Systems and Computations

Patrick Dewilde, Klaus Diepold and Alle-Jan van der Veen

October 17, 2018

Contents

Chapter 1

Introduction

Part I (Volume I) of 'Systems and Computations' is a first year graduate level introduction to system theory based on a novel systematic computational approach. It is followed, in the next parts, by a collection of further generalizations and a number of worked out concrete applications in various domains. The whole series of three parts is intended to provide for an easily accessible, reasonably complete and theoretically fully motivated computational workbench for electrical engineers, numerical analysts and signal processing engineers, who need to understand how systems behave and how they can be computationally mastered.

With reference to the immense literature on system theory, control theory, optimization and filtering, it is hard to believe that there would be such a thing like an easily accessible computational approach to all these topics, given the variety of mathematical methods people have developed to attack the various fields pose in the area of system behavior. But there is! Such an approach has gradually been appearing through a lot of searching for the 'real basics' in the various fields mentioned.

The ability to compute effectively *and* to connect computations to fundamental issues in system's engineering is a major challenge to the ability of our future engineers, whether they are more active in engineering or in data numerics, see fig. 1.1. One of the most appealing properties of our approach is the fact that computations and system theory *coincide* when properly considered! A computer executing computations is a system in its own right, even when it is used to perform computations for another, physical system.

To do this unification of fields right, we merge three main bodies of knowledge (system theory, numerical algebra and signal processing) in one core,

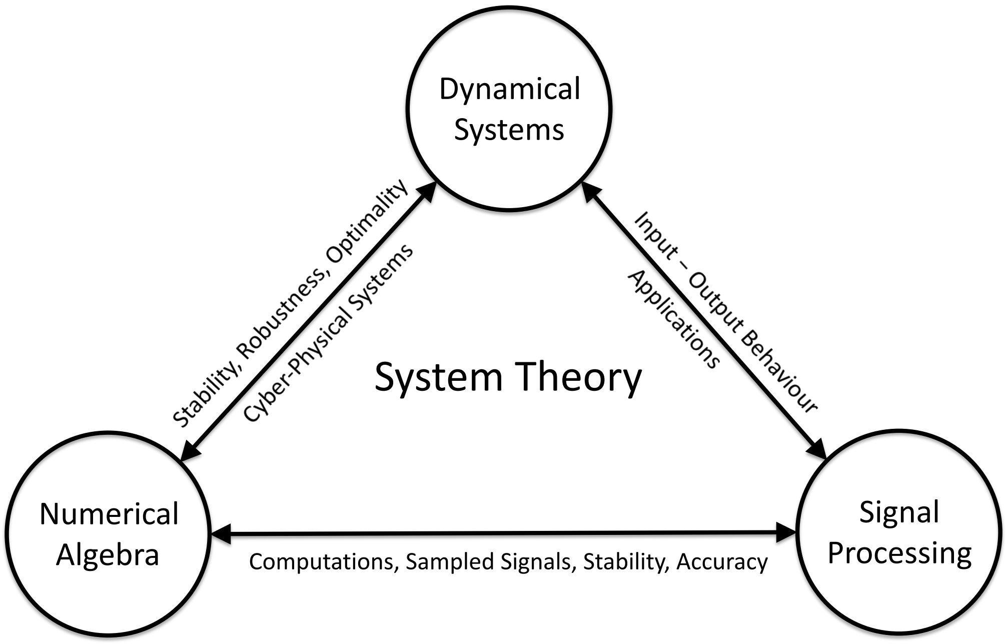


Figure 1.1: Our main disciplines are closely related!

with linear algebra playing the role of cement that glues everything together. It turns out that the linear algebra needed to achieve this feat is mostly if not wholly elementary. Simple matrix operations play the central role, but like wild horses, they have to be tamed to behave in a constructive way. That is the role numerical analysis plays in the set up.

The challenge to students and researchers studying system theory is the mastery of elementary matrix algebra, and the 'geometry' connected to it. A matrix defines a linear 'operator', and the range as well as the kernel of the matrix as operator play the central role. Most algebraic operations aim at characterizing ranges and kernels, mostly by deriving orthonormal bases for them, an operation that in signal processing terms may be called 'orthogonal filtering'. Depending on the application at hand, whether realization theory, optimal control or estimation theory, the filtering procedure goes with an idiosyncratic name and is called e.g., dynamic programming, a Wiener filter, a Kalman filter or a selective filter (in electrical engineering), or QR factorization, or SVD in numerical algebra.

