Numerical and Computational Issues in Linear Control and System Theory*

A.J. Laub
Electrical Engineering Department
University of California
Los Angeles, CA 90095-1594

R.V. Patel
Department of Electrical and Computer Engineering
University of Western Ontario
London, Ontario
Canada N6A 5B9

P.M. Van Dooren
Department of Mathematical Engineering
Université Catholique de Louvain
B-1348 Louvain-la-Neuve
Belgium

January 27, 2010

1 Introduction

This chapter provides a survey of various aspects of the numerical solution of selected problems in linear systems, control, and estimation theory. Space limitations preclude an exhaustive survey and extensive list of references. The interested reader is referred to [14],[4],[1],[10] for sources of additional detailed information.

Many of the problems considered in this chapter arise in the study of the "standard" linear model

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t) && (1)\\ y(t) &= Cx(t) + Du(t). && (2) \end{aligned}$$

Here, $x(t)$ is an n-vector of states, $u(t)$ is an m-vector of controls or inputs, and $y(t)$ is a p-vector of outputs. The standard discrete-time analog of Equations 1 and 2 takes the form

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k && (3)\\ y_k &= Cx_k + Du_k. && (4) \end{aligned}$$

*This material is based on a paper written by the same authors and published as pages 1-29 in Patel, R.V., Laub, A.J., and Van Dooren, P.M., Eds., Numerical Linear Algebra Techniques for Systems and Control, Selected Reprint Series, IEEE Press, New York, 1994,

Of course, considerably more elaborate models are also studied, including time-varying, stochastic, and nonlinear versions of the above, but these are not be discussed in this chapter. In fact, the above linear models are usually derived from linearizations of nonlinear models about selected nominal points.

The matrices considered here are, for the most part, assumed to have real coefficients and to be small (of order a few hundred or less) and dense, with no particular exploitable structure. Calculations for most problems in classical single-input, single-output control fall into this category. Large, sparse matrices or matrices with special, exploitable structures may involve significantly different concerns and methodologies than those to be discussed here.

The systems, control, and estimation literature is replete with *ad hoc* algorithms to solve the computational problems which arise in the various methodologies. Many of these algorithms work quite well on some problems (e.g., "small order" matrices) but encounter numerical difficulties, often severe, when "pushed" (e.g., on larger order matrices). The reason for this is that little or no attention has been paid to the way algorithms perform in "finite arithmetic," i.e., on a finite word length digital computer.

A simple example due to Moler and Van Loan [14](p. 649)[1] illustrates a typical pitfall. Suppose it is desired to compute the matrix e^A in single precision arithmetic on a computer which gives 6 decimal places of precision in the fractional part of floating-point numbers. Consider the case

$$A = \begin{bmatrix} -49 & 24 \\ -64 & 31 \end{bmatrix}$$

and suppose the computation is attempted with the Taylor series formula

$$e^A = \sum_{k=0}^{+\infty} \frac{1}{k!} A^k. \tag{5}$$

This is easily coded and it is determined that the first 60 terms in the series suffice for the computation, in the sense that terms for $k \geq 60$ of the order of 10^{-7} no longer add anything significant to the sum. The resulting answer is

$$\begin{bmatrix} -22.2588 & -1.43277 \\ -61.4993 & -3.47428 \end{bmatrix}.$$

Surprisingly, the true answer is (correctly rounded)

$$\begin{bmatrix} -0.735759 & 0.551819 \\ -1.47152 & 1.10364 \end{bmatrix}.$$

What happened here was that the intermediate terms in the series got very large before the factorial began to dominate. The 17th and 18th terms, for example, are of the order of 10^7 but of opposite signs so that the less significant parts of these numbers, while significant for the final answer, are "lost" because of the finiteness of the arithmetic.

For this particular example, various fixes and remedies are available. However, in more realistic examples one seldom has the luxury of having the "true answer" available so that it is not always easy simply to inspect or test a computed solution and determine that it is in error. Mathematical analysis (truncation of the series, in the example above) alone is simply not sufficient when a problem is analyzed or solved in finite arithmetic (truncation of the arithmetic). Clearly, a great deal of care must be taken.

[1]The page number indicates the location of the appropriate reprint in [14].

The finiteness inherent in representing real or complex numbers as floating-point numbers on a digital computer manifests itself in two important ways: floating-point numbers have only finite precision and finite range. The degree of attention paid to these two considerations distinguishes many reliable algorithms from more unreliable counterparts.

The development in systems, control, and estimation theory, of stable, efficient, and reliable algorithms which respect the constraints of finite arithmetic began in the 1970s and is continuing. Much of the research in numerical analysis has been directly applicable, but there are many computational issues in control (e.g., the presence of hard or structural zeros) where numerical analysis does not provide a ready answer or guide. A symbiotic relationship has developed, especially between numerical linear algebra and linear system and control theory, which is sure to provide a continuing source of challenging research areas.

The abundance of numerically fragile algorithms is partly explained by the following observation:

> If an algorithm is amenable to "easy" hand calculation, it is probably a poor method if implemented in the finite floating-point arithmetic of a digital computer.

For example, when confronted with finding the eigenvalues of a 2×2 matrix, most people would find the characteristic polynomial and solve the resulting quadratic equation. But when extrapolated as a general method for computing eigenvalues and implemented on a digital computer, this is a very poor procedure for reasons such as roundoff and overflow/underflow. The preferred method now would generally be the double Francis QR algorithm (see [17] for details) but few would attempt that manually, even for very small order problems.

Many algorithms, now considered fairly reliable in the context of finite arithmetic, are not amenable to hand calculations (e.g., various classes of orthogonal similarities). This is a kind of converse to the observation quoted above. Especially in linear system and control theory, we have been too easily seduced by the ready availability of closed-form solutions and numerically naive methods to implement those solutions. For example, in solving the initial value problem

$$\dot{x}(t) = Ax(t); \qquad x(0) = x_0 \tag{6}$$

it is not at all clear that one should explicitly compute the intermediate quantity e^{tA}. Rather, it is the vector $e^{tA}x_0$ that is desired, a quantity that may be computed by treating (6) as a system of (possibly stiff) differential equations and using an implicit method for numerically integrating the differential equation. But such techniques are definitely not attractive for hand computation.

Awareness of such numerical issues in the mathematics and engineering community has increased significantly in the last few decades. In fact, some of the background material well known to numerical analysts has already filtered down to undergraduate and graduate curricula in these disciplines. This awareness and education has affected system and control theory, especially linear system theory. A number of numerical analysts were attracted by the wealth of interesting numerical linear algebra problems in linear system theory. At the same time, several researchers in linear system theory turned to various methods and concepts from numerical linear algebra and attempted to modify them in developing reliable algorithms and software for specific problems in linear system theory. This cross-fertilization has been greatly enhanced by the widespread use of software packages and by developments over the last couple of decades in numerical linear algebra. This process has already begun to have a significant impact on the future directions and development of system and control theory, and on applications, as evident from the growth of computer-aided control system design (CACSD) as an intrinsic tool. Algorithms implemented as mathematical software are a critical "inner" component of a CACSD system.

In the remainder of this chapter, we survey some results and trends in this interdisciplinary research area. We emphasize numerical aspects of the problems/algorithms, which is why we also spend time discussing appropriate numerical tools and techniques. We discuss a number of control and filtering problems that are of widespread interest in control.

Before proceeding further we list here some notation to be used:

$\mathbb{F}^{n\times m}$	the set of all $n \times m$ matrices with coefficients in the field $\mathbb{F}$ ($\mathbb{F}$ is generally $\mathbb{R}$ or $\mathbb{C}$)
A^T	the transpose of $A \in \mathbb{R}^{n\times m}$
A^H	the complex-conjugate transpose of $A \in \mathbb{C}^{n\times m}$
A^+	the Moore-Penrose pseudoinverse of A
$\|A\|$	the spectral norm of A (i.e., the matrix norm subordinate to the Euclidean vector norm: $\|A\| = \max_{\|x\|_2=1} \|Ax\|_2$)
diag $(a_1, \cdots, a_n)$	the diagonal matrix $\begin{bmatrix} a_1 & & 0 \\ & \ddots & \\ 0 & & a_n \end{bmatrix}$
$\Lambda(A)$	the set of eigenvalues $\lambda_1, \cdots, \lambda_n$ (not necessarily distinct) of $A \in \mathbb{F}^{n\times n}$
$\lambda_i(A)$	the ith eigenvalue of A
$\Sigma(A)$	the set of singular values $\sigma_1, \cdots, \sigma_m$ (not necessarily distinct) of $A \in \mathbb{F}^{n\times m}$
$\sigma_i(A)$	the ith singular value of A.

Finally, let us define a particular number to which we make frequent reference following. The *machine epsilon* or *relative machine precision* is defined, roughly speaking, as the smallest positive number ϵ that, when added to 1 on our computing machine, gives a number greater than 1. In other words, any machine representable number δ less than ϵ gets " rounded off" when (floating-point) added to 1 to give exactly 1 again as the rounded sum. The number ϵ, of course, varies depending on the kind of computer being used and the precision of the computations (single precision, double precision, etc.). But the fact that such a positive number ϵ exists is entirely a consequence of finite word length.

2 Numerical Background

In this section we give a very brief discussion of two concepts fundamentally important in numerical analysis: *numerical stability* and *conditioning.* Although this material is standard in textbooks such as [8], it is presented here for completeness and because the two concepts are frequently confused in the systems, control, and estimation literature.

