Encyclopedia of Systems and Control
DOI 10.1007/978-1-4471-5102-9_143-1

Basic Numerical Methods and Software for Computer Aided Control Systems Design

Volker Mehrmann[a]* and Paul Van Dooren[b]
[a]Institut für Mathematik MA 4-5, Technische Universität Berlin, Berlin, Germany
[b]ICTEAM: Department of Mathematical Engineering, Catholic University of Louvain, Louvain-la-Neuve, Belgium

Introduction

Basic numerical methods for the analysis and design of dynamical systems are at the heart of most techniques in systems and control theory that are used to describe, control, or optimize industrial and economical processes. There are many methods available for all the different tasks in systems and control, but even though most of these methods are based on sound theoretical principles, many of them still fail when applied to real-life problems. The reasons for this may be quite diverse, such as the fact that the system dimensions are very large, that the underlying problem is very sensitive to small changes in the data, or that the method lacks numerical robustness when implemented in a finite precision environment.

To overcome such failures, major efforts have been made in the last few decades to develop robust, well-implemented, and standardized software packages for computer-aided control systems design (Grübel 1983; Nag Slicot 1990; Wieslander 1977). Following the standards of modern software design, such packages should consist of numerically robust routines with known performance in terms of reliability and efficiency that can be used to form the basis of more complex control methods. Also to avoid duplication and to achieve efficiency and portability to different computational environments, it is essential to make maximal use of the established standard packages that are available for numerical computations, e.g., the **B**asic **L**inear **A**lgebra **S**ubroutines (BLAS) (Dongarra et al. 1990) or the **L**inear **A**lgebra **Pack**ages (LAPACK) (Anderson et al. 1992). On the basis of such standard packages, the next layer of more complex control methods can then be built in a robust way.

In the late 1980s, a working group was created in Europe to coordinate efforts and integrate and extend the earlier software developments in systems and control. Thanks to the support of the European Union, this eventually led to the development of the **S**ubroutine **L**ibrary **i**n **Co**ntrol **T**heory (SLICOT) (Benner et al. 1999; SLICOT 2012). This library contains most of the basic computational methods for control systems design of linear time-invariant control systems.

An important feature of this and similar kind of subroutine libraries is that the development of further higher level methods is not restricted by specific requirements of the languages or data structures used and that the routines can be easily incorporated within other more user-friendly software systems (Gomez et al. 1997; MATLAB 2013). Usually, this *low-level reusability* can only be achieved by using a general-purpose programming language like C or Fortran.

We cannot present all the features of the SLICOT library here. Instead, we discuss its general philosophy in section “The Control Subroutine Library SLICOT” and illustrate these concepts

*E-mail: mehrmann@math.tu-berlin.de

DOI 10.1007/978-1-4471-5102-9_143-1

in section "An Illustration" using one specific task, namely, checking the controllability of a system. We refer to SLICOT (2012) for more details on SLICOT and to Varga (2004) for a general discussion on numerical software for systems and control.

The Control Subroutine Library SLICOT

When designing a subroutine library of basic algorithms, one should make sure that it satisfies certain basic requirements and that it follows a strict standardization in implementation and documentation. It should also contain standardized test sets that can be used for benchmarking, and it should provide means for maintenance and portability to new computing environments. The subroutine library SLICOT was designed to satisfy the following basic recommendations that are typically expected in this context (Benner et al. 1999).

Robustness: A subroutine must either return reliable results or it must return an error or warning indicator, if the problem has not been well posed or if the problem does not fall in the class to which the algorithm is applicable or if the problem is too ill-conditioned to be solved in a particular computing environment.

Numerical stability and accuracy: Subroutines are supposed to return results that are as good as can be expected when working at a given precision. They also should provide an option to return a parameter estimating the accuracy actually achieved.

Efficiency: An algorithm should never be chosen for its speed if it fails to meet the usual standards of robustness, numerical stability, and accuracy, as described above. Efficiency must be evaluated, e.g., in terms of the number of floating-point operations, the memory requirements, or the number and cost of iterations to be performed.

Modern computer architectures: The requirements of modern computer architectures must be taken into account, such as shared or distributed memory parallel processors, which are the standard environments of today. The differences in the various architectures may imply different choices of algorithms.