Conversely, the study we propose will equip the student with a wealth of examples in which they can refine their intimate knowledge of matrix algebra as the main mathematical vehicle to achieve key results. Familiarity with matrix algebra is a 'to be or not be' question for a modern engineer who has to handle data in a variety of situations, and the detailed study of system theory offers the ideal environment to develop that familiarity. This can be seen as the main 'end term' of the course we propose.

Prerequisites

We assume knowledge of elementary (undergraduate level) matrix algebra (real and complex numbers, vectors, matrices, matrix products, matrix inversion, eigenvectors and eigenvalues, QR decomposition and SVD) and basic analysis (functions, limits, ordinary and partial differentials, integration, series). Matrices are our 'lingua franca', and our students and readers should develop close familiarity for them (this book will help tremendously!). More advanced mathematical properties (in particular some elementary properties of Hilbert spaces) are introduced where they are used, and this happens only in later chapters. Only in few, specific places more advanced properties are needed and will be explained there. We have added in appendix a systematic introduction to linear (matrix) algebra, the properties and the algorithms used or assumed, both for easy reference and to communicate to the reader 'our' way of thinking on these matters.

Organization

We have attempted to make each chapter correspond with ca. two (sometimes three) hours of lectures or 'workshops'. We have experimented with a format in which students are asked to read the chapter beforehand, do some research on the internet, and then come to class weaponed with observations and questions. The class meeting then starts with an inventory of these, followed by an in depth discussion of critical items. In particular, a lot of attention then goes to 'how' to approach the critical issues rather than to a linear treatment of the material. It takes some discipline from both students and lecturers to work in a "flipped classroom" style. In particular, it takes effort to convince students to put time in the preparation of the material and to convince lecturers not to start right away explaining things during contact hours. What may help is to organize and schedule 'reading meetings', i.e., a cosy room and time slots set apart for preparations. When it works, the method is much more productive than just lectures.

To enhance creative participation and/or communal reading, we have added ‘discussion topics’ to each chapter as well as further exploration topics. Etymologically, ’algebra’ is ‘the art of structuring’, and its practice is quite a different thing than the mere acquisition of knowledge. This is our main goal: motivate and facilitate the development of algebraic practice in dynamic system theory and signal processing, with an important role for numerical analysis (and practice) to play in the process.

Chapters 2-12 cover the basic theory, using the discrete-time Linear Time-Variant (LTV) system theory as the main vehicle. Chapters 2-5 are purely introductory, in that they position our two main application domains ‘dynamical systems’ and ‘signal processing’ both historically and from what is commonly understood by them (semantically), using numerical algebra to convert the ‘narrative’ in concretely useable mathematics—in our case just elementary matrix calculus. Chapters 6-12 then develop all the basic concepts in the context of discrete-time, linear, time-variant systems with finite dimensional state-spaces, including two main application domains: Kalman filtering and Bellman optimal control. They do so strictly using elementary matrix algebra. This is possible because the computer, as a numerical engine, belongs itself to the class of LTV systems, at least so long as linear operations (addition and multiplication with coefficients) are used.

A follow up book then considers many further developments of the theory, mainly in three directions: (1) extensions to other types of systems (non-linear, continuous time or even hybrid) and (2) a variety of (often new) mathematical properties of matrices that can be derived using system theoretical methods (matrix interpolation, approximation theory, model reduction) and (3) a full blown treatment of some important applications (in circuit synthesis, optimization, data processing in telecommunication and optimal control).

The thirteen chapters of the present book form therefore a logical whole and have to be run through systematically and sequentially. Except for chapter 5 that provides the link with classical time-invariant filter theory, they develop the theory of dynamical systems systematically in the LTV context, using concepts that are much more generally applicable, but are made fully concrete thanks to the already mentioned direct connection with numerical analysis and matrix algebra. The basic concepts acquire in this way an eminently concrete meaning and immediately exemplify their power.

A short overview of the first Part

We start out with a motivating chapter to answer the question ‘why it is worthwhile to develop systems theory’. To do so,we jump from the start in the center of our methods, using an eminent and classical topic in optimization: optimal tracking. Although optimization is not our leading subject—system theory is—it definitely provides for one of our main application areas. This motivating chapter describes a straight matrix-algebra approach to the problem, which should become second nature throughout the course. Optimal tracking is based on a powerful principle called ‘dynamic programming’, which is a great and very practical method in its own right, but its recursive character also provides for a powerful link between optimization and the central ideas behind system theory. The chapter then illustrates the technique further with an appealing toy example that embodies in a most elementary way all the central ideas, and ends with a short introduction to our graphical method of choice to represent the resulting computational architectures.