Suppose we have some mathematically defined problem represented by f which acts on data d belonging to some set of data $\mathcal{D}$, to produce a solution $f(d)$ in a solution set $\mathcal{S}$. These notions are

kept deliberately vague for expository purposes. Given $d \in \mathcal{D}$ we desire to compute $f(d)$. Suppose d^* is some approximation to d. If $f(d^*)$ is "near" $f(d)$, the problem is said to be well-conditioned. If $f(d^*)$ may potentially differ greatly from $f(d)$ even when d^* is near d, the problem is said to be ill-conditioned. The concept of "near" can be made precise by introducing norms in the appropriate spaces. We can then define the condition of the problem f with respect to these norms as

$$\kappa[f(d)] = \lim_{\delta \to 0} \sup_{d_2(d,d^*)=\delta} \left[\frac{d_1(f(d), f(d^*))}{\delta} \right] \tag{7}$$

where d_i $(.\ ,\ .)$ are distance functions in the appropriate spaces. When $\kappa[f(d)]$ is infinite, the problem of determining $f(d)$ from d is *ill-posed* (as opposed to *well-posed*). When $\kappa[f(d)]$ is finite and *relatively large* (or *relatively small*), the problem is said to be *ill conditioned* (or *well conditioned*).

A simple example of an ill-conditioned problem is the following. Consider the $n \times n$ matrix

$$A = \begin{bmatrix} 0 & 1 & 0 & \cdot & \cdot & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & & & \cdot & \cdot & \cdot & 0 \\ \cdot & & & & \cdot & \cdot & 1 \\ 0 & \cdot & \cdot & \cdot & \cdot & \cdot & 0 \end{bmatrix}$$

with n eigenvalues at 0. Now consider a small perturbation of the data (the n^2 elements of A) consisting of adding the number 2^{-n} to the first element in the last (nth) row of A. This perturbed matrix then has n distinct eigenvalues $\lambda_1, \cdots, \lambda_n$ with $\lambda_k = 1/2 \exp(2k\pi j/n)$ where $j := \sqrt{-1}$. Thus, we see that this small perturbation in the data has been magnified by a factor on the order of 2^n resulting in a rather large perturbation in solving the problem of computing the eigenvalues of A. Further details and related examples can be found in [17],[9].

Thus far we have not mentioned how the problem f above (computing the eigenvalues of A in the example) was to be solved. Conditioning is a function solely of the problem itself. To solve a problem numerically, we must implement some numerical procedure or algorithm which we denote by f^*. Thus, given d, $f^*(d)$ is the result of applying the algorithm to d (for simplicity, we assume d is "representable"; a more general definition can be given when some approximation d^{**} to d must be used). The algorithm f^* is said to be numerically (backward) stable if, for all $d \in \mathcal{D}$, there exists $d^* \in \mathcal{D}$ near d so that $f^*(d)$ is near $f(d^*)$ ($f(d^*)$ = the exact solution of a nearby problem). If the problem is well conditioned, then $f(d^*)$ is near $f(d)$ so that $f^*(d)$ is near $f(d)$ if f^* is numerically stable. In other words, f^* does not introduce any more sensitivity to perturbation than is inherent in the problem. Example 1 below further illuminates this definition of stability which, on a first reading, can seem somewhat confusing.

Of course, one cannot expect a stable algorithm to solve an ill-conditioned problem any more accurately than the data warrant but an unstable algorithm can produce poor solutions even to well-conditioned problems. Example 2, below, illustrates this phenomenon. There are thus two separate factors to consider in determining the accuracy of a computed solution $f^*(d)$. First, if the algorithm is stable, $f^*(d)$ is near $f(d^*)$, for some d^*, and second, if the problem is well-conditioned then, as above, $f(d^*)$ is near $f(d)$. Thus, $f^*(d)$ is near $f(d)$ and we have an "accurate" solution.

Rounding errors can cause unstable algorithms to give disastrous results. However, it would be virtually impossible to account for every rounding error made at every arithmetic operation in a complex series of calculations. This would constitute a *forward* error analysis. The concept of *backward* error analysis based on the definition of numerical stability given above provides a more practical alternative. To illustrate this, let us consider the singular value decomposition of

an arbitrary $m \times n$ matrix A with coefficients in $\mathbb{R}$ or $\mathbb{C}$ [8] (see also part C of Section 3 of this chapter),

$$A = U\Sigma V^H \ . \tag{8}$$

Here U and V are $m \times m$ and $n \times n$ unitary matrices, respectively, and Σ is an $m \times n$ matrix of the form

$$\Sigma = \left[\begin{array}{c|c} \Sigma_r & 0 \\ \hline 0 & 0 \end{array}\right] ; \Sigma_r = \mathrm{diag}\{\sigma_1, \cdots, \sigma_r\} \tag{9}$$

with the *singular value* σ_i positive and satisfying $\sigma_1 \geq \sigma_2 \cdots \geq \sigma_r > 0$. The computation of this decomposition is, of course, subject to rounding errors. Denoting computed quantities by an overbar, for some *error matrix* E_A,

$$\overline{A} = A + E_A = \overline{U\Sigma V}^H. \tag{10}$$

The computed decomposition thus corresponds exactly to a *perturbed* matrix $\overline{A}$. When using the SVD algorithm available in the literature [8], this perturbation can be bounded by

$$\| E_A \| \leq \pi\epsilon \| A \| \tag{11}$$

where ϵ is the machine precision and π is some quantity depending on the dimensions m and n, but reasonably close to 1 (see also [14](p. 74)). Thus, the *backward error* E_A induced by this algorithm has roughly the same norm as the *input error* E_i resulting, for example, when reading the data A into the computer. Then, according to the definition of numerical stability given above, when a bound such as that in Equation 11 exists for the error induced by a numerical algorithm, the algorithm is said to be *backward stable* [17]. Notice that backward stability does not guarantee any bounds on the errors in the result $\overline{U}, \overline{\Sigma}$, and $\overline{V}$. In fact, this depends on how perturbations in the data (namely $E_A = \overline{A} - A$) affect the resulting decomposition (namely $E_U = \overline{U} - U, E_\Sigma = \overline{\Sigma} - \Sigma$, and $E_V = \overline{V} - V$). This is commonly measured by the condition $\kappa[f(A)]$.

Backward stability is a property of an algorithm and condition is associated with a problem and the specific data for that problem. The errors in the result depend on the stability of the algorithm used and the condition of the problem solved. A *good* algorithm should, therefore, be backward stable because the size of the errors in the result is then mainly due to the condition of the problem, not to the algorithm. An unstable algorithm, on the other hand, may yield a large error even when the problem is well-conditioned.

Bounds of the type Equation 11 are obtained by an error analysis of the algorithm used, and the condition of the problem is obtained by a sensitivity analysis, e.g., see [17],[9].

We close this section with two simple examples to illustrate some of the concepts introduced.

EXAMPLE 21.1

Let x and y be two floating-point computer numbers and let $fl(x*y)$ denote the result of multiplying them in floating-point computer arithmetic. In general, the product $x * y$ requires more precision to be represented exactly than was used to represent x or y. But for most computers

$$fl(x * y) = x * y(1 + \delta) \tag{12}$$

where $|\delta| < \epsilon$ (= relative machine precision). In other words, $fl(x * y)$ is $x * y$ correct to within a unit in the last place. Another way to write Equation 12 is as

$$fl(x * y) = x(1 + \delta)^{1/2} * y(1 + \delta)^{1/2} \tag{13}$$

where $|\delta| < \epsilon$. This can be interpreted as follows: the computed result $fl(x * y)$ is the exact product of the two slightly perturbed numbers $x(1+\delta)^{1/2}$ and $y(1+\delta)^{1/2}$. The slightly perturbed data (not unique) may not even be representable as floating-point numbers. The representation Equation 13 is simply a way of accounting for the roundoff incurred in the algorithm by an initial (small) perturbation in the data.

EXAMPLE 21.2

Gaussian elimination with no pivoting for solving the linear system of equations

$$Ax = b \tag{14}$$

is known to be numerically unstable; see, for example, [8] and Section 3. The following data illustrates this phenomenon. Let

$$A = \begin{bmatrix} 0.0001 & 1.000 \\ 1.000 & -1.000 \end{bmatrix}, b = \begin{bmatrix} 1.000 \\ 0.000 \end{bmatrix}.$$

All computations are carried out in four-significant-figure decimal arithmetic. The "true answer" $x = A^{-1}b$ is

$$\begin{bmatrix} 0.9999 \\ 0.9999 \end{bmatrix}.$$

Using row 1 as the "pivot row" (i.e., subtracting 10,000 $\times$ row 1 from row 2) we arrive at the equivalent triangular system

$$\begin{bmatrix} 0.0001 & 1.000 \\ 0 & -1.000 \times 10^4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1.000 \\ -1.000 \times 10^4 \end{bmatrix}.$$

The coefficient multiplying x_2 in the second equation should be $-10,001$, but because of roundoff, becomes $-10,000$. Thus, we compute $x_2 = 1.000$ (a good approximation), but back-substitution in the equation

$$0.0001x_1 = 1.000 - fl(1.000 * 1.000)$$

yields $x_1 = 0.000$. This extremely bad approximation to x_1 is the result of numerical instability. The problem, it can be shown, is quite well-conditioned.

3 Fundamental Problems in Numerical Linear Algebra

In this section we give a brief overview of some of the fundamental problems in numerical linear algebra which serve as building blocks or "tools" for the solution of problems in systems, control, and estimation.

A. Linear Algebraic Equations and Linear Least Squares Problems

Probably the most fundamental problem in numerical computing is the calculation of a vector x which satisfies the linear system

$$Ax = b \tag{15}$$

where $A \in \mathbb{R}^{n\times n}$(or $\mathbb{C}^{n\times n}$) and has rank n. A great deal is now known about solving Equation 15 in finite arithmetic both for the general case and for a large number of special situations, e.g., see [8],[9].