Comprehensive functional coverage: The routines of the library should solve control systems relevant computational problems and try to cover a comprehensive set of routines to make it functional for a wide range of users. The SLICOT library covers most of the numerical linear algebra methods needed in systems analysis and synthesis problems for standard and generalized state space models, such as Lyapunov, Sylvester, and Riccati equation solvers, transfer matrix factorizations, similarity and equivalence transformations, structure exploiting algorithms, and condition number estimators.

The implementation of subroutines for a library should be highly standardized, and it should be accompanied by a well-written online documentation as well as a user manual (see, e.g., standard Denham and Benson 1981; Working Group Software 1996) which is compatible with that of the LAPACK library (Anderson et al. 1992). Although such highly restricted standards often put a heavy burden on the programmer, it has been observed that it has a high importance for the reusability of software and it also has turned out to be a very valuable tool in teaching students how to implement algorithms in the context of their studies.

DOI 10.1007/978-1-4471-5102-9_143-1

Benchmarking

In the validation of numerical software, it is extremely important to be able to test the correctness of the implementation as well as the performance of the method, which is one of the major steps in the construction of a software library. To achieve this, one needs a standardized set of benchmark examples that allows an evaluation of a method with respect to correctness, accuracy, and efficiency and to analyze the behavior of the method in extreme situations, i.e., on problems where the limit of the possible accuracy is reached. In the context of basic systems and control methods, several such benchmark collections have been developed (see, e.g., Benner et al. 1997; Frederick 1998, or http://www.slicot.org/index.php?site=benchmarks).

Maintenance, Open Access, and Archives

It is a major challenge to maintain a well-developed library accessible and usable over time when computer architectures and operating systems are changing rapidly, while keeping the library open for access to the user community. This usually requires financial resources that either have to be provided by public funding or by licensing the commercial use.

In the SLICOT library, this challenge has been addressed by the formation of the Niconet Association (http://www.niconet-ev.info/en/) which provides the current versions of the codes and all the documentations. Those of Release 4.5 are available under the GNU General Public License or from the archives of http://www.slicot.org/.

An Illustration

To give an illustration for the development of a basic control system routine, we consider the specific problem of checking controllability of a linear time-invariant control system. A linear time-invariant control problem has the form

$$\frac{dx}{dt} = Ax + Bu,\ t \in [t_0, \infty) \tag{1}$$

Here x denotes the *state* and u the *input function*, and the system matrices are typically of the form $A \in \mathbb{R}^{n,n}$, $B \in \mathbb{R}^{n,m}$.

One of the most important topics in control is the question whether by an appropriate choice of input function $u(t)$ we can control the system from an arbitrary state to the null state. This property, called *controllability*, can be characterized by one of the following equivalent conditions (see Paige 1981).

Theorem 1 *The following are equivalent:*

(i) *System (1) is controllable.*
(ii) *Rank* $[B, AB, A^2B, \cdots, A^{n-1}B] = n$.
(iii) *Rank* $[B, A - \lambda I] = n \quad \forall \lambda \in \mathbb{C}$.
(iv) $\exists F$ *such that* A *and* $A + BF$ *have no common eigenvalues.*

DOI 10.1007/978-1-4471-5102-9_143-1

The conditions of Theorem 1 are nice for theoretical purposes, but none of them is really adequate for the implementation of an algorithm that satisfies the requirements described in the previous section. Condition (ii) creates difficulties because the controllability matrix $K = [B, AB, A^2B, \cdots, A^{n-1}B]$ will be highly corrupted by roundoff errors. Condition (iii) can simply not be checked in finite time. However, it is sufficient to check this condition only for the eigenvalues of A, but this is extremely expensive. And finally, condition (iv) will almost always give disjoint spectra between A and $A + BF$ since the computation of eigenvalues is sensitive to roundoff.

To devise numerical procedures, one often resorts to the computation of canonical or condensed forms of the underlying system. To obtain such a form one employs controllability preserving linear transformations $x \mapsto Px$, $u \mapsto Qu$ with nonsingular matrices $P \in \mathbb{R}^{n,n}$, $Q \in \mathbb{R}^{m,m}$. The canonical form under these transformations is the Luenberger form (see Luenberger 1967). This form allows to check the controllability using the above criterion (iii) by simple inspection of the condensed matrices. This is ideal from a theoretical point of view but is very sensitive to small perturbations in the data, in particular because the transformation matrices may have arbitrary large norm, which may lead to large errors.

