructure of the computer-aided control engineering environment ANDECS [17].

In the early 1980s, a British cooperative initiative led to the development of the control library SLICE, which contains a set of almost 40 control routines. When this initiative ended, the routines were further distributed by the Numerical Algorithms Group (NAG) from Oxford (U.K.), who issued a revised version of SLICE. In the same period, the Working Group on Software (WGS) was founded as a Benelux cooperation among several academic institutes and industries, aiming to develop reliable control software.The first achievement of the WGS was the development of the control library SYCOT. To produce a library that meets professional standards, the WGS associated itself in the late 1980s with NAG, and they decided to integrate their libraries, SLICE and SYCOT, into a new library, called SLICOT. This cooperation was effective and led to the first release of SLICOT in 1991. A second release of SLICOT in 1993 contained about 90 user-callable routines for computations related to the analysis, modeling, transformation, and synthesis of systems. Around 1995, the WGS began to operate on a European level.

SLICOT is a general purpose basic control library and can primarily be viewed as a mathematical library for control theoretical computations. The main emphasis in SLICOT is on numerical reliability of implemented algorithms and the numerical robustness and efficiency of routines. Special emphasis is placed on algorithmic flexibility and rigorous implementation and documentation standards (see [27]).

Recent Evolutions of the SLICOT Library

Around 1994, only SLICOT and RASP were in active development. To avoid duplication of effort, DLR and WGS introduced the mutual compatibility concept, which enabled the coordinated development of both libraries. Part of this agreement was to incorporate the numerical linear algebra packages BLAS Level 3 [4] and LAPACK [5] in both libraries.

The RASP/SLICOT cooperation was only a first step toward the realization of a standard, generally accepted, platform for computational control tools. A more recent development in this direction was the first public release (Release 3) of SLICOT in 1997, when WGS, NAG, and DLR took the initiative to extend the scope of cooperation to a European level and make SLICOT freely available to ensure faster and wider distribution. A European network, called NICONET (Numerics In COntrol NETwork), for defining standards, and producing and disseminating CACSD software was established, joining research centers and universities with complementary expertise in the development of numerical control software. The main objectives were to enlarge the control systems areas covered with new topics, such as subspace identification, robust control, model reduction, descriptor systems, and nonlinear systems, and to better fulfill the users' requirements and industrial needs. This successful cooperation between 11 universities/research institutes and six companies, started in 1997, led to the rapid development of the SLICOT Library, which now includes more than 200 user-callable routines and 150 lower level documented routines (compared to 90 user-callable routines in 1997), covering the basic areas in systems and control theory. In addition, extended documentation, benchmarks, and industrially oriented test examples, as well as gateways to important CACSD environments such as MATLAB and Scilab, are provided for many typical CACSD calculations, making the software more accessible for industry and control education. Detailed information about SLICOT and NICONET is provided at the Web site http://www.win.tue.nl/niconet/. Since 1997, SLICOT has been copyrighted freeware; go to http://www.win.tue.nl/niconet/NIC2/slicot.html to download the software. Online html documentation files are available. Built on LAPACK and BLAS (and partly on their extensions for parallel computers, ScaLAPACK and PBLAS), the SLICOT routines can exploit the capabilities of some modern high-performance computer architectures.

SLICOT Library Functional Capabilities

The SLICOT Library includes the following chapters: Analysis Routines, Benchmark and Test Problems, Data Analysis, Filtering, Identification, Mathematical Routines, Synthesis Routines, Transformation Routines, and Utility Routines. A chapter-by-chapter SLICOT library contents with sections and subsections is given in [28]. The basic computational tools in the SLICOT library cover several problem areas in systems and control theory.

System analysis routines perform tasks such as finding condensed forms of system matrices exhibiting structural properties, computation of invariant zeros of a system, or various system norms. A rich collection of routines is available for model reduction using accuracy enhanced methods. Analysis of generalized state-space systems is also covered.

Benchmark and test problems routines generate benchmark examples for time-invariant dynamical systems, as well as for (generalized) Lyapunov and algebraic Riccati equations.

Data analysis routines deal with the convolution/deconvolution of two signals and perform transforms of real or complex signals.

Filtering routines cover enhanced accuracy (square-root) covariance propagation schemes for time-varying and time-invariant filters, including the conventional Kalman filter and fast recursive least-squares filter.

Identification routines implement subspace identification techniques for both linear time-invariant state-space systems and nonlinear Wiener-type systems.

Mathematical routines implement algorithms for computations that are not available in LAPACK or BLAS. Examples include structured matrix factorizations, including those for (block) Toeplitz or Hessenberg matrices, solution of the corresponding linear systems, special updating of QR-like factorizations, calculations related to polynomials and matrix polynomials, and solving (structured) nonlinear least-squares problems. Worth mentioning are specialized algorithms such as the computation of the Kronecker-like staircase form of a linear pencil, the computation of the minimal polynomial basis of a polynomial matrix, and the reduction of a product of matrices to the real Schur form without forming the product using the periodic Hessenberg and Schur forms.

System synthesis routines cover the computational problems in control systems design: pole assignment; solution of algebraic Riccati equations, solution of (generalized) Lyapunov and Stein equations, of Sylvester equations $AX + XB = C$, $AXB + X = C$ and generalized Sylvester equations $AR - LB = C$, $DR - LE = F$ (solved for R and L); minimum norm feedback matrix for deadbeat control; coprime factorization of transfer-function matrices; and designing H_∞ (sub)optimal, and H_2 optimal state controllers. All factorization and synthesis routines, as well as most matrix equation solvers, cover continuous-time as well as discrete-time settings.

Transformation routines include conversions between various system representations. A set of routines perform conversions between state-space representations, such as calculation of the controller/observer Hessenberg forms, or compute the minimal realization of a state-space representation. Other routines transform a given system representation to another representation, such as state-space to polynomial, or rational matrix representation (or conversely), polynomial representation to frequency response. Transformation routines for generalized state-space systems are also provided.

Most SLICOT routines work for matrices with real entries.