Chapter 3 then moves into a more philosophical mood to introduce the basic notions on which system theory is based. In this endeavor, it follows the insights first provided by the late Rudy Kalman and his colleagues in [26], but it does so mostly as a narrative, without going into a resulting general mathematical treatment. The reason is that, on the one hand, a couple of fundamental, and very appealing notions such as ‘state’, ‘behavior’, ‘reachability’, ‘controllability’ suffice as a basis for the whole theory, but, at the other, we want to keep the mathematical treatment concrete. In further chapters we shall gradually develop the mathematical consequences of the concepts introduced here, giving priority to the matrix algebra approach, but expanding on them where interesting or necessary.

In chapter 4, and based on the narrative given in chapter 3, we spend some time introducing the reader to the main types of systems they may encounter in practice. The chapter puts some more relief in the possible contexts in which systems and their dynamics may arise, and gives some arguments for the proposition that there is at least one reasonably general type of system to which many of the more complex types (continuous time, non-linear, distributed) have to be reduced, as soon as concrete computations on systems’s properties or behavior have to be executed, namely discrete time, Linear Time-Variant systems (LTV systems).

Chapter 5 then starts to develop our general ideas in this central LTV prototype environment, an environment that appears to coincide perfectly with the linear algebra or matrix algebra context, making the correspondence systems-computations a pregnant reality. At the same time, we develop our algebraic formalism systematically. People familiar with the classical approach in which z-transform or other types of transforms are used will easily recognize the notational or graphic resemblance, but there is a major difference: everything remains elementary, no function calculus is involved and only the simplest matrix operations (addition and multiplication) are needed. The algebra just follows the system theoretical and at the same time computational processing needed. Particularly appealing expressions for the state space realization of a system appear and the global representation of the input-output operator in terms of four block-diagonal matrices $\{A, B, C, D\}$ and the 'causal shift' Z. The consequences for and relation to Linear Time Invariant (LTI) systems and infinitely indexed systems are also considered in *-sections, which can be skipped by students or readers not interested in these topics.

From this point on, we are ready to tackle some main issues in system theory. The very first, considered in chapter 6, is the question of *system identification*. It is the problem of deriving the state space equations from input-output data. The problem is, in the first place, the derivation of the state space evolution equations and output equations from the input-output (transfer) operators or input-output matrix. In this chapter, only the causal, or block-lower triangular case is considered, although the theory applies just as well to an anti-causal system, for which one just has to let the time run backwards and apply the same theory in a dual way.

In chapter 7, an important and central issue is considered: that of minimality of state space system representations. This is perhaps the most classical issue in system theory, with a big pedigree starting from Kronecker to its solution in time-variant systems. The question introduces important basic operators and spaces related to systems in general and, in a strong way, linear systems in particular. In the time-variant context (and by extension in all other contexts), what we call the 'Hankel operator' plays the central role, and in particular, a minimal factorization of it in a 'reachability operator' and an 'observability operator'. From our LTV treatment, the corresponding results for LTI systems (a special case) and infinitely indexed systems follow easily, but they entail some extra complications, which are not essential for the main treatment offered and can be skipped on first reading.

Chapter 8 then dives into elementary matrix operations that exploit the recursive structure of their state space representations if such is available or derived, to wit: addition, multiplication and elementary inversion. The latter when possible, of course—the issue of generalized inversion will occupy us in more advanced chapters in Part 2. The chapter also introduces some special types of matrices that play an important role for orthogonalization, as was already exemplified in our motivating first chapter 2, namely inner matrices. Their relation to system realization theory is worked out in the next two chapters.

As we already observed in the case of the Hankel operator, factorizations play a central role in system theory, and there are several types of them, depending on what is factorized, and what form the factors have to take. In this section we develop one type of factorization that is of great importance for the characterization of an LTV system, and which is traditionally called 'coprime' factorization. Because not all such factorizations have to be coprime (when non-minimal realizations are to be considered, which may either be needed or just happens), we prefer the term 'external factorization' (in contrast to inner-outer which is the topic of the next chapter). Exernal and coprime factorizations play an essential role in control theory, but that will be considered in advanced and application chapters in Parts 2 and 3.

Chapter 10 considers perhaps the most important operation in all time-variant system theory: inner-outer (and its dual, outer-inner) factorization. This factorization plays a different role than the previously treated external or coprime factorization, in that it involves properties of the inverse or pseudo-inverse system(s), computed on the state-space representation of the original. Inner-outer (or outer-inner) is nothing but recursive 'QL-factorization', as we already observed in our motivational chapter 2, and outer-inner is recursive 'RQ-factorization'. This type of factorization will play the central role both in a variety of applications (like optimal tracking, state estimation, system pseudo-inversion and spectral factorization), all topics that are individually treated in further chapters.