The most commonly used algorithm for solving Equation 15 with general A and small n (say $n \leq 200$) is Gaussian elimination with some sort of pivoting strategy, usually "partial pivoting."

This amounts to factoring some permutation of the rows of A into the product of a unit lower triangular matrix L and an upper triangular matrix U. The algorithm is effectively stable, i.e., it can be proved that the computed solution is near the exact solution of the system

$$(A + E)x = b \tag{16}$$

with $|e_{ij}| \leq \phi(n)\,\gamma\,\beta\,\epsilon$ where $\phi(n)$ is a modest function of n depending on details of the arithmetic used, γ is a "growth factor" (which is a function of the pivoting strategy and is usually—but not always—small), β behaves essentially like $\|A\|$, and ϵ is the machine precision. In other words, except for moderately pathological situations, E is "small"—on the order of $\epsilon\,\|A\|$.

The following question then arises. If, because of rounding errors, we are effectively solving Equation 16 rather than Equation 15, what is the relationship between $(A + E)^{-1}b$ and $A^{-1}b$? To answer this question we need some elementary perturbation theory and this is where the notion of condition number arises. A condition number for Equation 15 is given by

$$\kappa(A) := \|A\|\ \|A^{-1}\|. \tag{17}$$

Simple perturbation results can show that perturbation in A and/or b can be magnified by as much as $\kappa(A)$ in the computed solution. Estimating $\kappa(A)$ (since, of course, A^{-1} is unknown) is thus a crucial aspect of assessing solutions of Equation 15 and the particular estimating procedure used is usually the principal difference between competing linear equation software packages. One of the more sophisticated and reliable condition estimators presently available is implemented in LINPACK [5] and its successor LAPACK [2]. LINPACK and LAPACK also feature many codes for solving Equation 14 in case A has certain special structures (e.g., banded, symmetric, or positive definite).

Another important class of linear algebra problems, and one for which codes are available in LINPACK and LAPACK, is the linear least squares problem

$$\min \|Ax - b\|_2 \tag{18}$$

where $A \in \mathbb{R}^{m\times n}$ and has rank k, with (in the simplest case) $k = n \leq m$, e.g., see [8]. The solution of Equation 18 can be written formally as $x = A^{+}b$. The method of choice is generally based upon the QR factorization of A (for simplicity, let $\text{rank}(A) = n$)

$$A = QR \tag{19}$$

where $R \in \mathbb{R}^{n\times n}$ is upper triangular and $Q \in \mathbb{R}^{m\times n}$ has orthonormal columns, i.e., $Q^TQ = I$. With special care and analysis, the case $k < n$ can also be handled similarly. The factorization is effected through a sequence of Householder transformations H_i applied to A. Each H_i is symmetric and orthogonal and of the form $I - 2uu^T/u^Tu$ where $u \in \mathbb{R}^m$ is specially chosen so that zeros are introduced at appropriate places in A when it is premultiplied by H_i. After n such transformations,

$$H_nH_{n-1}\cdots H_1A = \begin{bmatrix} R \\ 0 \end{bmatrix}$$

from which the factorization Equation 19 follows. Defining c and d by

$$\begin{bmatrix} c \\ d \end{bmatrix} := H_nH_{n-1}\cdots H_1b$$

where $c \in \mathbb{R}^n$, it is easily shown that the least squares solution x of Equation 18 is given by the solution of the linear system of equations

$$Rx = c\ . \tag{20}$$

The above algorithm is numerically stable and, again, a well-developed perturbation theory exists from which condition numbers can be obtained, this time in terms of

$$\kappa(A) := \|A\| \ \|A^+\|.$$

Least squares perturbation theory is fairly straightforward when $\text{rank}(A) = n$, but is considerably more complicated when A is rank-deficient. The reason for this is that, although the inverse is a continuous function of the data (i.e., the inverse is a continuous function in a neighborhood of a nonsingular matrix), the pseudoinverse is discontinuous. For example, consider

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} = A^+$$

and perturbations

$$E_1 = \begin{bmatrix} 0 & 0 \\ \delta & 0 \end{bmatrix}$$

and

$$E_2 = \begin{bmatrix} 0 & 0 \\ 0 & \delta \end{bmatrix}$$

with δ small. Then

$$(A + E_1)^+ = \begin{bmatrix} \frac{1}{1+\delta^2} & \frac{\delta}{1+\delta^2} \\ 0 & 0 \end{bmatrix}$$

which is close to A^+ but

$$(A + E_2)^+ = \begin{bmatrix} 1 & 0 \\ 0 & \frac{1}{\delta} \end{bmatrix}$$

which gets arbitrarily far from A^+ as δ is decreased towards 0.

In lieu of Householder transformations, Givens transformations (elementary rotations or reflections) may also be used to solve the linear least squares problem [8]. Givens transformations have received considerable attention for solving linear least squares problems and systems of linear equations in a parallel computing environment. The capability of introducing zero elements selectively and the need for only local interprocessor communication make the technique ideal for "parallelization."

B. Eigenvalue and Generalized Eigenvalue Problems

In the algebraic eigenvalue/eigenvector problem for $A \in \mathbb{R}^{n\times n}$, one seeks nonzero solutions $x \in \mathbb{C}^n$ and $\lambda \in \mathbb{C}$ which satisfy

$$Ax = \lambda x. \tag{21}$$

The classic reference on the numerical aspects of this problem is Wilkinson [17]. A briefer textbook introduction is given in [8].

Quality mathematical software for eigenvalues and eigenvectors is available; the EISPACK [7], [15] collection of subroutines represents a pivotal point in the history of mathematical software. The successor to EISPACK (and LINPACK) is LAPACK [2] in which the algorithms and software have been restructured to provide high efficiency on vector processors, high performance workstations, and shared memory multiprocessors.

The most common algorithm now used to solve Equation 21 for general A is the QR algorithm of Francis [17]. A shifting procedure enhances convergence and the usual implementation is called the double-Francis-QR algorithm. Before the QR process is applied, A is initially reduced to upper Hessenberg form $A_H (a_{ij} = 0 \text{ if } i - j \geq 2)$. This is accomplished by a finite sequence of similarities

of the Householder form discussed above. The QR process then yields a sequence of matrices orthogonally similar to A and converging (in some sense) to a so-called quasi upper triangular matrix S also called the real Schur form (RSF) of A. The matrix S is block upper triangular with 1×1 diagonal blocks corresponding to real eigenvalues of A and 2×2 diagonal blocks corresponding to complex-conjugate pairs of eigenvalues. The quasi upper triangular form permits all arithmetic to be real rather than complex as would be necessary for convergence to an upper triangular matrix. The orthogonal transformations from both the Hessenberg reduction and the QR process may be accumulated in a single orthogonal transformation U so that

$$U^T AU = R \tag{22}$$

compactly represents the entire algorithm. An analogous process can be applied in the case of symmetric A, and considerable simplifications and specializations result.

Closely related to the QR algorithm is the QZ algorithm for the generalized eigenvalue problem

$$Ax = \lambda Mx \tag{23}$$

where $A, M \in \mathbb{R}^{n\times n}$. Again, a Hessenberg-like reduction, followed by an iterative process, is implemented with orthogonal transformations to reduce Equation 23 to the form

$$QAZy = \lambda QMZy \tag{24}$$

where QAZ is quasi upper triangular and QMZ is upper triangular. For a review and references to results on stability, conditioning, and software related to Equation 23 and the QZ algorithm, see [8]. The generalized eigenvalue problem is both theoretically and numerically more difficult to handle than the ordinary eigenvalue problem, but it finds numerous applications in control and system theory [14](p. 109).

C. The Singular Value Decomposition and Some Applications

One of the basic and most important tools of modern numerical analysis, especially numerical linear algebra, is the singular value decomposition (SVD). Here we make a few comments about its properties and computation as well as its significance in various numerical problems.

Singular values and the singular value decomposition have a long history, especially in statistics and numerical linear algebra. These ideas have found applications in the control and signal processing literature, although their use there has been overstated somewhat in certain applications. For a survey of the singular value decomposition, its history, numerical details, and some applications in systems and control theory, see [14](p. 74).

The fundamental result was stated in Section 2 (for the complex case). The result for the real case is similar and is stated below.

Theorem 3.1 *Let $A \in \mathbb{R}^{m\times n}$ with rank$(A) = r$. Then there exist orthogonal matrices $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$ so that*

$$A = U\Sigma V^T \tag{25}$$

where

$$\Sigma = \begin{bmatrix} \Sigma_r & 0 \\ 0 & 0 \end{bmatrix}$$

and $\Sigma_r = diag\,\{\sigma_1, \cdots, \sigma_r\}$ with $\sigma_1 \geq \cdots \geq \sigma_r > 0$.

The proof of Theorem 3.1 is straightforward and can be found, for example, in [8]. Geometrically, the theorem says that bases can be found (separately) in the domain and codomain spaces of a linear map with respect to which the matrix representation of the linear map is diagonal. The numbers $\sigma_1, \cdots, \sigma_r$ together with $\sigma_{r+1} = 0, \cdots, \sigma_n = 0$ are called the singular values of A, and they are the positive square roots of the eigenvalues of $A^T A$. The columns $\{u_k, k = 1. \cdots, m\}$ of U are called the left singular vectors of A (the orthonormal eigenvectors of AA^T), while the columns $\{v_k, k = 1, \cdots, n\}$ of V are called the right singular vectors of A (the orthonormal eigenvectors of $A^T A$). The matrix A can then also be written (as a dyadic expansion) in terms of the singular vectors as follows:

$$A = \sum_{k=1}^{r} \sigma_k u_k v_k^T.$$

The matrix A^T has m singular values, the positive square roots of the eigenvalues of AA^T. The r [= rank (A)] nonzero singular values of A and A^T are, of course, the same. The choice of $A^T A$ rather than AA^T in the definition of singular values is arbitrary. Only the nonzero singular values are usually of any real interest and their number, given the SVD, is the rank of the matrix. Naturally, the question of how to distinguish nonzero from zero singular values in the presence of rounding error is a nontrivial task.