We illustrate the functional flexibility of the basic computational tools, taking as an example the `SB03OD` routine, which solves small or medium-size stable non-negative definite Lyapunov equations. Let op(M) denote either the matrix M or its transpose M^T. `SB03OD` computes the solution X of either the stable nonnegative definite continuous-time Lyapunov equation

$$\mathrm{op}(A)^T X + X\mathrm{op}(A) = -\sigma^2 \mathrm{op}(B)^T \mathrm{op}(B), \qquad (1)$$

or the convergent discrete-time Lyapunov equation

$$\mathrm{op}(A)^T X \mathrm{op}(A) - X = -\sigma^2 \mathrm{op}(B)^T \mathrm{op}(B), \qquad (2)$$

in the factored form $X = \mathrm{op}(U)^T \mathrm{op}(U)$, where U is upper triangular. The factor U is determined instead of X by Hammarling's variant of the Bartels/Stewart method [9]. The first step of this algorithm is the reduction of A to the real Schur form (RSF). Next, the respective equation is solved for the reduced form of A, and the solution is recovered by back-transforming the computed factor. Note that the resulting scalar σ is a subunitary scaling factor (usually set to one) determined by the routine to prevent solution overflow. In many applications such as model reduction, the solution for both forms of op(A)in (1) or (2) are needed (see also Example 3). The solver's ability to deal with the two forms of op($\cdot$) in conjunction with the reduced form of A is advantageous in this context since only one reduction of A to the RSF is needed to solve both equations. Since the reduction to RSF is more expensive than the solution of reduced equations and the back-transformation of the factor, the computational effort for solving two equations is practically the same as that of a single equation. The special case in which A is already in RSF is handled by the solver `SB03OD` as an option.

Similar capabilities are implemented in other solvers. The codes for solving algebraic Riccati equations have options for various scaling strategies and for sorting the eigenvalues (so that antistabilizing Riccati solutions can be obtained); linear quadratic optimization problems with coupling terms can be optionally solved. Special cases of matrices in real Schur form, and/or Hessenberg form can efficiently be dealt with by additional solvers. Estimates of the reciprocal condition numbers for Riccati and Lyapunov equations can be computed.

The SLICOT Library contents is summarized in Table 1. The current version includes 415 documented user-callable or programmer-callable routines as well as 193 example programs, with associated data files and results. The downloadable library archive file contains over 2,200 files.

Table 1. SLICOT Library summary. The numbers in the column labeled "Main" refer to user-callable routines, while the numbers in the column labeled "Support" refer to programmer-callable or low-level auxiliary routines.

Chapter	Main	Support	Total
Analysis	37	15	52
Benchmark	6	0	6
Data analysis	7	2	9
Filtering	6	0	6
Identification	5	10	15
Mathematical	72	67	139
Nonlinear	0	16	16
Synthesis	48	68	116
Transformation	37	14	51
Utility	5	0	5
Total	223	192	415

SLICOT Performance

In this section we compare some performance characteristics of SLICOT implementations such as speed and accuracy to those of functionally equivalent software tools available in MATLAB. The comparisons are performed entirely within MATLAB by using appropriate gateway functions for the SLICOT subroutines. The MATLAB functions that we compare are implemented in the MATLAB language and belong to the Control Toolbox of MATLAB. Note that the Control Toolbox, as well as similar toolboxes, are widely used CACSD tools in control education and industry. Therefore, we consider it important to understand the limitations of presently available tools and the need for an alternative paradigm for high-performance CACSD.

Our comparisons illustrate the increased speed of SLICOT-based gateways at a same or better level of accuracy. However, before discussing the results, it is appropriate to comment on the limitations of a high-level interpretative language such as MATLAB for implementing control-oriented algorithms.

In this comparison it is helpful to recognize the balance between MATLAB's matrix handling power and the desire to exploit intrinsic structural aspects of the problem. Exploiting the structural features of computational problems often has the paradoxical effect of causing larger execution times, due to overhead in the interpretational operation mode of MATLAB. Thus, high-order control problems can rarely be tackled in an efficient way. In contrast, implementing algorithms in Fortran or C allows the use of appropriate data structures, as well as the exploitation and preservation of structural features, and can drastically improve the performance of algorithms. Because of the flexibility allowed by such programming languages, algorithmic details can be explicitly addressed, the computational flow can be optimized, and memory use minimized. Such opportunities have been exploited when implementing the SLICOT codes. Consequently, the efficiency of many MATLAB functions provided in toolboxes is minor compared to similar implementations in Fortran control libraries.

Overcoming such limitations has several side effects, which can lead to severe performance losses. For example, in many control-related algorithms the RSF of a real matrix plays an important role in computations. However, because of possible 2×2 blocks on the diagonal, the exploitation of the RSF structure in some algorithms (for example, when solving linear matrix equations) involves complicated index manipulations and nonstandard cycling, which significantly deteriorate the performance. The widely used ad-hoc solution is to turn computations to the complex field, where the simpler, complex Schur form is upper triangular. The net result is at least doubling the computation times for the complex part of the algorithm. This point is illustrated in the case of the `lyap` function for solving Lyapunov equations.

The lack of structure exploitation can lead to wrong results or unreliable computations. A typical example is the computation of Hankel singular values (see Example 3) in the context of balancing linear systems. Without solving the Lyapunov equations for the Cholesky factors of the Gramians directly, the computed Gramians can become slightly nonnegative for nearly nonminimal systems. In this case, the computation of the Cholesky factors from the computed Gramians can fail. Consequently, the balancing function `balreal` from the Control Toolbox can fail on random stable systems [for example, when executing the command `balreal(rss(20,1,1))`]. At the same time, the dimension of the problem can increase to 1,000 without problems when employing the SLICOT gateway function `sysred` for the same purpose.

Finally, the lack of structure exploitation can lead to a catastrophic slowdown of algorithms as in the case of the Control Toolbox function `ctrbf` for single-input systems. Here, the computation reduces a matrix to Hessenberg form by using an algorithm employing orthogonal Householder transformations. These transformations can be compactly stored and efficiently applied to other matrices, making the overall reduction algorithm of computational complexity $O(n^3)$, where n is the order of the system. However, if the transformations are explicitly accumulated at each step and then applied to a matrix, the computational complexity of the reduction becomes $O(n^4)$. Note that the structure-exploiting implementation of the reduction algorithm in SLICOT prevents loss of efficiency.