The set of basic topics then concludes in Part 1 with two major and elementary application domains of our theory: linear least squares estimation (llse) of the state of an evolving system (Kalman filtering) and its dual quadratically optimal linear tracking (Bellman filtering), both straight applications of the inner-outer theory. Further deepening of these in several directions as well as many more applications are treated in the following parts of this trilogy.

In appendices we give a systematic overview of the linear algebra we use and we also discuss some methods used in LTI theory, with an emphasis on the transition from LTI to LTV. This chapter is intended for making this transition natural to students who were trained in classical electrical engineering methods, and for whom our systematic matrix-based may seem outlandish.

A motivation for the approach

In our present technical world, systems and computers are ubiquitous. We may experience them as different, but in many ways, the notions overlap. Technical systems are often steered or controlled by an embedded computer that executes some algorithm, and, conversely, a computer executing a computational or algorithmic task can be viewed as a (dynamical) system in its own right. This correspondence can be exploited to great benefit for the understanding of the behavior of a system on the one hand, and, conversely, the development of efficient computations on the other. This is also the key idea behind the present book.

Traditionally, system theory, numerical algebra and signal (or data) processing have developed and evolved in very different ways. We show in these volumes that bringing the fields together has great advantages both from a theoretical and a practical point of view. It turns out that only very few and absolutely elementary basic concepts and numerical methods are needed to cover the whole field, provided one is willing to leave the old ways behind and look at the key issues in a novel way.

The insight that such an approach was possible gradually arose, first in the seminal work of Bellman and Kalman, who dared to dispose of the classical, Fourier and Laplace transform based theory into the new environment of time variant systems, but then were forced to re-incorporate their views in the classical framework of time invariant algebraic structures like modules or Hardy spaces, lacking an effective time variant algebra. The latter came up gradually, at the mathematical side as 'nest algebras' and at the numerical side as 'semi-separable systems'. It took a while to harmonize the two approaches, but that is now the case and the present book offers a fully didactical introduction to the unified theory.

The basis for the transformation or rejuvenation of the field is remarkably simple (but it took a long time to develop it systematically). The late Rudy Kalman had the vision that 'the state of a system is what the system remembers of its past', or, dually, 'the minimal information needed at a given moment to produce its future development, given future inputs as well.' Mathematically, this information presents itself as 'Nerode equivalent classes', and, depending on the formalism one adopts, as an 'ideal' or a 'submodule' or a 'restricted shift invariant subspace'. It was thought for a long time that these notions could not extend beyond a time invariant framework.

At the numerical algebra side, the power of orthogonal transformations gradually became evident, exemplified by the resurgence of QR-factorization and the Singular Value Decomposition (SVD), insights going way back to Jacobi and Gauss, but then resurrected by numerical analysists like Givens and Householder. It turns out that the only really central concept needed is that of 'range of an operator'. For system theoretical purposes, the 'operator' that characterizes the transition from past to future is what is called the 'Hankel operator', and its range or co-range plays the role of state space in the Nerode-Kalman view on systems.

In the time-invariant setting, the Hankel operator is simply a Hankel matrix, that is, a matrix with identical elements along the anti-diagonals (or NE to SW diagonals), but in the more general matrix setting, it has a somewhat more involved recursive structure that reflects the shift invariance of its kernel. This observation is the key to the development of a unified and extremely powerful time variant theory. Translated to numerics, the property amounts to efficient recursive QR or SVD, efficient because the numerical complexity turns out to be dominated by the dimension of the state space (which is often limited) rather than the dimension of the system's input-output relation (which may very well be infinite).

In the present book, we treat matrix operations the way we treat a dynamical system, that is, respecting its recursive order in time. General matrix computations can just be viewed as operations on a numerical system and, conversely, time variant dynamical systems produce general matrices as their numerical behavior. The notions 'matrix' plus 'linear order in time' or 'indexing order' coincide with the notion 'discrete time dynamical system' in this approach.

Basic matrix operations are operations on systems and vice versa. For the development of the whole theory only the most elementary matrix operations (additions and multiplications) are needed. The large majority of results ranging from system identification to system optimization, estimation and model reduction can remarkably be obtained without anything more than elementary matrix operations. Conversely, the approach often leads to novel matrix methods as well, especially in the area of matrix approximation and interpolation.