It is not generally advisable to compute the singular values of A by first finding the eigenvalues of $A^T A$, tempting as that is. Consider the following example, where μ is a real number with $|\mu| < \sqrt{\epsilon}$ (so that $fl(1 + \mu^2) = 1$ where $fl(\cdot)$ denotes floating-point computation). Let

$$A = \begin{bmatrix} 1 & 1 \\ \mu & 0 \\ 0 & \mu \end{bmatrix}.$$

Then

$$fl(A^T A) = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}.$$

So we compute $\hat{\sigma}_1 = \sqrt{2},\ \hat{\sigma}_2 = 0$ leading to the (erroneous) conclusion that the rank of A is 1. Of course, if we could compute in infinite precision, we would find

$$A^T A = \begin{bmatrix} 1 + \mu^2 & 1 \\ 1 & 1 + \mu^2 \end{bmatrix}$$

with $\sigma_1 = \sqrt{2 + \mu^2}, \sigma_2 = |\mu|$ and thus rank $(A) = 2$. The point is that by working with $A^T A$ we have unnecessarily introduced μ^2 into the computations. The above example illustrates a potential pitfall in attempting to form and solve the normal equations in a linear least squares problem and is at the heart of what makes square root filtering so attractive numerically. Very simplistically, square root filtering involves working directly on an "A-matrix," for example, updating it, as opposed to updating an "$A^T A$-matrix."

Square root filtering is usually implemented with the QR factorization (or some closely related algorithm) as described previously rather than SVD. Moreover, critical information may be lost irrecoverably by simply forming $A^T A$.

Returning now to the SVD, two features of this matrix factorization make it attractive in finite arithmetic: first, it can be computed in a numerically stable way, and second, singular values are well-conditioned. Specifically, there is an efficient and numerically stable algorithm due to Golub and Reinsch [8] which works directly on A to give the SVD. This algorithm has two phases. In the first phase, it computes orthogonal matrices U_1 and V_1 so that $B = {U_1}^T A V_1$ is in bidiagonal

form, i.e., only the elements on its diagonal and first superdiagonal are nonzero. In the second phase, the algorithm uses an iterative procedure to compute orthogonal matrices U_2 and V_2 so that ${U_2}^T B V_2$ is diagonal and nonnegative. The SVD defined in Equation 25 is then $\Sigma = U^T B V$, where $U = U_1 U_2$ and $V = V_1 V_2$. The computed U and V are orthogonal approximately to the working precision, and the computed singular values are the exact σ_i's for $A+E$ where $\|E\|/\|A\|$ is a modest multiple of ϵ. Fairly sophisticated implementations of this algorithm can be found in [5] and [7]. The well-conditioned nature of the singular values follows from the fact that if A is perturbed to $A + E$, then it can be proved that

$$|\sigma_i(A + E) - \sigma_i(A)| \leq \|E\|.$$

Thus, the singular values are computed with small absolute error although the relative error of sufficiently small singular values is not guaranteed to be small.

It is now acknowledged that the singular value decomposition is the most generally reliable method of determining rank numerically (see [14](p. 589) for a more elaborate discussion). However, it is considerably more expensive to compute than, for example, the QR factorization which, with column pivoting [5], can usually give equivalent information with less computation. Thus, while the SVD is a useful theoretical tool, its use for actual computations should be weighed carefully against other approaches.

The problem of numerical determination of rank is now well-understood. The essential idea is to try to determine a "gap" between "zero" and the "smallest nonzero singular value" of a matrix A. Since the computed values are exact for a matrix near A, it makes sense to consider the ranks of all matrices in some δ-ball (with respect to the spectral norm $\|\cdot\|$, say) around A. The choice of δ may also be based on measurement errors incurred in estimating the coefficients of A, or the coefficients may be uncertain because of rounding errors incurred in a previous computation. However, even with SVD, numerical determination of rank in finite arithmetic is a difficult problem.

That other methods of rank determination are potentially unreliable is demonstrated by the following example. Consider the Ostrowski matrix $A \in \mathbb{R}^{n\times n}$ whose diagonal elements are all -1, whose upper triangle elements are all $+1$, and whose lower triangle elements are all 0. This matrix is clearly of rank n, i.e., is invertible. It has a good "solid" upper triangular shape. All of its eigenvalues $(= -1)$ are well away from zero. Its determinant $(-1)^n$ is definitely not close to zero. But this matrix is, in fact, very nearly singular and becomes more nearly so as n increases. Note, for example, that

$$\begin{bmatrix} -1 & +1 & \cdots & & \cdots & +1 \\ 0 & \ddots & \ddots & & & \vdots \\ \vdots & \ddots & \ddots & \ddots & & \\ & & & \ddots & \ddots & \vdots \\ \vdots & & & \ddots & \ddots & +1 \\ 0 & \cdots & & \cdots & 0 & -1 \end{bmatrix} \begin{bmatrix} 1 & & & \\ & 2^{-1} & & \\ & & \vdots & \\ & & & 2^{-n+1} \end{bmatrix}$$

$$= \begin{bmatrix} -2^{-n+1} \\ -2^{-n+1} \\ \vdots \\ -2^{-n+1} \end{bmatrix} \rightarrow \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} (n \rightarrow +\infty).$$

Moreover, adding 2^{-n+1} to every element in the first column of A gives an exactly singular matrix. Arriving at such a matrix by, say Gaussian elimination, would give no hint as to the

near singularity. However, it is easy to check that $\sigma_n(A)$ behaves as 2^{-n+1}. A corollary for control theory is that eigenvalues do not necessarily give a reliable measure of "stability margin." It is useful to note that in this example of an invertible matrix, the crucial quantity, $\sigma_n(A)$, which measures nearness to singularity, is simply $1/\|A^{-1}\|$, and the result is familiar from standard operator theory. There is nothing intrinsic about singular values in this example and, in fact, $\|A^{-1}\|$ might be more cheaply computed or estimated in other matrix norms.

Because rank determination, in the presence of rounding error, is a nontrivial problem, the same difficulties naturally arise in any problem equivalent to, or involving, rank determination, such as determining the independence of vectors, finding the dimensions of certain subspaces, etc. Such problems arise as basic calculations throughout systems, control, and estimation theory. Selected applications are discussed in more detail in [14](p. 74) and in [1],[4],[10].

Finally, let us close this section with a brief example illustrating a totally inappropriate use of SVD. The rank condition

$$\text{rank } [B, AB, \cdots, A^{n-1}B] = n \tag{26}$$

for the controllability of Equation 1 is too well-known. Suppose

$$A = \begin{bmatrix} 1 & \mu \\ 0 & 1 \end{bmatrix}, B = \begin{bmatrix} 1 \\ \mu \end{bmatrix}$$

with $|\mu| < \sqrt{\epsilon}$. Then

$$fl[B, AB] = \begin{bmatrix} 1 & 1 \\ \mu & \mu \end{bmatrix}$$

and now even applying SVD, the erroneous conclusion of uncontrollability is reached. Again the problem is in just forming AB; not even SVD can come to the rescue after that numerical *faux pas*.

4 Applications to Systems and Control

A reasonable approach to developing numerically reliable algorithms for computational problems in linear system theory would be to reformulate the problems as concatenations of subproblems for which numerically stable algorithms are available. Unfortunately, one cannot ensure that the stability of algorithms for the subproblems results in stability of the overall algorithm. This requires separate analysis which may rely on the sensitivity or condition of the subproblems. In the next section we show that delicate (i.e., badly-conditioned) subproblems should be avoided whenever possible; a few examples are given where a possibly badly-conditioned step is circumvented by carefully modifying or completing existing algorithms; see, e.g., [14](p. 109).

A second difficulty is the ill-posedness of some of the problems occurring in linear system theory. Two approaches can be adopted. One can develop an acceptable perturbation theory for such problems, using a concept such as *restricted condition* which is condition under perturbations for which a certain property holds, e.g., fixed rank [14](p. 109). One then looks for restricting assumptions that make the problem well-posed. Another approach is to delay any such *restricting choices* to the end and leave it to the user to decide which choice to make by looking at the results. The algorithm then provides quantitative measures that help the user make this choice; see, e.g., [14](p. 171, p. 529). By this approach one may avoid artificial restrictions of the first approach that sometimes do not respect the practical significance of the problem.

A third possible *pitfall* is that many users almost always prefer fast algorithms to slower ones. However, slower algorithms are often more reliable.

In the subsections that follow, we survey a representative selection of numerical linear algebra problems arising in linear systems, control, and estimation theory, which have been examined with some of the techniques described in the preceding sections. Many of these topics are discussed briefly in survey papers such as [11] and [16] and in considerably more detail in the papers included or referenced in [14] and in [4],[1],[10]. Some of the scalar algorithms discussed here do not extend trivially to the matrix case. When they do, we mention only the matrix case. Moreover, we discuss only the numerical aspects here; for the system-theoretical background, we refer the reader to the control and systems literature.