To illustrate some of the above points, we performed several test runs using SLICOT-based gateways and equivalent MATLAB functions. The calculations were performed on an IBM PC computer at 500 MHz, with 128 MB memory,

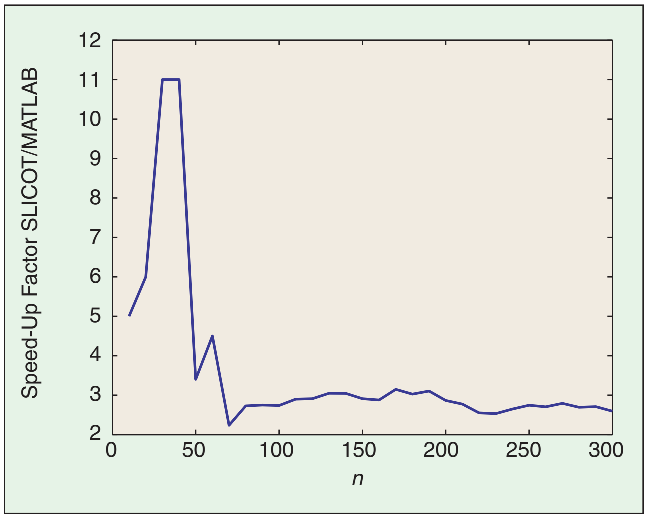


Figure 1. *Speed-up factor comparison: SLICOT* `sllyap` *versus MATLAB* `lyap`*. Random continuous-time Lyapunov equations with* $n = 10 : 10 : 300$ *are solved. The function* `sllyap` *is 4 to 11 times faster than* `lyap` *for small orders* ($n < 50$) *and more than twice as fast as* `lyap` *for larger orders.*

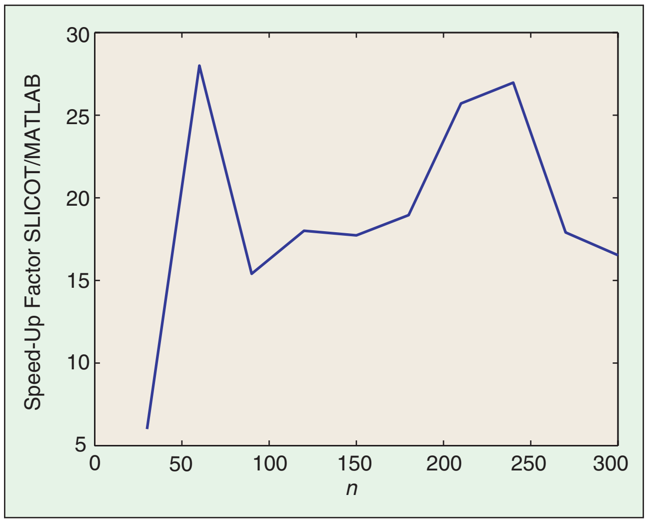


Figure 2. *Speed-up factor comparison: SLICOT* `sllyap` *versus MATLAB* `lyap`*. Random continuous-time Lyapunov equations with A in real Schur form and* $n = 30 : 30 : 300$ *are solved. By exploiting the problem structure, SLICOT* `sllyap` *is 16 to 28 times faster than* `lyap`*.*

using Compaq Visual Fortran V5.1, nonoptimized BLAS, and MATLAB 6.1 (R12). Other results, obtained on a SUN Ultra 2 Creator 2200 workstation with 128 MB RAM and operating system SunOS 5.5, by calling from MATLAB the gateways produced by the NAGWare Gateway Generator for the corresponding SLICOT codes are given in [1]. These results show that SLICOT routines usually outperform MATLAB calculations. While the accuracy is comparable, and frequently better, the gain in efficiency by calling SLICOT routines can be significant. Even better efficiency is to be expected by calling the SLICOT routines directly in Fortran (not through MATLAB gateways) and using optimized BLAS libraries.

Figure 1 shows the ratio of the execution times (speed-up factor) for the SLICOT gateway function `sllyap` (calling the SLICOT Lyapunov equation solvers) and MATLAB function `lyap`, for solving randomly generated continuous-time Lyapunov equations with known solutions, and $A \in \mathbb{R}^{n\times n}$, for $n = 10 : 10 : 300$. It can be observed that the speed-up is four or larger up to dimension 50 and always larger than two for larger dimensions. To explain these results, we note that the most expensive step in the computational algorithm of [29] is the reduction to RSF of the coefficient matrix. This computation is performed in both cases by using highly efficient LAPACK codes. The complexification of computations (see above) is performed only during the backsubstitution step, which leads to at least a doubling of the computational times for larger dimensions. To measure the net effect in performance loss of the complexification, we solved Lyapunov equations whose coefficient matrices were already in RSF. Figure 2 shows the resulting speed-up factors for increasing dimensions ($n = 30 : 30 : 300$). Clearly, the SLICOT function is faster (between 16 and 28 times faster) since it can exploit the problem structure.

Figures 3 and 4 plot the speed-up factor for the SLICOT gateway function `slconf` and MATLAB function `ctrbf`, for computing the controllability staircase form of a random system (A, B, C), for state vector dimensions $n = 30 : 30 : 300$, and input and output vector dimensions of $m = p = n/15$, and $m = p = 1$, respectively. In the second case, the speed-up factors of `slconf` over `ctrbf` are much larger, between 5 and 45. Note that rank decisions are based on the rank-revealing QR factorization with pivoting and incremental condition estimation, in `slconf`, and on the singular value decomposition, in `ctrbf`.

Figure 5 shows the speed-up factor for the SLICOT system identification function `slmoen` (combined MOESP and N4SID techniques), using the structure-exploiting fast QR algorithm [15], and the latest `n4sid` implementation [30], based on a standard QR factorization; default options have been used for `n4sid`. The 22 input–output data sets used, also considered in [31], include mainly the sets from the DAISY collection, freely available at the site www.esat.kuleuven.ac.be/sista/daisy, which contains some large data sequences from various domains. Application 1 (Simulation of an ethane-ethylene distillation column) and Application 18 (Simulation of the western basin of Lake Erie) consist of several data batches and cannot be directly dealt with by `n4sid`; Application 16 (Steel subframe flexible structure, with two inputs, 28 outputs, and 8,523 data samples) could not be solved by `n4sid`, since an "Out of memory" error appeared. All SLICOT system identification codes successfully solved the identification problems for these applications. The problem for Application 16 seems to be too large for `n4sid`, at least when using a standard PC or Sun workstation. The matrix whose R factor in a QR factorization should be computed has about 8,500 rows (if all data are used for identification) and over 1,700 columns. The SLICOT codes could exploit the block-Hankel-block structure of that matrix, and efficiently obtain the needed R factor.