For the implementation as robust numerical software one uses instead transformations with real orthogonal matrices P, Q that can be implemented in a backward stable manner, i.e., the resulting backward error is bounded by a small constant times the unit roundoff $\mathbf{u}$ of the finite precision arithmetic, and employs for reliable rank determinations the well-known singular value decomposition (SVD) (see, e.g., Golub and Van Loan 1996).

Theorem 2 (Singular value decomposition) *Given $A \in \mathbb{R}^{n,m}$, then there exist orthogonal matrices U, V with $U \in \mathbb{R}^{n,n}$, $V \in \mathbb{R}^{m,m}$, such that $A = U\Sigma V^T$ and $\Sigma \in \mathbb{R}^{n,m}$ is quasi-diagonal, i.e.,*

$$\Sigma = \begin{bmatrix} \Sigma_r & 0 \\ 0 & 0 \end{bmatrix} \text{ where } \Sigma_r = \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_r \end{bmatrix},$$

and the nonzero singular values σ_i are ordered as $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$.

The SVD presents the *best* way to determine (numerical) ranks of matrices in finite precision arithmetic by counting the number of singular values satisfying $\sigma_j \geq \mathbf{u}\sigma_1$ and by putting those for which $\sigma_j < \mathbf{u}\sigma_1$ equal to zero. The computational method for the SVD is well established and analyzed, and it has been implemented in the LAPACK routine `SGESVD` (see http://www.netlib.org/lapack/). A faster but less reliable alternative to compute the numerical rank of a matrix A is its QR factorization with pivoting (see, e.g., Golub and Van Loan 1996).

Theorem 3 (QRE decomposition) *Given $A \in \mathbb{R}^{n,m}$, then there exists an orthogonal matrix $Q \in \mathbb{R}^{n,n}$ and a permutation $E \in \mathbb{R}^{m,m}$, such that $A = QRE^T$ and $R \in \mathbb{R}^{n,m}$ is trapezoidal, i.e.,*

$$R = \begin{bmatrix} r_{11} & \dots & r_{1l} & \dots & r_{1m} \\ & \ddots & & & \vdots \\ & & r_{ll} & \dots & r_{lm} \\ 0 & & & 0 & \end{bmatrix}.$$

and the nonzero diagonal entries r_{ii} are ordered as $r_{11} \geq \cdots \geq r_{ll} > 0$.

DOI 10.1007/978-1-4471-5102-9_143-1

The (numerical) rank in this case is again obtained by counting the diagonal elements $r_{ii} \geq \mathbf{u}r_{11}$.

One can use such orthogonal transformations to construct the controllability staircase form (see Van Dooren 1981).

Theorem 4 (Staircase form) *Given matrices $A \in R^{n,n}$, $B \in \mathbb{R}^{n,m}$, then there exist orthogonal matrices P, Q with $P \in \mathbb{R}^{n,n}$, $Q \in \mathbb{R}^{m,m}$, so that*

$$PAP^T = \left[\begin{array}{cccc|c} A_{11} & \cdots & \cdots & A_{1,r-1} & A_{1,r} \\ A_{21} & \ddots & & \vdots & \vdots \\ & \ddots & \ddots & \vdots & \vdots \\ & & A_{r-1,r-2} & A_{r-1,r-1} & A_{r-1,r} \\ \hline 0 & \cdots & 0 & 0 & A_{rr} \end{array}\right] \begin{matrix} n_1 \\ n_2 \\ \vdots \\ n_{r-1} \\ n_r \end{matrix}$$
$$\begin{matrix} n_1 & \ldots & n_{r-2} & n_{r-1} & n_r \end{matrix}$$

$$PBQ = \left[\begin{array}{cc} B_1 & 0 \\ 0 & 0 \\ \vdots & \vdots \\ \vdots & \vdots \\ 0 & 0 \end{array}\right] \begin{matrix} n_1 \\ n_2 \\ \vdots \\ \vdots \\ n_r \end{matrix} \qquad (2)$$
$$\begin{matrix} n_1 & m-n_1 \end{matrix}$$

where $n_1 \geq n_2 \geq \cdots \geq n_{r-1} \geq n_r \geq 0, n_{r-1} > 0$, $A_{i,i-1} = \begin{bmatrix} \Sigma_{i,i-1} & 0 \end{bmatrix}$, with nonsingular blocks $\Sigma_{i,i-1} \in \mathbb{R}^{n_i,n_i}$ and $B_1 \in \mathbb{R}^{n_1,n_1}$.