4.1 Some Typical Techniques

Most of the reliable techniques in numerical linear algebra are based on the use of orthogonal transformations. Typical examples of this are the QR decomposition for least squares problems, the Schur decomposition for eigenvalue and generalized eigenvalue problems, and the singular value decomposition for rank determinations and generalized inverses. Orthogonal transformations also appear in most of the reliable linear algebra techniques for control theory. This is partly due to the direct application of existing linear algebra decompositions to problems in control. Obvious examples of this are the Schur approach for solving algebraic Riccati equations, both continuous- and discrete-time [14](p. 529, p. 562, p. 573), for solving Lyapunov equations [14](p. 430), and for performing pole placement [14](p. 415). New orthogonal decompositions have also been introduced that rely heavily on the same principles but were specifically developed for problems encountered in control. Orthogonal state-space transformations on a system $\{A, B, C\}$ result in a new state-space representation $\{U^H AU, U^H B, CU\}$ where U performs some kind of decomposition on the matrices A, B, and C. These special forms, termed "condensed forms," include

- the state Schur form [14](p. 415),
- the state Hessenberg form [14](p. 287),
- the observer Hessenberg form [14](p. 289, p. 392), and
- the controller Hessenberg form [14](p. 128, p. 357).

Staircase forms or block Hessenberg forms are other variants of these condensed forms that have proven useful in dealing with MIMO systems [14](p. 109, p. 186, p. 195).

There are two main reasons for using these orthogonal state-space transformations:

- The numerical sensitivity of the control problem being solved is not affected by these transformations because sensitivity is measured by norms or angles of certain spaces and these are unaltered by orthogonal transformations.
- Orthogonal transformations have minimum condition number, essential in proving bounded error propagation and establishing numerical stability of the algorithm that uses such transformations.

More details on this are given in the paper [14](p. 128) and in subsequent sections where some of these condensed forms are used for particular applications.

4.2 Transfer Functions, Poles, and Zeros

In this section, we discuss important structural properties of linear systems and the numerical techniques available for determining them. The transfer function $R(\lambda)$ of a linear system is given by a polynomial representation $V(\lambda)T^{-1}(\lambda)U(\lambda)+W(\lambda)$ or by a state-space model $C(\lambda I - A)^{-1}B+D$. The results in this subsection hold for both the discrete-time case (where λ stands for the shift operator z) and the continuous-time case (where λ stands for the differentiation operator D).

A. The Polynomial Approach

One is interested in a number of structural properties of the transfer function $R(\lambda)$ such as poles, transmission zeros, decoupling zeros, etc. In the scalar case, where $\{T(\lambda), U(\lambda), V(\lambda), W(\lambda)\}$ are scalar polynomials, all of this can be found with a greatest common divisor (GCD) extraction routine and a rootfinder, for which reliable methods exist. In the matrix case, the problem becomes much more complex and the basic method for GCD extraction, the Euclidean algorithm, becomes unstable (see [14](p. 109)). Moreover, other structural elements (null spaces, etc.) come into the picture, making the polynomial approach less attractive than the state-space approach [14](p. 109).

B. The State-Space Approach (see [14](p. 109), and references therein)

The structural properties of interest are poles and zeros of $R(\lambda)$, decoupling zeros, controllable and unobservable subspaces, supremal (A, B)-invariant and controllability subspaces, factorizability of $R(\lambda)$, left and right null spaces of $R(\lambda)$, etc. These concepts are fundamental in several design problems and have received considerable attention over the last few decades; see, e.g., [14](pp. 74, 109, 174, 186, 529). In [14](p. 109), it is shown that all the concepts mentioned above can be considered generalized eigenstructure problems and that they can be computed via the Kronecker canonical form of the pencils

$$\begin{array}{cc} \left[\begin{array}{c} \lambda I - A \end{array}\right] & \left[\begin{array}{c|c} \lambda I - A & B \end{array}\right] \\ \left[\begin{array}{c} \lambda I - A \\ \hline -C \end{array}\right] & \left[\begin{array}{c|c} \lambda I - A & B \\ \hline -C & D \end{array}\right] \end{array} \tag{27}$$

or from other pencils derived from these. Backward stable software is also available for computing the Kronecker structure of an arbitrary pencil. A remaining problem here is that determining several of the structural properties listed above may be ill-posed in some cases in which one has to develop the notion of restricted condition (see [14](p. 109)). A completely different approach is to reformulate the problem as an approximation or optimization problem for which *quantitative measures* are derived, leaving the final choice to the user. Results in this vein are obtained for controllability, observability [14](pp. 171, 186, 195), (almost) (A, B)-invariant, and controllability subspaces.

4.3 Controllability and Other "Abilities"

The various "abilities," such as controllability, observability, reachability, reconstructibility, stabilizability, and detectability are basic to the study of linear control and system theory. These concepts can also be viewed in terms of decoupling zeros, controllable and unobservable subspaces, controllability subspaces, etc. mentioned in the previous section. Our remarks here are confined, but not limited, to the notion of controllability.

A large number of algebraic and dynamic characterizations of controllability have been given; see [11] for a sample. But every one of these has difficulties when implemented in finite arithmetic. For a survey of this topic and numerous examples, see [14](p. 186). Part of the difficulty in dealing with

controllability numerically lies in the intimate relationship with the invariant subspace problem [14](p. 589). The controllable subspace associated with Equation 1 is the smallest A-invariant subspace (subspace spanned by eigenvectors or principal vectors) containing the range of B. Since A-invariant subspaces can be extremely sensitive to perturbation, it follows that, so too, is the controllable subspace. Similar remarks apply to the computation of the so-called controllability indices. The example discussed in the third paragraph of Section 2 dramatically illustrates these remarks. The matrix A has but one eigenvector (associated with 0) whereas the slightly perturbed A has n eigenvectors associated with the n distinct eigenvalues.

Attempts have been made to provide numerically stable algorithms for the pole placement problem discussed in a later section. It suffices to mention here that the problem of pole placement by state feedback is closely related to controllability. Work on developing numerically stable algorithms for pole placement is based on the reduction of A to a Hessenberg form; see, e.g., [14](pp. 357, 371, 380). In the single-input case, a good approach is the controller Hessenberg form mentioned above where the state matrix A is upper Hessenberg and the input vector B is a multiple of $(1, 0, \cdots, 0)^T$. The pair (A, B) is then controllable if, and only if, all $(n-1)$ subdiagonal elements of A are nonzero. If a subdiagonal element is 0, the system is uncontrollable, and a basis for the uncontrollable subspace is easily constructed. The transfer function gain or first nonzero Markov parameter is also easily constructed from this "canonical form." In fact, the numerically more robust system Hessenberg form, playing an ever-increasing role in system theory is replacing the numerically more fragile special case of the companion or rational canonical or Luenberger canonical form.

A more important aspect of controllability is topological notions such as "near uncontrollability." But there are numerical difficulties here also, and we refer to Parts 3 and 4 of [14] for further details. Related to this is an interesting system-theoretic concept called "balancing" discussed in Moore's paper [14] (p. 171). The computation of "balancing transformations" is discussed in [14](p. 642).

There are at least two distinct notions of near-uncontrollability [11] in the parametric sense and in the energy sense. In the parametric sense, a controllable pair (A, B) is said to be near-uncontrollable if the parameters of (A, B) need be perturbed by only a relatively small amount for (A, B) to become uncontrollable. In the energy sense, a controllable pair is near uncontrollable if large amounts of control energy ($\int u^T u$) are required for a state transfer. The pair

$$A = \begin{pmatrix} 0 & 1 & & \cdots & 0 \\ \vdots & \ddots & \ddots & & \vdots \\ & & & & \\ \vdots & & & \ddots & 1 \\ 0 & \cdots & & \cdots & 0 \end{pmatrix}, \; B = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix}$$

is very near uncontrollable in the energy sense but not as badly so in the parametric sense. Of course, both measures are coordinate dependent and "balancing" is one attempt to remove this coordinate bias. The pair (A, B) above is in "controllable canonical form." It is now known that matrices in this form (specifically, the A matrix in rational canonical form) almost always exhibit poor numerical behavior and are "close to" uncontrollable (unstable, etc.) as the size n increases. For details, see [14](p. 59).

4.4 Computation of Objects Arising in the Geometric Theory of Linear Multivariable Control

A great many numerical problems arise in the geometric approach to control of systems modeled as Equation 1 and 2. Some of these are discussed in the paper by Klema and Laub [14](p. 74). The power of the geometric approach derives in large part from the fact that it is independent of specific coordinate systems or matrix representations. Numerical issues are a separate concern.

A very thorough numerical treatment of numerical problems in linear system theory has been given by Van Dooren [14](p. 109). This work has applications for most calculations with linear state-space models. For example, one byproduct is an extremely reliable algorithm (similar to an orthogonal version of Silverman's structure algorithm) for the computation of multivariable system zeros [14](p. 271). This method involves a generalized eigenvalue problem (the Rosenbrock pencil), but the "infinite zeros" are first removed by deflating the given matrix pencil.

4.5 Frequency Response Calculations

Many properties of a linear system such as Equations 1 and 2 are known in terms of its frequency response matrix

$$G(j\omega) := C(j\omega I - A)^{-1}B + D; \quad (\omega \geq 0) \tag{28}$$

(or $G(e^{j\theta})$; $\theta \in [0, 2\pi]$ for Equations 3 and 4). In fact, various norms of the return difference matrix $I + G(j\omega)$ and related quantities have been investigated in control and system theory to providing robust linear systems with respect to stability, noise response, disturbance attenuation, sensitivity, etc.