User-Friendly Interfaces to SLICOT

There are essentially two ways to provide easy-to-use interfaces for computational routines like those available in SLICOT:

1) Provide gateways to existing problem-solving environments. The conventional way to simplify the interface by using new language constructs, such as those available in Fortran 90, can be combined with building interfaces or gateways to environments such as MATLAB or the free software. Developing suitable gateway functions provides the user with seamless access to the powerful computational routines available in SLICOT and to build interactive graphical or command language-based interfaces for specifying and solving computational control problems. In the first subsection we describe the collection of prototype MEX- and M-function gateways developed as part of the SLICOT implementation project, which can be used for testing purposes and for solving computational control problems.
2) Build new computational environments in which parts of the SLICOT library have been fully integrated, as described in the last two subsections. We first briefly present the free software Scilab and the substantial computational enhancements achieved by integrating SLICOT into this package. Next, we describe the Descriptor Systems Toolbox for MATLAB, built around the SLICOT computational routines, to illustrate a new paradigm for developing high-performance CACSD tools.

MATLAB Gateways

The essential functionality and performance of SLICOT routines have been made accessible from the high-level software environments MATLAB and Scilab by means of a large collection of MEX- and M-function gateways representing user-friendly interfaces to the main user-callable Fortran routines available in SLICOT. These MEX- and M-functions can be seen as prototype gateway software offering a rich functionality and flexibility for solving control-related computational problems.

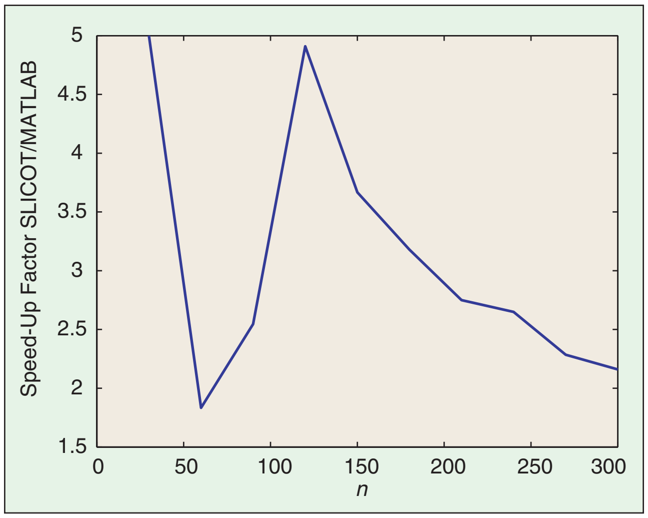


Figure 3. *Speed-up factor comparison: SLICOT* `slconf` *versus MATLAB* `ctrbf`*. The controllability staircase form is computed for random continuous-time systems with* $n = 30:30:300$ *and* $m = p = n/15$*. The function* `slconf` *is faster since it uses the rank-revealing QR factorization without accumulating the transformations, while* `ctrbf` *computes a full singular value decomposition at each step of the algorithm.*

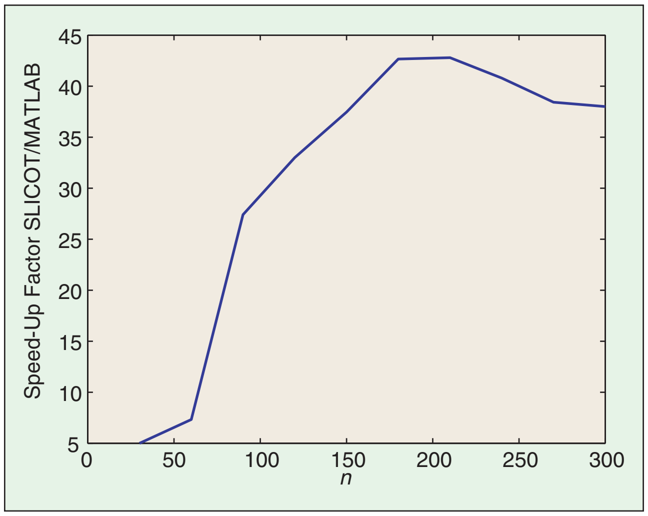


Figure 4. *Speed-up factor comparison: SLICOT* `slconf` *versus MATLAB* `ctrbf`*. The controllability staircase form is computed for random continuous-time systems with* $n = 30:30:300$ *and* $m = p = 1$*. The function* `slconf` *is much faster since it is significantly cheaper to apply Householder transformations than to compute the full singular value decomposition of column vectors, as is done in* `ctrbf`*.*

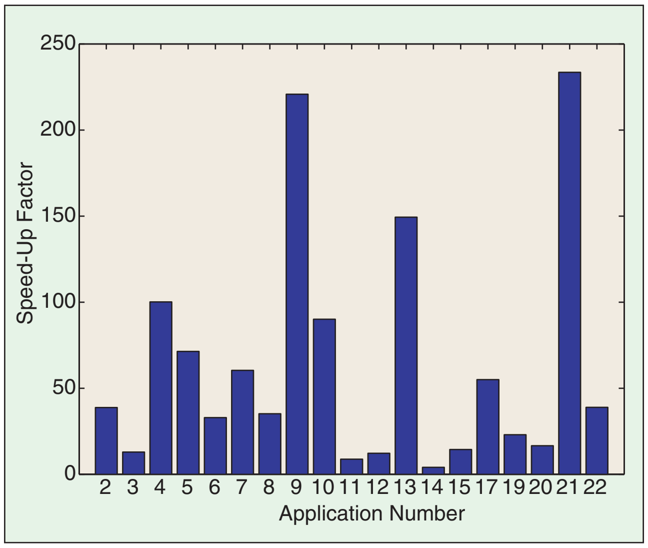


Figure 5. *Speed-up factor comparison: SLICOT* `slmoen4` *versus MATLAB 6.1* `n4sid` *with default options. Data sets from the DAISY collection are used. The function* `slmoen4` *employs a structure-exploiting fast QR factorization, revealing a clear benefit over* `n4sid`*, which uses the standard QR factorization.*