The result is that time variant system theory and elementary matrix calculus largely coincide. This is a major didactical advantage of the approach. No complex function calculus, transform theory, module theory or whatever complicated algebraic structures are needed for most of the results, provided one stays within the context time-variant systems and matrices. On the other hand, with some extra effort all the time invariant results can be derived from the time variant basics, but they often require the solution of an additional fixed-point problem. Historically, the time invariant way was seen to be simpler and more insightful. The opposite is true: invariance complicates matters considerably and may be seen as 'unnatural'. By going time variant one can sidestep most of the classical literature in favor of a straight and simple matrix theory—a great logical and didactical simplification.

A further step in which the power of the time variant approach is exploited is towards non-linear systems. A non-linear system is only a differential away from a linear, time variant system. This fact leads to the applicability of most time variant results in the non-linear context, but there is a catch, and that is that the opposite direction, from time variant result to non-linear result is far from trivial in many cases. Nonetheless, as already mentioned, range theory remains the main ingredient and our main workhorse, inner-outer factorization works non-linear as well, but the whole non-linear development necessitates some differential geometry, which is often viewed as non-elementary. Therefore, these topics will only come into play in later, more specialized chapters falling outside the scope of our first part.

Notation

We mostly use standard algebraic notation, a survey of which is in the Appendix. We do, however, use special notations for mathematical objects that occur often in our developments, and/or to avoid annoying overloads of symbols (which sometimes cannot be avoided). Here is a short summary of non-standard notations used in this book:

- Many of our objects are matrices, and we have often to consider either special indexing conventions or take out submatrices from a given matrix. As such operations can become unwieldy, we adopt systematically a MATLAB-like annotation of index ranges. Suppose A is a matrix, then $A_{k:\ell,m:n}$ is a submatrix of A consisting of a selection of rows from index k to ℓ (inclusive) and columns from index m to n inclusive. If such columns are not there originally, they are just considered empty (see the next item). We also systematically economize the notation (in contrast to many textbook): A, a and $\mathbf{A}$ may e.g., be different matrices with $A_{i,j}$, $a_{i,j}$ and $\mathbf{A}_{i,j}$ different elements of each respectively.

- Very often we consider block matrices, i.e., matrices whose entries are themselves matrices (called 'blocks'). Blocks may consist of blocks themselves, but they are indexed in the usual fashion and have to be commensurate (dimensions in rows or columns must match throughout). For example: $A_{k,\ell}$ may be a block in a block matrix A, and $[A_{k,\ell}]_{m,n}$ is then a block entry at the position (m, n) in that original block. We do allow blocks with zero dimensions: they are just empty, but do have index numbers. We do have some peculiar notation for such a situation: an entry of dimension $(0, 1)$ is denoted '$-$', an entry of dimension $(1, 0)$ is denoted '$|$', and one of dimension $(0, 0)$ is denoted '$\cdot$' (these are actually place holders: they have indices but no entries). Special (logical) computational rules apply for such entries (they are introduced in the text where this type of notation is first used). This extension of matrix algebra appears to be very useful: in many matrix operations (in particular reductions and approximations) one cannot say before hand whether certain entries survive the operation (for example: deleting rows or columns of zeros in a matrix). Just like the introduction of the empty set $\emptyset$ and the number 0 proved extremely useful in set theory and algebra, so are indexed empty entries. They also correspond to the empty symbol $(\perp)$ often used in computer science.

- In the literature, many notations are used to indicate the transpose of a real matrix or the hermitian transpose (conjugate transpose) of a complex matrix. In this book and as in MATLAB, we shall only use real or complex arithmetic and indicate these transpositions by a single symbol, namely a single accent (i.e., A' is in all cases the hermitian transpose of A. In the real case it is then also just the transpose.). A motivation for this choice is 1., the overload of the symbols T or H, as we have special use for those (we do not like the notation T^T for the transpose of T nor H^H for the hermitian transpose of the Hankel matrix H!) and 2., some consistency with MATLAB, however with the proviso that in MATLAB the accent is just a transpose (not a transpose conjugate)—but as most of our computations are real, this should not be a problem. We reserve the star (for example, with a an operator, a^*) for the dual of an object in a context where duality is well defined (a dual is not necessarily a transpose, although it often will be). Sometimes, the dual of a matrix is indeed its transpose, but that should be clear from the context. We use tildes and hats as normal typographical symbols—they do not have any other meaning than to characterize the object under discussion.

- We use 'constructors' systematically, following a good habit of computer science. Constructors are written in normal font. For example 'col' is the column constructor. It takes a sequence of elements (e.g., numbers or blocks with appropriate dimensions) and makes a column out of the them. For example: $\mathrm{col}(u_1, u_2) = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$ (compare this with $\begin{bmatrix} u_1^T & u_2^T \end{bmatrix}^T$). Further constructors are 'row', 'Toeplitz', 'Hankel' etc. (often abbreviated).