Thus it is important to compute $G(j\omega)$ efficiently, given A, B, and C for a (possibly) large number of values of ω (for convenience we take D to be 0 because if it is nonzero it is trivial to add to G). An efficient and generally applicable algorithm for this problem is presented in [14](p. 287). Rather than solving the linear equation $(j\omega I - A)X = B$ with dense unstructured A, which would require $O(n^3)$ operations for each successive value of ω, the new method initially reduces A to upper Hessenberg form H. The orthogonal state-space coordinate transformations used to obtain the Hessenberg form of A are incorporated into B and C giving $\tilde{B}$ and $\tilde{C}$. As ω varies, the coefficient matrix in the linear equation $(j\omega I - H)X = \tilde{B}$ remains in upper Hessenberg form. The advantage is that X can now be found in $O(n^2)$ operations rather than $O(n^3)$ as before, a substantial saving. Moreover, the method is numerically very stable (via either LU or QR factorization) and has the advantage of being independent of the eigenstructure (possibly ill-conditioned) of A. Another efficient and reliable algorithm for frequency response computation [14](p. 289) uses the observer Hessenberg form mentioned in Section 4.1 together with a determinant identity and a property of the LU decomposition of a Hessenberg matrix.

The methods above can also be extended to state-space models in implicit form, i.e., where Equation 1 is replaced by

$$E\dot{x} = Ax + Bu. \tag{29}$$

Then Equation 28 is replaced with

$$G(j\omega) = C(j\omega E - A)^{-1}B + D, \tag{30}$$

and the initial triangular/Hessenberg reduction [8] can be employed again to reduce the problem to updating the diagonal of a Hessenberg matrix and consequently an $O(n^2)$ problem.

An improvement for the frequency response evaluation problem is using matrix interpolation methods to achieve even greater computational efficiency.

4.6 Numerical Solution of Linear Ordinary Differential Equations and Matrix Exponentials

The "simulation" or numerical solution of linear systems of ordinary differential equations (ODEs) of the form,

$$\dot{x}(t) = Ax(t) + f(t), \qquad x(0) = x_0, \tag{31}$$

is a standard problem that arises in finding the time response of a system in state-space form. However, there is still debate as to the most effective numerical algorithm, particularly when A is defective (i.e., when A is $n \times n$ and has fewer than n linearly independent eigenvectors) or nearly defective. The most common approach involves computing the matrix exponential e^{tA}, because the solution of Equation 31 can be written simply as

$$x(t) = e^{tA}x_0 + \int_0^t e^{(t-s)A} f(s)\, ds.$$

A delightful survey of computational techniques for matrix exponentials is given in [14](p. 649). Nineteen "dubious" ways are explored (there are many more ways not discussed) but no clearly superior algorithm is singled out. Methods based on Padé approximation or reduction of A to real Schur form are generally attractive while methods based on Taylor series or the characteristic polynomial of A are generally found unattractive. An interesting open problem is the design of a special algorithm for the matrix exponential when the matrix is known a priori to be stable ($\Lambda(A)$ in the left half of the complex plane).

The reason for the adjective "dubious" in the title of [14](p. 649) is that in many (maybe even most) circumstances, it is better to treat Equation 31 as a system of differential equations, typically stiff, and to apply various ODE techniques, specially tailored to the linear case. ODE techniques are preferred when A is large and sparse for, in general, e^{tA} is unmanageably large and dense.

4.7 Lyapunov, Sylvester, and Riccati Equations

Certain matrix equations arise naturally in linear control and system theory. Among those frequently encountered in the analysis and design of continuous-time systems are the Lyapunov equation

$$AX + XA^T + Q = 0, \tag{32}$$

and the Sylvester equation

$$AX + XF + Q = 0. \tag{33}$$

The appropriate discrete-time analogs are

$$AXA^T - X + Q = 0 \tag{34}$$

and

$$AXF - X + Q = 0. \tag{35}$$

Various hypotheses are posed for the coefficient matrices A, F, Q to ensure certain properties of the solution X.

The literature in control and system theory on these equations is voluminous, but most of it is *ad hoc*, at best, from a numerical point of view, with little attention to questions of numerical stability, conditioning, machine implementation, and the like.

For the Lyapunov equation the best overall algorithm in terms of efficiency, accuracy, reliability, availability, and ease of use is that of Bartels and Stewart [14](p. 430). The basic idea is to reduce A to quasi-upper-triangular form [or real Schur form (RSF)] and to perform a back substitution for the elements of X.

An attractive algorithm for solving Lyapunov equations has been proposed by Hammarling [14](p. 500). This algorithm is a variant of the Bartels-Stewart algorithm but instead solves directly for the Cholesky factor Y of X: $Y^TY = X$ and Y is upper triangular. Clearly, given Y, X is easily recovered if necessary. But in many applications, for example [14](p. 642), only the Cholesky factor is required.

For the Lyapunov equation, when A is stable, the solutions of the equations above are also equal to the reachability and observability Grammians $P_r(T)$ and $P_o(T)$, respectively, for $T = +\infty$ for the system $\{A, B, C\}$:

$$\begin{array}{rcl} P_r(T) & = & \int_0^T e^{tA} B B^T e^{tA^T} dt; \\ P_o(T) & = & \int_0^T e^{tA^T} C^T C e^{tA} dt \\ P_r(T) & = & \sum_{k=0}^T A^k B B^T (A^T)^k; \\ P_o(T) & = & \sum_{k=0}^T (A^T)^k C^T C A^k. \end{array} \tag{36}$$

These can be used along with some additional transformations (see [14](pp. 171, 642)) to compute so-called *balanced* realizations $\{\tilde{A}, \tilde{B}, \tilde{C}\}$. For these realizations both P_o and P_r are equal and diagonal. These realizations have some nice sensitivity properties with respect to poles, zeros, truncation errors in digital filter implementations, etc. [14](p. 171). They are, therefore, recommended whenever the choice of a realization is left to the user. When A is not stable, one can still use the *finite range* Grammians Equation 36, for $T < +\infty$, for balancing [14](p. 171). A reliable method for computing integrals and sums of the type Equation 36 can be found in [14](p. 681). It is also shown in [14](p. 171) that the reachable subspace and the unobservable subspace are the image and the kernel of $P_r(T)$ and $P_o(T)$, respectively. From these relationships, sensitivity properties of the spaces under perturbations of $P_r(T)$ and $P_o(T)$ can be derived.

For the Sylvester equation, the Bartels-Stewart algorithm reduces both A and F to real Schur form (RSF) and then a back substitution is done. It has been demonstrated in [14](p. 495) that some improvement in this procedure is possible by only reducing the larger of A and F to upper Hessenberg form. The stability of this method has been analyzed in [14](p. 495). Although only *weak stability* is obtained, this is satisfactory in most cases.

Algorithms are also available in the numerical linear algebra literature for the more general Sylvester equation

$$A_1 X {F_1}^T + A_2 X {F_2}^T + Q = 0$$

and its symmetric Lyapunov counterpart

$$AXF^T + FXA^T + Q = 0.$$

Questions remain about estimating the condition of Lyapunov and Sylvester equations efficiently and reliably in terms of the coefficient matrices. A deeper analysis of the Lyapunov and Sylvester equations is probably a prerequisite to at least a better understanding of condition of the Riccati equation for which, again, there is considerable theoretical literature but not as much known from a purely numerical point of view. The symmetric $n \times n$ algebraic Riccati equation takes the form

$$Q + AX + XA^T - XGX = 0 \tag{37}$$

for continuous-time systems and

$$A^T XA - X - A^T XG_1(G_2 + G_1^T XG_1)^{-1}G_1^T XA + Q = 0 \tag{38}$$

for discrete-time systems. These equations appear in several design/analysis problems, such as optimal control, optimal filtering, spectral factorization, e.g., see the papers in Part 7 of [14] and references therein. Again, appropriate assumptions are made on the coefficient matrices to guarantee the existence and/or uniqueness of certain kinds of solutions X. Nonsymmetric Riccati equations of the form

$$Q + A_1X + XA_2 - XGX = 0 \tag{39}$$

for the continuous-time case (along with an analog for the discrete-time case) are also studied and can be solved numerically by the techniques discussed below.

Several algorithms have been proposed based on different approaches. One of the more reliable general-purpose methods for solving Riccati equations is the Schur method [14](p. 529). For the case of Equation 37, for example, this method is based upon reducing the associated $2n \times 2n$ Hamiltonian matrix

$$\begin{pmatrix} A & -G \\ -Q & -A^T \end{pmatrix} \tag{40}$$

to RSF. If the RSF is ordered so that its stable eigenvalues (there are exactly n of them under certain standard assumptions) are in the upper left corner, the corresponding first n vectors of the orthogonal matrix, which effects the reduction, forms a basis for the stable eigenspace from which the nonnegative definite solution X is then easily found.

Extensions to the basic Schur method have been made [14] (p. 562, p. 573) which were prompted by the following situations:

- G in Equation 37 is of the form $BR^{-1}B^T$ where R may be nearly singular, or G_2 in Equation 38 may be exactly or nearly singular.
- A in Equation 38 is singular (A^{-1} is required in the classical approach involving a symplectic matrix which plays a role analogous to Equation 40).