The MEX-functions represent the basic interface to the SLICOT computational tools. Typically, each MEX-function interface has an extended scope by providing full access to the complete functionality of several user-callable routines. The MEX interfaces are necessarily quite complex, and therefore primarily intended for expert use and further software developments. To allow a user-friendly operation, easy-to-use M-functions have also been implemented, where specific aspects are exploited by using system objects defined in the Control Toolbox or providing functionality covering a particular class of algorithms or problems. In many cases, several M-functions have been implemented as higher-level interfaces to a single MEX-function. For instance, the MEX-function `aresol` calls the SLICOT routines `SB02MD`, `SB02MT`, `SB02ND`, and `SB02OD` for solving either continuous-time or discrete-time algebraic Riccati equations (CARE/DARE) using standard or generalized Schur vector methods, while the higher-level M-functions `slcaregs`, `slcares`, `sldaregs`, `sldares`, and `sldaregsv` call `aresol` with appropriate settings to solve a specific CARE or DARE using a certain method.

The SLICOT MEX-function gateways are implemented as Fortran 90 subroutines. The main benefit of using Fortran 90 (instead of Fortran 77) is the use of allocatable arrays to reduce storage requirements. For MEX-functions overloaded with several functionalities, special input arguments are used to select the user or method options (for example, to indicate the type of the system or scaling options). Frequently, optional input arguments are allowed, and the absence of some output arguments indicates the possibility of avoiding unnecessary computations such as evaluating internal transformation matrices. All MEX-functions perform an exhaustive test of input parameters for their types, shapes, and dimensions. Error or warning messages are issued to indicate incorrect input-output arguments or computational failures.

The current version of the SLICOT library is accompanied by 50 MEX-functions covering several main computational areas. These include the reduction of system matrices to condensed forms, system scaling, minimal realization, solution of various matrix equations (Lyapunov, Sylvester, Riccati), coprime factorization and additive spectral decomposition of transfer-function matrices, computation of system norms, pole assignment, solving linear equations with structured matrices (for example, Toeplitz), model and controller reduction, linear and Wiener system identification, structured-singular value, and $\mathcal{H}_\infty/\mathcal{H}_2$ robust synthesis. The collection of M-functions is based on the provided MEX-functions and covers the same functionality but in a more method-oriented way. There are currently 238 M-functions (including help and test files). Many of the M-functions can be seen as enhancements of similar functions available in the Control Toolbox of MATLAB. Other functions represent unique pieces of software, which are available for free. This software includes the large collection of functions covering frequency-weighted model and controller reduction [10], [32], or the powerful suite of subspace identification methods. Also worthwhile mentioning is the collection of benchmark examples for continuous- and discrete-time systems, as well as for Lyapunov- and Riccati-equation solvers.

Scilab Computational Environment

Scilab is an open-source, free, general-purpose scientific package for numerical applications. It provides a user-friendly environment for systems, control, and signal processing applications. Developed at INRIA and ENPC, originally as a research tool for testing and developing new control and optimization algorithms, Scilab is now widely distributed and used in academia and industry. Scilab is popular among students because they can access it for free at home, as well as among researchers and engineers who have contributed toolboxes that are freely available at Scilab's Web site scilab.org. Since the beginning, Scilab has been strongly tied to SLICOT and has incorporated many SLICOT routines. Scilab is currently developed under Linux, although it works on most operating systems including Windows, Mac, and Unix workstations.

Although the Scilab syntax is not identical to MATLAB syntax, it is a matrix-based language in which all basic vector-matrix operations are performed in the same way. Indeed, Scilab is similar to MATLAB in many respects (they are both inspired by the original public domain MATLAB program), and, in most control applications, Scilab can easily be used in place of MATLAB. Simulation is, of course, an essential task for validating and analyzing control laws, and most of the CACSD packages provide interactive simulation tools. Scilab has a toolbox called Scicos for modeling and simulation of dynamic systems through a block diagram editor. Both continuous and discrete-time systems can be modeled in Scicos. Moreover, Scicos formalism can handle events and thus, to some extent, hybrid system modeling and simulation.

For numerical packages such as Scilab, reliability and speed are the key issues. Regarding linear algebra algorithms, Scilab is now based on the LAPACK library, which is well tested, complete, and fast since LAPACK uses the BLAS library for which optimized code exists. For systems and control applications, SLICOT plays a role similar to LAPACK, and Scilab's basic control tools, such as the Lyapunov and the Riccati equation solvers, are based on SLICOT modules implemented as built-in primitives. The recently developed SLICOT identification programs are also available. The robust control toolbox also uses SLICOT programs. The nonlinear simulation programs that have been selected in SLICOT are also available in Scilab. Linear (control) systems are manipulated in an object-oriented manner. By operator overloading, these objects are treated the same way as ordinary matrices. For example, the series concatenation of two systems is denoted by S1*S2. Descriptor systems that require sophisticated algorithms are also handled.

In general, it is difficult to choose appropriate default values for the parameters that appear in the calling sequence of the basic primitives and are used to control error tolerance. Scilab, in concert with error estimates provided by SLICOT programs, facilitates this task by proposing appropriate default values for the parameters.

For most applications, it is not necessary to have access to the complete SLICOT library. In fact, since the SLICOT high-level algorithms are powerful, even a moderate subset of the SLICOT library can be useful for sophisticated control applications. For instance, the Kronecker decomposition of a possibly singular pencil (which is available in Scilab as a built-in SLICOT based primitive) can be used in various control algorithms such as calculating multivariable zeros or solving singular control problems. Consequently, general purpose routines of the SLICOT library have been included in Scilab, and facilities are provided for users to interface specific routines. In many cases, such routines can be emulated using a Scilab program without seriously affecting the efficiency. The advantage of such an approach is the readability and the adaptability of the code.