- Names and formulas: the general convention is that 'names' are written in 'regular font'. For example: 'ker' or 'ran' are names for functions that return respectively the kernel or the range of an operator, while ker is the product of k, e and r. This convention allows to use indexed names as well: so is K7 just a name, while the notation K_7 assumes K to be a vector, whose element with index 7 is K_7. A new name occurring in the text, a theorem or a proof is introduced with the symbol ':='; example: $T := D + C(I - ZA)^{-1}ZB$ defines T in terms of the right hand side symbols (assumed to be already defined).

- Another convenient notation that we shall use extensively is a short hand for system realizations. Z is systematically used as the 'forward' or 'causal' shift, with its conjugate Z' the 'backward' or 'anti-causal' shift. A causal, linear time variant (LTV) system T may have a 'realization' $T = D + C(I - ZA)^{-1}ZB$ for which we use the shorthand

$$T \sim_c \begin{bmatrix} A & B \\ C & D \end{bmatrix}. \tag{1.1}$$

Similarly, and anticausal LTV system may have a realization $T = D +$

$C(I - Z'A)^{-1}Z'B$, with the shorthand

$$T \sim_a \begin{bmatrix} A & B \\ C & D \end{bmatrix}. \tag{1.2}$$

For purists: Z is a generic operator or constructor. It just shifts the indexing scheme, but stands also for a generic collection of shift matrices, which have a different effect whether applied to the left or the right of an indexed object. E.g., suppose u is an indexed column, then Zu is again an indexed column whose elements are $(Zu)_k = u_{k-1}$. In our formalism, the dimensions of the u_k may vary (and even become empty), and the dimensions of the entries in Z, interpreted as a matrix, have to adapt. This question is addressed at length in the chapter on LTV systems. Some constructors can be represented as matrices. What characterizes a constructor is that it entails an organizational operation rather than a numerical operation.

- We also need shifts along diagonals. We denote $A^{<+1>}$ as a 'forward' diagonal shift on a matrix, i.e., a shift in the South-East direction, with as its conjugate the 'backward' diagonal shift denoted as $A^{<-1>}$, i.e., a shift in the North-West direction.

- We shall also occasionally use 'continuous products', defined for integers $i > k$ as $A^{>}_{i,k} = A_{i-1} \cdots A_k$ with $A_{i,i} := I$ (notice: this convention may differ with what is done in the literature!).

Acknowledgements

This book would not have been possible without the collaboration with and influence of many colleagues, students, assistants and relatives. A full list would be prohibitively long, and the short list that we are giving here may (or even will) overlook important contributors, a fate that appear unavoidable. Nonetheless, and with excuses for our oversights, here are persons whom we definitely want to thank for their contributions to our knowledge and insights.

In the first place we would like to commemorate the massive seminal contributions of the giants of our field who passed away in recent times: Rudy Kalman, Jan Willems, Alfred Fettweis and Uwe Helmke.

From a methodological point of view, we have been most strongly influenced by Tom Kailath and quite a few of his students. Besides Tom, who

really pioneered the algebraic and computational approach to system theory, we have been most strongly influenced by his students Martin Morf, Sun-Yuan Kung, George Verghese, Hanoch Lev-Ari, Ali Sayed and Augusto Vieira. Our contacts with the ISL group in Stanford have always been exceedingly productive, and we cannot sufficiently be grateful to Tom for mostly making them happen. We were gracefully invited to heaven, so many times.

Quite a few colleagues contributed ideas and specificities to our work, besides those already mentioned. In particular we wish to mention Bob Newcomb, Vitold Belevitch, Brian Anderson, Harry Dym, Bill Helton, Israel Gohberg, Rien Kaashoek, Daniel Alpay, Jo Ball, Shiv Chandrasekaran, Yuli Eidelman, Paul Fuhrmann, Keith Glover, Malcolm Smith, Leon Chua, Thanos Antoulas, Michel Verhaegen, Justin Rice, Tryphon Georgiou, Eugene Tyrtyshinkov, Eric Verriest, Jan Zarzycki, Geert Leus, Lang Tong and quite a few of their students.

Also many of our students were instrumental in shaping the theory we are presenting or contributing examples. Most directly involved in the systems theory area were Joos Vandewalle, Paul Van Dooren, Adhemar Bultheel, Ed Deprettere, Prabhakar Chitrapu.

Last, but not least we would like to acknowledge the inspiring contributions from colleagues who accompanied us through many years by collaboration, inspiration, discussion and friendship, notably Rainer Pauli, Wolfgang Mathis, Albrecht Reibiger, Harald Martens and Sunil Tatavarti.