This resulted in the generalized eigenvalue approach requiring the computation of a basis for the deflating subspace corresponding to the stable generalized eigenvalues. For the solution of Equation 37, the generalized eigenvalue problem is given by

$$\lambda \begin{bmatrix} I & 0 & 0 \\ 0 & I & 0 \\ 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} A & 0 & B \\ -Q & -A^T & 0 \\ 0 & B^T & R \end{bmatrix}; \tag{41}$$

for Equation 38, the corresponding problem is

$$\lambda \begin{bmatrix} I & 0 & 0 \\ 0 & A^T & 0 \\ 0 & {G_1}^T & 0 \end{bmatrix} - \begin{bmatrix} A & 0 & -G_1 \\ -Q & I & 0 \\ 0 & 0 & G_2 \end{bmatrix}. \tag{42}$$

The extensions above can be generalized even further, as the following problem illustrates. Consider the optimal control problem

$$\min \frac{1}{2} \int_0^{+\infty} [x^T Qx + 2x^T Su + u^T Ru]dt \tag{43}$$

subject to

$$E\dot{x} = Ax + Bu. \tag{44}$$

The Riccati equation associated with Equations 43 and 44 then takes the form

$$\begin{aligned} &E^T X B R^{-1} B^T X E - (A - BR^{-1}S^T)^T X E \\ &- E^T X (A - BR^{-1}S^T) - Q + SR^{-1}S^T = 0 \end{aligned} \tag{45}$$

or

$$\begin{aligned} &(E^T X B + S) R^{-1} (B^T X E + S^T) \\ &- A^T X E - E^T X A - Q = 0. \end{aligned} \tag{46}$$

This so-called "generalized" Riccati equation can be solved by considering the associated matrix pencil

$$\begin{pmatrix} A & 0 & B \\ -Q & -A^T & -S \\ S^T & B^T & R \end{pmatrix} - \lambda \begin{pmatrix} E & 0 & 0 \\ 0 & E^T & 0 \\ 0 & 0 & 0 \end{pmatrix}. \tag{47}$$

Note that S in Equation 43 and E in Equation 44 are handled directly and no inverses appear. The presence of a nonsingular E in state-space models of the form Equation 44 adds no particular difficulty to the solution process and is numerically the preferred form if E is, for example, near singular or even sparse. Similar remarks apply to the frequency response problem in Equations 29 and 30 and, indeed, throughout all of linear control and system theory. The stability and conditioning of these approaches are discussed in [14](pp. 529, 573). Other methods, including Newton's method and iterative refinement, have been analyzed in, for example, [14](p. 517). Numerical algorithms for handling Equations 41, 42, and 47 and a large variety of related problems are described in [14](pp. 421, 573). A thorough survey of the Schur method, generalized eigenvalue/eigenvector extensions, and the underlying algebraic structure in terms of "Hamiltonian pencils" and "symplectic pencils" is included in [12], [3].

Schur techniques can also be applied to Riccati differential and difference equations and to nonsymmetric Riccati equations which arise, for example, in invariant imbedding methods for solving linear two-point boundary value problems.

As with the linear Lyapunov and Sylvester equations, satisfactory results have been obtained concerning condition of Riccati equations, a topic of great interest independent of the solution method used, be it a Schur-type method or one of numerous alternatives. We refer to [1] for a further discussion on this.

A very interesting class of invariant-subspace-based algorithms for solving the Riccati equation and related problems uses the so-called matrix sign function. These methods, which are particularly attractive for very large order problems, are described in detail in [14](p. 486) and the references therein. These algorithms are based on Newton's method applied to a certain matrix equation. A new family of iterative algorithms of arbitrary order convergence has been developed in [14](p. 624). This family of algorithms can be parallelized easily and yields a viable method of solution for very high order Riccati equations.

4.8 Pole Assignment and Observer Design

Designing state or output feedback for a linear system, so that the resulting closed-loop system has a desired set of poles, can be considered an inverse eigenvalue problem. The state feedback

pole assignment problem is as follows: Given a pair (A, B), one looks for a matrix F so that the eigenvalues of the matrix

$$A_F = A + BF$$

lie at specified locations or in specified regions. Many approaches have been developed for solving this problem. However, the emphasis is on numerically reliable methods and consideration of the numerical sensitivity of the problem, e.g., see the papers in Part 6 of [14]. Special cases of the pole assignment problem arise in observer design [14](p. 407) and in deadbeat control for discrete-time systems (where $A + BF$ is required to be nilpotent) [14](p. 392). The numerically reliable methods for pole assignment are based on reducing A to either a real Schur form [14](p. 415), or to a Hessenberg or block Hessenberg (staircase) form [14](p. 357, p. 380). The latter may be regarded a numerically robust alternative to the controllable or Luenberger canonical form whose computation is known to be numerically unreliable [14](p. 59). For multi-input systems, the additional freedom available in the state-feedback matrix can be used for eigenvector assignment and sensitivity minimization for the closed-loop poles [14](p. 333). There the resulting matrix A_F is not computed directly but instead the matrices Λ and X of the decomposition

$$A_F = X\Lambda X^{-1}$$

are computed via an iterative technique. The iteration aims to minimize the sensitivity of the placed eigenvalues λ_i or to maximize the orthogonality of the eigenvectors x_i.

Pole assignment by output feedback is more difficult, theoretically as well as computationally. Consequently, there are few numerically reliable algorithms available [14](p. 371). Other work on pole assignment has been concerned with generalized state space or descriptor systems.

The problem of observer design for a given state-space system $\{A, B, C\}$ is finding matrices T, A_K, and K so that

$$TA_K - AT = KC \tag{48}$$

whereby the spectrum of A_K is specified. Because this is an underdetermined (and nonlinear) problem in the unknown parameters of T, A_K, and K, one typically sets $T = I$ and Equation 48 then becomes

$$A_K = A + KC$$

which is a transposed pole placement problem. In this case the above techniques of pole placement automatically apply here. In reduced order design, T is nonsquare and thus cannot be equated to the identity matrix. One can still solve Equation 48 via a recurrence relationship when assuming A_K in Schur form [14](p. 407).

4.9 Robust Control

In the last decade, there has been significant growth in the theory and techniques of robust control; see, e.g., [6] and the references therein. However, the area of robust control is still evolving and its numerical aspects have just begun to be addressed [13]. Consequently, it is premature to survey reliable numerical algorithms in the area. To suggest the flavor of the numerical and computational issues involved, in this section we consider a development in robust control that has attracted a great deal of attention, the so-called H_∞ approach. H_∞ and the related structured singular value approach have provided a powerful framework for synthesizing *robust* controllers for linear systems. The controllers are robust because they achieve desired system performance despite a significant amount of uncertainty in the system.

In this section, we denote by $\mathbb{R}(s)^{n\times m}$ the set of proper real rational matrices of dimension $n \times m$. The H_∞ norm of a stable matrix $G(s) \in \mathbb{R}(s)^{n\times m}$ is defined as

$$\|G(s)\|_\infty := \sup_{\omega\in\mathbb{R}} \sigma_{\max}[G(j\omega)] \tag{49}$$

where $\sigma_{\max}[\cdot]$ denotes the largest singular value of a (complex) matrix. Several iterative methods are available for computing this norm. In one approach, a relationship is established between the singular values of $G(jw)$ and the imaginary eigenvalues of a Hamiltonian matrix obtained from a state-space realization of $G(s)$. This result is then used to develop an efficient bisection algorithm for computing the H_∞ norm of $G(s)$.

To describe the basic H_∞ approach, consider a linear, time-invariant system described by the state-space equations

$$\begin{aligned} \dot{x}(t) &= Ax(t) + B_1 w(t) + B_2 u(t), \\ z(t) &= C_1 x(t) + D_{11} w(t) + D_{12} u(t), \\ \text{and} \quad y(t) &= C_2 x(t) + D_{21} w(t) + D_{22} u(t), \end{aligned} \tag{50}$$

where $x(t) \in \mathbb{R}^n$ denotes the state vector; $w(t) \in \mathbb{R}^{m_1}$ is the vector of disturbance inputs; $u(t) \in \mathbb{R}^{m_2}$ is the vector of control inputs, $z(t) \in \mathbb{R}^{p_1}$ is the vector of error signals, and $y(t) \in \mathbb{R}^{p_2}$ is the vector of measured variables. The transfer function relating the inputs $\begin{bmatrix} w \\ u \end{bmatrix}$ to the outputs $\begin{bmatrix} z \\ y \end{bmatrix}$ is

$$G(s) := \begin{bmatrix} G_{11}(s) & G_{12}(s) \\ G_{21}(s) & G_{22}(s) \end{bmatrix} \tag{51}$$

$$= \begin{bmatrix} D_{11} & D_{12} \\ D_{21} & D_{22} \end{bmatrix} + \begin{bmatrix} C_1 \\ C_2 \end{bmatrix} (sI - A)^{-1} \begin{bmatrix} B_1 & B_2 \end{bmatrix}. \tag{52}$$

Implementing a feedback controller defined by

$$u = K(s)y \tag{53}$$

where $K(s) \in \mathbb{R}(s)^{m_2\times p_2}$, we get the closed-loop transfer matrix $T_{zw}(s) \in \mathbb{R}(s)^{p_1\times m_1}$ from the disturbance w to the regulated output z

$$T_{zw} := G_{11} + G_{12}K(I - G_{22}K)^{-1}G_{21}. \tag{54}$$

Next, define the set $\mathcal{K}$ of all internally stabilizing feedback controllers for the system in Equation 50, i.e.,

$$\mathcal{K} := \{K(s) \in \mathbb{R}(s)^{m_2\times p_2} : T_{zw}(s) \text{ is internally stable}\}.$$

Now let $K(s) \in \mathcal{K}$, and define

$$\gamma := \|T_{zw}(s)\|_\infty. \tag{55}$$

Then the H_∞ control problem is to find a controller $K(s) \in \mathcal{K}$ that minimizes γ. The optimal value of γ is defined as

$$\gamma_{\text{opt}} := \inf_{K\in\mathcal{K}} \|T_{zw}(s)\|_\infty. \tag{56}$$

The original formulation of this problem was in an input-output setting, and the early methods for computing γ_{opt} used either an iterative search involving spectral factorization and solving the

resulting Nehari problem or computed the spectral norm of the associated Hankel plus Toeplitz operator. In a state-space formulation for computing γ_{opt}, promising from the viewpoint of numerical computation, the problem is formulated in terms of two algebraic Riccati equations which depend on a gain parameter γ. Then, under certain assumptions (see e.g. [13] for details), it can be shown that for a controller $K(s) \in \mathcal{K}$ to exist so that $\|T_{zw}\|_\infty < \gamma$, three conditions have to be satisfied, namely, stabilizing solutions exist for the two Riccati equations, and the spectral radius of the product of the solutions is bounded by γ^2. If these conditions are satisfied for a particular value of γ, the corresponding controller $K(s)$ can be obtained from the solutions of the Riccati equations. The optimal gain, γ_{opt}, is the infimum over all suboptimal values of γ such that the three conditions are satisfied.