The Scilab Application Program Interface (API) was designed to facilitate the addition of new incrementally linked and interfaced routines. This interface is particularly useful for specific routines of the SLICOT library that are not linked by default. In particular, it is possible to pass a linear system as a single object to the interface program, which is often needed to exploit the particular structure of a matrix. The technique of interfacing programs in Scilab is slightly different from MATLAB. However, specific functions that emulate the MATLAB MEX API, and MATLAB MEX-files can also be used for interfacing programs. For example, the SLICOT toolbox developed for MATLAB in the NICONET project has been adapted with minor syntax modifications for use in the Scilab environment.

Descriptor Systems Toolbox

SLICOT provides a multitude of high-performance system-theoretic computational tools, which are not provided in standard CACSD tools such as the Control Toolbox of MATLAB. To illustrate these tools, consider a descriptor system of the form

$$\begin{aligned} E\dot{x}(t) &= Ax(t) + Bu(t), \\ y(t) &= Cx(t) + Du(t), \end{aligned}$$

with E square and possibly singular and with $A - \lambda E$ a regular matrix pencil, which is the most general description for a linear time-invariant, continuous-time system. Such systems arise when modeling interconnected systems, even with standard tools such as Simulink (recall the "algebraic loop" warning). Descriptor models are also common in modeling constrained mechanical systems (contact problems). Moreover, the descriptor representation is necessary to perform some operations even with standard systems such as conjugation or inversion. Discrete-time descriptor representations are frequently used to model economic processes.

The Descriptor Systems Toolbox was primarily intended to provide an extended functionality for the Control Toolbox of MATLAB, which formally supports descriptor systems, but not those with singular E. Consequently, some functions in the Descriptor Systems Toolbox simply represent extensions of functions already present in the Control Toolbox. Other functions are new and allow a convenient user-friendly environment for solving complicated dynamics analysis problems such as, for example, the determination of the complete Kronecker-structure of a linear pencil. Note also that the numerically reliable solution of many standard control problems relies on descriptor system techniques. Important examples are the solution of Riccati equations, computation of system zeros, and design of fault detection and isolation filters. The Descriptor Systems Toolbox extends the capabilities of basic MATLAB with matrix pencil methods, such as reordering of (generalized) Schur forms, computation of Kronecker-like forms, and solving generalized linear matrix equations.

The Descriptor Systems Toolbox is also useful for manipulating rational and polynomial matrices. Recall that each rational matrix $R(\lambda)$ can be seen as the transfer-function matrix of a continuous- or discrete-time descriptor system. Thus, each $R(\lambda)$ can be equivalently realized by a descriptor system quadruple $(A - \lambda E, B, C, D)$ satisfying

$$R(\lambda) = C(\lambda E - A)^{-1}B + D,$$

where $\lambda = s$ or $\lambda = z$ for a continuous- or discrete-time realization, respectively. It is widely accepted that most numerical operations on rational or polynomial matrices are best done by manipulating the matrices of the corresponding descriptor system representations. Many operations on standard matrices (such as finding the rank, determinant, inverse or generalized inverses) or the solution of linear matrix equations have natural generalizations for rational matrices. The conjugate transposition of a complex matrix generalizes to the conjugation of a rational matrix, while the full-rank, inner–outer, and spectral factorizations can be seen as generalizations of the familiar LU, QR, and Cholesky factorizations, respectively. Many problems for scalar polynomials and rational functions (poles and zeros, minimum degree or normalized coprime factorizations, and spectral factorization) have nontrivial extensions to polynomial and rational matrices.

The approach used to develop the Descriptor Systems Toolbox exploited MATLAB's matrix and object manipulation features of by means of a flexible and functionally rich collection of M-functions, intended for noncritical computations, while simultaneously enforcing highly efficient and numerically sound computations via MEX-functions (calling selected Fortran routines from LAPACK, SLICOT, and recent additions to RASP), to solve critical numerical problems by using structure-exploiting algorithms. The basic set of MEX-functions covers the following problems: computation of Kronecker-like forms, minimal realization of descriptor systems, generalized system similarity transformations, computation of generalized system zeros and Kronecker structure, stable/unstable and finite/infinite spectral separations, partial pole placement and solution of generalized Sylvester and Lyapunov matrix equations. An important aspect of the toolbox design was to ensure that standard systems with $E = I$ are fully supported, using specific algorithms. In the same vein, all algorithms are available for both continuous- and discrete-time systems.

The Descriptor Systems Toolbox supports the three basic system representations in the standard Control Toolbox: descriptor state space, rational, and pole/zero/gain representations. By function overloading, the same function performs, if appropriate, on all three representations. Automatic model conversions are performed when necessary and the results are provided in accordance with the original system representation. In contrast to the Control Toolbox, conversions always result in minimal representations.

Version 1.0 of the Descriptor Systems Toolbox is described in [33], where the underlying algorithms are also indicated and several examples illustrating the basic operations are given. For the contents of the current version of the toolbox (presently 1.04), see the Web site http://www.robotic.dlr.de/control/num/desctool.html. The implementations of all functions exploit the best of MATLAB and Fortran programming, by trying to balance the matrix manipulation power of MATLAB with the intrinsic high efficiency of carefully implemented structure-exploiting Fortran codes available in LAPACK and SLICOT. This approach illustrates the possibility of turning high complexity structure-exploiting algorithms into numerically robust and user-friendly CACSD software. This paradigm is applicable to the development of future computer-aided control engineering environments.

Future Directions

The following are the main objectives for future developments of SLICOT.