Part I

Part I: Basics and First Examples

Chapter 2

Optimal quadratic tracking

Our first chapter is intended to be purely motivational, while also covering an important and highly useable general topic, dynamic programming. Dynamic programming is a perfect vehicle to start appreciating the power of recursion, which is also why system theory is such an important engineering topic. Recursion is where systems and computations meet. We start out by introducing one of the most powerful operations of linear algebra, the Moore-Penrose inverse, which we shall use extensively to solve least squares optimization problems throughout the book. Next, we describe the prototype optimal quadratic tracking situation, and use Moore-Penrose to find an optimal recursive solution. This approach is known as a case of *dynamic programming* and it exhibits the importance of the notion of state, which we shall pursue intensely in the further chapters. The following section is then devoted to a toy example, deceptive in its simplicity but very instructive to build dynamic programming intuition. The core of the chapter then ends with the introduction of our method of choice to map computations into architectural implementations, here applied to the dynamic programming situation.

Menu

Hors d'oeuvre

The Moore-Penrose inverse

First course

The Bellman problem: optimal quadratic tracking of a dynamical system

Second course
A toy example of
the power of dynamic programming

Third course
Unbiased architectural representations

Dessert
Notes

2.1 Preliminary: the Moore-Penrose inverse

Let us consider an overdetermined system of linear equations $b - Ax = e$ in which A is an $n \times m$ matrix of linearly independent columns, $n \geq m$ (necessarily), x is an unknown vector of dimension m, b a given vector of dimension n, and the goal is to find x so that the quadratic norm of the 'error' or 'cost' e, namely $\|x\|_2 = \sqrt{e'e}$ with e' the transpose of e (we use MATLAB notation as much as possible) is minimum. We have:

Proposition 1 *The solution to the minimization problem* $\text{argmin}_x \|b - Ax\|_2$ *with A an $n \times m$ matrix ($n \geq m$) with independent columns is unique and given by*

$$x_{min} = A^\dagger b, \tag{2.1}$$

in which $A^\dagger := (A'A)^{-1}A'$. $A^\dagger$ is, by definition, the Moore-Penrose inverse of A.

Proof

(We follow the traditional 'Wiener' orthogonality argument). Whatever x of dimension m, Ax will lie in the linear space (hyperplane) generated by the columns of A, i.e., the *range* of A. The 'best' x_{min} will then be such that the least squares error $e_{\text{min}} = b - Ax_{\text{min}}$ is orthogonal on the range space of A, i.e., we should have

$$A'(b - Ax_{\text{min}}) = 0 \tag{2.2}$$

and hence $x_{\text{min}} = (A'A)^{-1}A'b$ since $A'A$ is an $m \times m$ non-singular matrix thanks to the assumed independence of the columns of A. (Notice: Any

x will have $b - Ax = (b - Ax_{\min}) + h$ for $h = A(x_{\min} - x) \perp e_{\min}$, see fig. 2.1, and hence $\|b - Ax\|^2 = \|e_{\min}\|^2 + h^2 > \|e_{\min}\|^2$ when $h \neq 0$.) QED

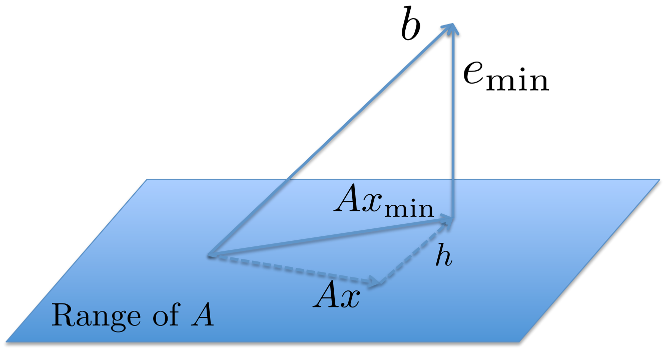


Figure 2.1: Best linear quadratic approximation

Example

Subpose we have two measurements of a quantity x, the first giving $x = 9$ and the second $x = 11$. What is the 'best' x in the least squares sense? Writing the measurements in matrix form gives $b - Ax = e$ with $A = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ and $b = \begin{bmatrix} 9 \\ 11 \end{bmatrix}$. We find $A'A = 2$ and $A^{\dagger} = \frac{1}{2}\begin{bmatrix} 1 & 1 \end{bmatrix}$ and hence $x_{\min} = 10$ with $e_{\min} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$, the overall square root error being $\sqrt{e'_{\min}e_{\min}} = \sqrt{2}$ as one would expect.