The approach above immediately suggests a bisection-type algorithm for computing γ_{opt}. However, such an algorithm can be very slow in the neighborhood of the optimal value. To obtain speedup near the solution, a gradient approach is proposed in [13]. The behavior of the Riccati solution as a function of γ is used to derive an algorithm that couples a gradient method with bisection. It has been pointed out in [13] that the Riccati equation can become ill-conditioned as the optimal value of γ is approached. It is therefore recommended in [13] that, instead of computing the Riccati solutions explicitly, invariant subspaces of the associated Hamiltonian matrices should be used.

5 Mathematical Software

A. General Remarks

The previous sections have highlighted some topics from numerical linear algebra and their application to numerical problems arising in systems, control, and estimation theory. These problems represent only a very small subset of numerical problems of interest in these fields but, even for problems apparently "simple" from a mathematical viewpoint, the myriad of details which constitute a sophisticated implementation become so overwhelming that the only effective means of communicating an algorithm is through mathematical software. Mathematical or numerical software is an implementation on a computer of an algorithm for solving a mathematical problem. Ideally, such software would be reliable, portable, and unaffected by the machine or system environment.

The prototypical work on reliable, portable mathematical software for the standard eigenproblem began in 1968. EISPACK, Editions I and II ([7], [15]), were an outgrowth of that work. Subsequent efforts of interest to control engineers include LINPACK [5] for linear equations and linear least squares problems, FUNPACK (Argonne) for certain function evaluations, MINPACK (Argonne) for certain optimization problems, and various ODE and PDE codes. High quality algorithms are published regularly in the *ACM Transactions on Mathematical Software*. LAPACK, the successor to LINPACK and EISPACK is designed to run efficiently on a wide range of machines, including vector processors, shared-memory multiprocessors, and high-performance workstations.

Technology to aid in developing mathematical software in Fortran has been assembled as a package called TOOLPACK. Mechanized code development offers other advantages with respect to modifications, updates, versions, and maintenance.

Inevitably, numerical algorithms are strengthened when their mathematical software is portable because they can be used widely. Furthermore, such software is markedly faster, by factors of 10 to 50, than earlier and less reliable codes.

Many other features besides portability, reliability, and efficiency characterize "good" mathematical software, for example,

- high standards of documentation and style so as to be easily understood and used,

- ease of use; ability of the user to interact with the algorithm,
- consistency/compatibility/modularity in the context of a larger package or more complex problem,
- error control, exception handling,
- robustness in unusual situations,
- graceful performance degradation as problem domain boundaries are approached,
- appropriate program size (a function of intended use, e.g., low accuracy, real-time applications),
- availability and maintenance,
- "tricks" such as underflow-/overflow-proofing, if necessary, and implementation of columnwise or rowwise linear algebra.

Clearly, the list can go on.

What becomes apparent from these considerations is that evaluating mathematical software is a challenging task. The quality of software is largely a function of its operational specifications. It must also reflect the numerical aspects of the algorithm being implemented. The language used and the compiler (e.g., optimizing or not) for that language have an enormous impact on quality, perceived and real, as does the underlying hardware and arithmetic. Different implementations of the same algorithm can have markedly different properties and behavior.

One of the most important and useful developments in mathematical software for most control engineers has been very high level systems such as Matlab, Xmath, Ctrl-C, etc. These systems spare the engineer the drudgery of working at a detailed level with languages such as Fortran and C, and they provide a large number of powerful computational "tools" (frequently through the availability of formal "toolboxes"). For many problems, the engineer must still have some knowledge of the algorithmic details embodied in such a system.

B. Mathematical Software in Control

Many aspects of systems, control, and estimation theory are at the stage from which one can start the research and design necessary to produce reliable, portable mathematical software. Certainly many of the underlying linear algebra tools (for example, in EISPACK, LINPACK, and LAPACK) are considered sufficiently reliable to be used as black, or at least gray, boxes by control engineers. An important development in this area is the SLICOT library, which is described in [1] (p. 60). Much of that theory and methodology can and has been carried over to control problems, but this applies only to a few basic control problems. Typical examples are Riccati equations, Lyapunov equations, and certain basic state-space transformations and operations. Much of the work in control, particularly design and synthesis, is simply not amenable to nice, "clean" algorithms. The ultimate software must be capable of enabling a dialogue between the computer and the control engineer, but with the latter probably still making the final engineering decisions.

6 Concluding Remarks

Several numerical issues and techniques from numerical linear algebra together with a number of important applications of these ideas have been outlined. A key question in these and other problems in systems, control, and estimation theory is what can be computed reliably and used

in the presence of parameter uncertainty or structural constraints (e.g., certain "hard zeros") in the original model, and rounding errors in the calculations. However, because the ultimate goal is to solve real problems, reliable tools (mathematical software) and experience must be available to effect real solutions or strategies. The interdisciplinary effort during the last few decades has significantly improved our understanding of the issues involved in reaching this goal and has resulted in some high quality control software based on numerically reliable and well-tested algorithms. This provides clear evidence of the fruitful symbiosis between numerical analysis and numerical problems from control. We expect this symbiotic relationship to flourish as control engineering realizes the full potential of the computing power becoming more widely available in multiprocessing systems and high-performance workstations. However, as in other applications areas, software continues to act as a constraint and a vehicle for progress. Unfortunately, high quality software is very expensive.

In this chapter we have focused only on dense numerical linear algebra problems in systems and control. Several related topics that have not been covered here are, for example, parallel algorithms, algorithms for sparse or structured matrices, optimization algorithms, ODE algorithms, algorithms for differential-algebraic systems, and approximation algorithms. These areas are well-established in their own right, but for control applications a lot of groundwork remains undone. The main reason we have confined ourselves to dense numerical linear algebra problems in systems and control is that, in our opinion, this area has reached a mature level where definitive statements and recommendations can be made about various algorithms and other developments.

References

References

[1] The Amazing Power of Numerical Awareness in Control, *IEEE Control Systems Magazine*, vol. 24, Feb. 2004.

[2] Anderson, E., Bai, Z., Bischof, C., Demmel, J., Dongarra, J., DuCroz, J., Greenbaum, A., Hammarling, S., McKenney, A., Ostrouchov, S., and Sorenson, D., *LAPACK Users' Guide*, SIAM, Philadelphia, PA, 1992.

[3] Bunse-Gerstner, A., Byers, R., and Mehrmann, V., Numerical methods for algebraic Riccati equations. In *Proc. Workshop on the Riccati Equation in Control, Systems, and Signals (Como, Italy)* Bittanti, S., Ed., Pitagora, Bologna, Italy, 1989, pp 107–116.

[4] Datta, B.N., *Numerical Methods for Linear Control Systems*, Elsevier Academic Press, San Diego, 2004.

[5] Dongarra, J.J., Bunch, J.R., Moler, C.B., and Stewart, G.W., *LINPACK Users' Guide*, SIAM, Philadelphia, PA, 1979.

[6] Dorato, P. and Yedavalli, R.K., Eds., *Recent Advances in Robust Control*, Selected Reprint Series, IEEE, New York, 1990.

[7] Garbow, B.S., Boyle, J.M., Dongarra, J.J., and Moler, C.B., *Matrix Eigensystem Routines—EISPACK Guide Extension*, in *Lecture Notes in Computer Science*, Springer, New York, 1977, vol. 51.

[8] Golub, G.H. and Van Loan, C.F., *Matrix Computations*, 2nd ed., Johns Hopkins University Press, Baltimore, MD, 1989.

[9] Higham, N.J., *Accuracy and Stability of Numerical Algorithms*, 2nd Ed., SIAM, Philadelphia, PA, 2002.

[10] Higham, N.J., *Functions of Matrices. Theory and Computation*, SIAM, Philadelphia, PA, 2008.

[11] Laub, A.J., Survey of computational methods in control theory, in *Electric Power Problems: The Mathematical Challenge*, Erisman, A.M., Neves, K.W., and Dwarakanath, M.H., Eds., SIAM, Philadelphia, PA, 1980, pp 231–260.

[12] Laub, A.J., Invariant subspace methods for the numerical solution of Riccati equations. In *The Riccati Equation* Bittanti, S., Laub, A.J., and Willems, J.C., Eds., Springer, Berlin, 1991, pp 163–196.

[13] Pandey, P. and Laub, A.J., Numerical issues in robust control design techniques, in *Control and Dynamic Systems – Advances in Theory and Applications: Digital and Numeric Techniques and Their Applications in Control Systems*, Leondes, C.T., Ed., Academic, San Diego, CA, 1993, vol. 55, pp 25–50.

[14] Patel, R.V., Laub, A.J., and Van Dooren, P.M., Eds., *Numerical Linear Algebra Techniques for Systems and Control*, Selected Reprint Series, IEEE Press, New York, 1994.

[15] Smith, B.T, Boyle, J.M., Dongarra, J.J., Garbow, B.S., Ikebe, Y., Klema, V.C., and Moler, C.B., *Matrix Eigensystem Routines – EISPACK Guide*, in *Lecture Notes in Computer Science*. Springer, New York, 1976, vol. 6.

[16] Van Dooren, P., Numerical aspects of system and control algorithms, *Journal A*, 30, 1989, pp 25–32.

[17] Wilkinson, J.H., *The Algebraic Eigenvalue Problem*, Oxford University Press, Oxford, England, 1965.