- *Develop control-oriented software for large-scale problems.* The need for such software is higher nowadays because of the increased use of high fidelity modeling in industrial practice. The mathematical models originating from discretization of partial differential equations are usually large-scale linearized models, whose constituent matrices are often sparse. Developing tools to perform analysis, order reduction, simulation, or controller design for large scale and possibly sparse system models will broaden the ability to solve industrial CACSD applications, where currently available packages fail because of large dimensions. Many computations based on sparse matrix techniques (for example, in model reduction), can best be done by implementing software that runs on parallel machines. The main advantages over the currently used dense matrix techniques are the improvement in memory usage and increased computational efficiency. Another direction is to develop new block-oriented algorithms for control-relevant computations to increase the computational performance on high-performance computer architectures.
- *Extend SLICOT to cover new industrially relevant areas.* Such areas include optimization-based control system design, approximation of large-scale and sparse control systems, simulation and control of differential-algebraic systems, and robust and computationally feasible methods for solving large-scale control problems, which occur in aerospace, automotive, and robotic applications.
- *Develop a Fortran 95 version of SLICOT.* The Fortran 95 language standard is significantly enhanced compared to Fortran 77 and is better adapted for exploiting high-performance computer architectures. Since

Fortran 95 is more frequently used by software developers, LAPACK, the underlying linear algebra software for SLICOT, has been upgraded to Fortran 95 by providing Fortran 95 interfaces by means of modules to the Fortran 77 version of the routines. The intended upgrading of SLICOT will benefit from the new features of the language such as array calculation, dynamic memory allocation, or flexible parameter lists (with optional and keyword arguments). Using the structure constructs available in the language, system objects similar to those in MATLAB and Scilab can also be easily defined and manipulated. These features will greatly facilitate the integration of SLICOT into CACSD environments.

- *Integration into user-friendly CACSD environments.* The final goal is to develop a self-contained and functionally rich Systems and Control Toolbox. Providing appropriate gateways to the de facto standard user-friendly CACSD environments, MATLAB and Scilab, will improve the computational facilities provided by these packages and will ensure an easy transfer of SLICOT software to industry. The main focus is to develop a self-contained collection of M-functions in MATLAB and Scilab, which integrate the available computational facilities in the SLICOT library. The M-functions will serve as high-level MATLAB/Scilab interfaces to a number of Fortran-based computational gateways calling SLICOT routines. The final product will be a powerful Systems and Control Toolbox, which combines the best of MATLAB/Scilab and Fortran, that is, the object-oriented matrix manipulation available in MATLAB and Scilab supported by the efficient and robust numerical computations provided by the structure-exploiting algorithms available in SLICOT. A first step in this direction has already been achieved by implementing the Descriptor Systems Toolbox [33].

Acknowledgments

This work was supported in part by the European Community BRITE-EURAM III Thematic Networks Programme NICONET (project BRRT-CT97-5040), the Belgian Programme on Interuniversity Poles of Attraction, initiated by the Belgian State, Prime Minister's Office for Science, Technology and Culture (IUAP Phase V-22), the Concerted Action Project MEFISTO-666 (Mathematical Engineering for Information and Communications Technology) of the Flemish Community, and the FWO projects G.0078.01 (structured matrices) and G.0270.02 (nonlinear Lp approximation).

References

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

[2] C.L. Lawson, R.J. Hanson, D.R. Kincaid, and F.T. Krogh, "Basic linear algebra subprograms for Fortran usage," *ACM Trans. Math. Softw.*, vol. 5, no. 3, pp. 308–323, 1979.

[3] 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.*, vol. 14, no. 1, pp. 1–32, 1988.

[4] J.J. Dongarra, J. Du Croz, I.S. Duff, and S. Hammarling, "A set of level 3 basic linear algebra subprograms," *ACM Trans. Math. Softw.*, vol. 16, no. 1, pp. 1–17, 1990.

[5] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide,* 3rd ed. Philadelphia, PA: SIAM, 1999.

[6] C. Gomez, Ed., *Engineering and Scientific Computing with SciLab*. Boston, MA: Birkhäuser, 1999.

[7] J.W. Demmel, J.J. Dongarra, and W. Kahan, "On designing portable high performance numerical libraries," in *Proc. Numerical Analysis 1991, Proc. 14th Dundee Conf.*, D.F. Griffiths and G.A. Watson, Eds., Essex, U.K.: Longman Scientific and Technical, 1992, vol. 260, pp. 69–84.

[8] C.F. Van Loan, "A symplectic method for approximating all the eigenvalues of a Hamiltonian matrix," *Linear. Alg. Appl.*, vol. 61, pp. 233–251, Sept. 1984.

[9] S.J. Hammarling, "Numerical solution of the stable, non-negative definite Lyapunov equation," *IMA J. Numer. Anal.*, vol. 2, no. 3, pp. 303–323, 1982.

[10] A. Varga, "Model reduction software in the SLICOT library," in *Applied and Computational Control, Signals, and Circuits*, B.N. Datta, Ed., Boston: Kluwer, 2001, vol. 2, pp. 239–282.

[11] A. Varga, "Periodic Lyapunov equations: Some applications and new algorithms," *Int. J. Contr.*, vol. 67, no. 1, pp. 69–87, 1997.

[12] P. Misra, P. Van Dooren, and A. Varga, "Computation of structural invariants of generalized state-space systems," *Automatica*, vol. 30, no. 12, pp. 1921–1936, 1994.

[13] P. Benner, "Contributions to the numerical solution of algebraic Riccati equations and related eigenvalue problems," dissertation, Fakultät für Mathematik, Technische Universität Chemnitz-Zwickau, Germany, Feb. 1997.

[14] V. Sima, *Algorithms for Linear-Quadratic Optimization*. New York: Marcel Dekker, 1996.

[15] N. Mastronardi, D. Kressner, 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.*, vol. 132, no. 1, pp. 71–81, 2001.

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

[17] G. Grübel, H.-D. Joos, M. Otter, and R. Finsterwalder, "The ANDECS design environment for control engineering," in *Proc. 12th IFAC World Congress*, Sydney, Australia, 1993.

[18] J.H. Wilkinson and C. Reinsch, Eds., *Handbook for Automatic Computation, vol. 2, Linear Algebra*. Berlin, Germany: Springer-Verlag, 1971.

[19] B.T. Smith, J.M. Boyle, J.J. Dongarra, B.S. Garbow, Y. Ikebe, V.C. Klema, and C.B. Moler, *Matrix Eigensystem Routines—EISPACK Guide* 2nd ed. New York: Springer, 1976.

[20] B.S. Garbow, J.M. Boyle, J.J. Dongarra, and C.B. Moler, *Matrix Eigensystem Routines—EISPACK Guide Extension.* Berlin: Springer-Velag, 1977.