Notice that when using the reduced pair in condition (iii) of Theorem 1, the controllability condition is just $n_r = 0$, which is simply checked by inspection. A numerically stable algorithm to compute the staircase form of Theorem 4 is given below. It is based on the use of the singular value decomposition, but one could also have used instead the QR decomposition with column pivoting.

Staircase Algorithm
Input: $A \in \mathbb{R}^{n,n}$, $B \in \mathbb{R}^{n,m}$
Output: PAP^T, PBQ in the form (2), P, Q orthogonal

Step 0: Perform an SVD $B = U_B \begin{bmatrix} \Sigma_B & 0 \\ 0 & 0 \end{bmatrix} V_B^T$ with nonsingular and diagonal $\Sigma_B \in \mathbb{R}^{n_1,n_1}$. Set $P := U_B^T$, $Q := V_B$, so that

$$A := U_B^T A U_B = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix},$$

$$B := U_B^T B V_B = \begin{bmatrix} \Sigma_B & 0 \\ 0 & 0 \end{bmatrix}$$

with A_{11} of size $n_1 \times n_1$.

Step 1: Perform an SVD $A_{21} = U_{21} \begin{bmatrix} \Sigma_{21} & 0 \\ 0 & 0 \end{bmatrix} V_{21}^T$ with nonsingular and diagonal $\Sigma_{21} \in \mathbb{R}^{n_2,n_2}$. Set

DOI 10.1007/978-1-4471-5102-9_143-1

$$P_2 := \begin{bmatrix} V_{21}^T & 0 \\ 0 & U_{21}^T \end{bmatrix}, \; P := P_2 P$$

so that

$$A := P_2 A P_2^T =: \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ 0 & A_{32} & A_{33} \end{bmatrix},$$

$$B := P_2 B =: \begin{bmatrix} B_1 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix},$$

where $A_{21} = [\Sigma_{21} \quad 0]$, and $B_1 := V_{21}^T \Sigma_B$.
Step 2:
i = 3
DO WHILE ($n_{i-1} > 0$ AND $A_{i,i-1} \neq 0$).
Perform an SVD of $A_{i,i-1} = U_{i,i-1} \begin{bmatrix} \Sigma_{i,i-1} & 0 \\ 0 & 0 \end{bmatrix} V_{i,i-1}^T$ with
$\Sigma_{i,i-1} \in \mathbb{R}^{n_i, n_i}$ nonsingular and diagonal.
Set

$$P_i := \begin{bmatrix} I_{n_1} & & & & \\ & \ddots & & & \\ & & I_{n_{i-2}} & & \\ & & & V_{i,i-1}^T & \\ & & & & U_{i,i-1}^T \end{bmatrix}, \; P := P_i P,$$

so that

$$A := P_i A P_i^T =: \begin{bmatrix} A_{11} & & \cdots & & A_{1,i+1} \\ A_{21} & \ddots & & & A_{2,i+1} \\ & \ddots & \ddots & & \vdots \\ & & A_{i,i-1} & A_{i,i} & \vdots \\ 0 & & & A_{i+1,i} & A_{i+1,i+1} \end{bmatrix}$$

where $A_{i,i-1} = [\Sigma_{i,i-1} \quad 0]$.
$i := i + 1$
END
$r := i$

It is clear that this algorithm will stop with $n_i = 0$ or $A_{i,i-1} = 0$. In every step, the remaining block shrinks at least by 1 row/column, as long as Rank $A_{i,i-1} > 1$, so that the algorithm stops after maximally $n-1$ steps. It has been shown in Van Dooren (1981) that system (1) is controllable if and only if in the staircase form of (A, B) one has $n_r = 0$.

DOI 10.1007/978-1-4471-5102-9_143-1

It should be noted that the updating transformations P_i of this algorithm will affect previously created "stairs" so that the blocks denoted as $\Sigma_{i,i-1}$ will not be diagonal anymore, but their singular values are unchanged. This is critical in the decision about the controllability of the pair (A, B) since it depends on the numerical rank of the submatrices $A_{i,i-1}$ and B (see Demmel and Kågström 1993). Based on this and a detailed error and perturbation analysis, the Staircase Algorithm has been implemented in the SLICOT routine `AB01OD`, and it uses in the worst-case $\mathcal{O}(n^4)$ flops (a "flop" is an elementary floating-point operation $+, -, *$, or $/$). For efficiency reasons, the SLICOT routine `AB01OD` does not use SVDs for rank decisions, but QR decompositions with column pivoting. When applying the corresponding orthogonal transformations to the system without accumulating them, the complexity can be reduced to $\mathcal{O}(n^3)$ flops. It has been provided with error bounds, condition estimates, and warning strategies.