[21] J.J. Dongarra, J.R. Bunch, C.B. Moler, and G.W. Stewart, *LINPACK User's Guide*. Philadelphia, PA: SIAM, 1979.

[22] The Working Group on Software: WGS, "An inventory of basic software for computer aided control system design," WGS-report 85-1, 1985.

[23] T. Popescu, V. Sima, A. Varga, and C. Vasiliu, "Program package for identification and control systems design," in *Computer Aided Design of Control Systems*, M.A. Cuenod, Ed. New York: Pergamon Press, 1980.

[24] A. Varga and V. Sima, "BIMAS—A basic mathematical package for computer aided systems analysis and design," in *Prepr. 9th IFAC World Congr.*, Budapest, Hungary, 1985, vol. 8, pp. 202–207.

[25] A. Varga and A. Davidoviciu, "BIMASC—A package of Fortran subprograms for analysis, modelling, design and simulation of control systems," in *Prepr. 3rd IFAC/IFIP Int. Symp. Computer Aided Design in Control and Engineering Systems (CADCE'85)*, Copenhagen, Denmark, 1985, pp. 151–156.

[26] P.H. Petkov, N.D. Christov, and M.M. Konstantinov, "SYSLAB: An interactive system for analysis and design of linear multivariable systems," in *Prepr. 3th IFAC/IFIP Int. Symposium on Computer Aided Design in Control and Engineering Systems (CADCE'85)*, Copenhagen, Denmark, 1985, pp. 140–145.

[27] The Working Group on Software: WGS, *SLICOT Implementation and Documentation Standards 2.1* WGS-report 96-1, 1996 [Online]. Available: ftp://wgs.esat.kuleuven.ac.be/pub/WGS/REPORTS, file rep96-1.ps.Z

[28] S. Van Huffel and V. Sima, "SLICOT and control systems numerical software packages," in *Proc. 2002 IEEE Int. Conf. Control Applications and IEEE Int. Symp. Computer Aided Control System Design, CCA/CACSD 2002*, Glasgow, U.K., 2002, pp. 39–44.

[29] R.H. Bartels and G.W. Stewart, "Algorithm 432: Solution of the matrix equation $AX + XB = C$," *Commun. ACM*, vol. 15, no. 9, pp. 820–826, 1972.

[30] L. Ljung, *System Identification Toolbox for Use with MATLAB. User's Guide, Version 5*. Natick, MA: MathWorks, 2000.

[31] V. Sima, D.M. Sima, and S. Van Huffel, "SLICOT system identification software and applications," in *Proc. 2002 IEEE Int. Conf. Control Applications and IEEE Int. Symp. Computer Aided Control System Design, CCA/CACSD 2002*, Glasgow, U.K., 2002, pp. 45–50.

[32] A. Varga, "Numerical software in SLICOT for low order controller design," in *Proc. 2002 IEEE Int. Conf. Control Applications and IEEE Int. Symp. Computer Aided Control System Design, CCA/CACSD 2002*, Glasgow, U.K., 2002, pp. 51–56.

[33] A. Varga, "A descriptor systems toolbox for MATLAB," in *Proc. CACSD 2000 Symposium*, Anchorage, AK, 2000, pp. 150–155.

Sabine Van Huffel (***Sabine.VanHuffel@esat.kuleuven.ac.be***) is a full professor at the Department of Electrical Engineering from the Katholieke Universiteit Leuven, Leuven, Belgium. Her research interests are in numerical linear algebra, errors-in-variables regression, system identification, pattern recognition, (non)linear modeling, numerically reliable software for systems and control, parameter estimation, and signal processing. In these areas, she has authored one book and more than 90 papers in international journals and 120 conference contributions. She was the central coordinator of the European thematic Numerics in Control network NICONET (1996–2002) and is chair of the numerics in control international society NICONET. She can be contacted at the Department of Electrical Engineering (ESAT), Katholieke Universiteit Leuven, Kasteelpark Arenberg 10, B–3001 Leuven–Heverlee, Belgium.

Vasile Sima is a first-degree senior researcher at the National Institute for Research and Development in Informatics, Bucharest, Romania. He received the dipl. ing. and Dr. ing. degrees (control engineering) from Polytechnical Institute of Bucharest in 1972 and 1983, and the dipl. math. degree from Bucharest University in 1978. His research interests include control system design, system identification, and scientific computations. He authored the book *Algorithms for Linear-Quadratic Optimization* (Marcel Dekker, 1996), and coauthored many papers. He has been the SLICOT librarian since 1996. He received a Romanian Academy Award and is a Senior Member of IEEE and a member of AMS.

Andras Varga is a senior scientist at the German Aerospace Center in Oberpfaffenhofen. He received the diploma in control engineering in 1974 and the Ph.D. degree in electrical engineering in 1981, both from the University "Politechnica" of Bucharest, Romania. His main research interests are in developing reliable numerical methods and robust numerical software for computer aided control system design (CACSD). He is a Fellow of the IEEE and past associate editor of *IEEE Transactions on Automatic Control*. He was program chair of the 1999 Symposium on CACSD and general chair of the 2000 Symposium on CACSD. Since 2000, he has been chair of the Technical Committee on CACSD within the IEEE Control Systems Society. For 2002–2003 he was a nominated member of the Board of Governors of the Control Systems Society. He received a research fellowship award from the Alexander von Humboldt Foundation. He coauthored three books, coedited one book, and published over 130 journal and conference publications.

Sven Hammarling is a principal consultant in the Development Division of The Numerical Algorithms Group Ltd., Oxford, U.K., and is a visiting professor at Cranfield University, RMCS, Shrivenham, U.K. His interests include numerical linear algebra, high-performance computing, and portable numerical software and associated standards. He is one of the authors of the Level 2 and 3 Basic Linear Algebra Subprograms, as well as the linear algebra software LAPACK and ScaLAPACK. He is an associate editor for *ACM Transactions on Mathematical Software*.

François Delebecque received a Ph.D. in mathematics from the University of Paris Dauphine in 1976. He is currently senior scientist at INRIA, the French National Research Institute in Automatics and Computer Science. He has been an associate editor of *Automatica*. His main interests are in numerical analysis and the development of numerical tools for control and signal processing. He is one of the developers of the open source CACSD package Scilab.