Summary and Future Directions

We have presented the SLICOT library and the basic principles for the design of such basic subroutine libraries. To illustrate these principles, we have presented the development of a method for checking controllability for a linear time-invariant control system. But the SLICOT library contains much more than that. It essentially covers most of the problems listed in the selected reprint volume (Patel et al. 1994). This volume contained in 1994 the state of the art in numerical methods for systems and control, but the field has strongly evolved since then. Examples of areas that were not in this volume but that are included in SLICOT are periodic systems, differential algebraic equations, and model reduction. Areas which still need new results and software are the control of large-scale systems, obtained either from discretizations of partial differential equations or from the interconnection of a large number of interacting systems. But it is unclear for the moment which will be the methods of choice for such problems. We still need to understand the numerical challenges in such areas, before we can propose numerically reliable software for these problems: the area is still quite open for new developments.

Cross-References

▸ CACSD Introduction
▸ CACSD Software Tools

Bibliography

Anderson E, Bai Z, Bischof C, Demmel J, Dongarra J, Du Croz J, Greenbaum A, Hammarling S, McKenney A, Ostrouchov S, Sorensen D (1995) LAPACK users' guide, 2nd edn. SIAM, Philadelphia. http://www.netlib.org/lapack/

Benner P, Laub AJ, Mehrmann V (1997) Benchmarks for the numerical solution of algebraic Riccati equations. Control Syst Mag 17:18–28

Benner P, Mehrmann V, Sima V, Van Huffel S, Varga A (1999) SLICOT-A subroutine library in systems and control theory. Appl Comput Control Signals Circuits 1:499–532

DOI 10.1007/978-1-4471-5102-9_143-1

Demmel JW, Kågström B (1993) The generalized Schur decomposition of an arbitrary pencil $A - \lambda B$: robust software with error bounds and applications. Part I: theory and algorithms. ACM Trans Math Softw 19:160–174

Denham MJ, Benson CJ (1981) Implementation and documentation standards for the software library in control engineering (SLICE). Technical report 81/3, Kingston Polytechnic, Control Systems Research Group, Kingston

Dongarra JJ, Du Croz J, Duff IS, Hammarling S (1990) A set of level 3 basic linear algebra subprograms. ACM Trans Math Softw 16:1–17

Frederick DK (1988) Benchmark problems for computer aided control system design. In: Proceedings of the 4th IFAC symposium on computer-aided control systems design, Bejing, pp 1–6

Golub GH, Van Loan CF (1996) Matrix computations, 3rd edn. The Johns Hopkins University Press, Baltimore

Gomez C, Bunks C, Chancelior J-P, Delebecque F (1997) Integrated scientific computing with scilab. Birkhäuser, Boston. https://www.scilab.org/

Grübel G (1983) Die regelungstechnische Programmbibliothek RASP. Regelungstechnik 31: 75–81

Luenberger DG (1967) Canonical forms for linear multivariable systems. IEEE Trans Autom Control 12(3):290–293

Paige CC (1981) Properties of numerical algorithms related to computing controllability. IEEE Trans Autom Control AC-26:130–138

Patel R, Laub A, Van Dooren P (eds) (1994) Numerical linear algebra techniques for systems and control. IEEE, Piscataway

The Control and Systems Library SLICOT (2012) The NICONET society. NICONET e.V. http://www.niconet-ev.info/en/

The MathWorks, Inc. (2013) MATLAB version 8.1. The MathWorks, Inc., Natick

The Numerical Algorithms Group (1993) NAG SLICOT library manual, release 2. The Numerical Algorithms Group, Wilkinson House, Oxford. Updates Release 1 of May 1990

The Working Group on Software (1996) SLICOT implementation and documentation standards 2.1. WGS-report 96-1. http://www.icm.tu-bs.de/NICONET/reports.html

Van Dooren P (1981) The generalized eigenstructure problem in linear system theory. IEEE Trans Autom Control AC-26:111–129

Varga A (ed) (2004) Special issue on numerical awareness in control. Control Syst Mag 24-1: 14–17

Wieslander J (1977) Scandinavian control library. A subroutine library in the field of automatic control. Technical report, Department of Automatic Control, Lund Institute of Technology, Lund