This is the basic 'geometric' result that we use in most optimization problems that we shall consider, since most of them will be of the least squares type. Still, a number of remarks and/or refinements can be considered, to wit:
- A' is an $m \times n$ matrix so the dimension of $A'b$ is the same as that of x. $\Pi_A := A(A'A)^{-1}A'$ is the orthogonal projection operator on the range of A, and we shall sometimes write $\widehat{b} := \Pi_A b$. $\widehat{b}$ acquires a special meaning as llse or *linear least squares estimate of* b in a stochastic estimation context.
- In case the columns of A are not linearly independent, more work has to be

done to solve the minimization problem, which also does not have a unique solution any more. We refer to the algebraic introduction at the end of this book for details.
- The QR-algorithm applied to A produces a factorization of the form

$$A = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R \\ 0 \end{bmatrix} \tag{2.3}$$

in which $Q = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix}$ is an $n \times n$ orthogonal matrix and R a non-singular $m \times m$ upper-triangular matrix (see the algebraic introduction at the end of the book). The columns of Q_1 form an orthonormal basis for the range of A, while the columns of Q_2 form an orthonormal basis for the kernel of A', also known as the *co-kernel* of A. When we dispose of such a QR-factorization, then we can immediately write $A^\dagger = R^{-1}Q_1'$. QR is not the only possibility for such a result, we could (and shall) also use a *QL* version of the same type of algorithm, writing $A = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} 0 \\ L \end{bmatrix}$, in which Q (different from before!) is still an orthogonal matrix and L is a non-singular lower triangular matrix. In this latter case we shall have $A^\dagger = L^{-1}Q_2'$. Both R and L can be seen as 'compressed' versions of the rows of A with a special (upper or lower) structure.
- QR (or QL) are not the only possibilities to obtain the range basis. A numerically more refined method is the *singular value decomposition* also described in the algebraic introduction.

Example

In the previous example we have $A = \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} \sqrt{2} \\ 0 \end{bmatrix}$, giving orthonormal bases for both the range of A and the *co-kernel of* A, which is the kernel of A'.

2.2 The Bellman tracking model

By way of introduction and provision of a specific instance of motivation for the development of system theory, let us consider the problem of optimal quadratic state control or optimal tracking of a linear dynamical system, as originally defined and studied by Bellman [7]. Throughout this book, we work with signals discretized in time (for brevity, we skip the standard step of discretization here, which typically could be done using a trapezoidal integration rule to protect numerical stability. We treat the issue in a more advanced chapter in Part 2). Our dynamical system will possess a *state*, which summarizes its past at any given indexed time instant k and is assumed to be a vector of dimension η_k, denoted as x_k. Moreover, let the system be driven by an input u_k at index k, where u_k is a vector of dimension m_k, and let the evolution of the state of the system at k be given by linear state equations (we use a state space formalism throughout, because the systems we consider will mostly be time variant)

$$x_{k+1} = A_k x_k + B_k u_k \tag{2.4}$$

in which A_k is a $\eta_{k+1} \times \eta_k$ and B_k a $\eta_{k+1} \times m_k$ matrix. A simple *signal flow diagram* as shown in fig. 2.2 is often used to represent such a calculation (we shall extend such representations by a *functional model* data model to be defined in section 2.4).

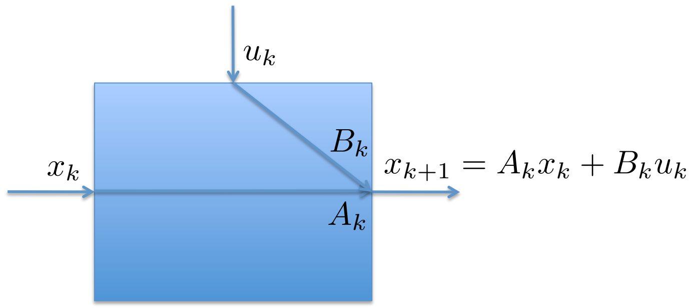


Figure 2.2: Signal flow model of the state evolution at index k

We assume furthermore that our system starts at index 0 with a given initial state x_0 and that we wish to control the evolution of our system in the interval $[0, n+1]$ where n is a final index, so as to minimize a positive quadratic cost function restraining states and inputs.

For simplicity and in order to represent a quadratic cost adequately, we write the cost of a state x_k as $x_k' M_k' M_k x_k$ in which M_k is a matrix of appropriate dimensions and the accent indicates real or complex conjugation, making x_k' a row vector, in line with MATLAB, whose notation convention we use throughout (often the cost is defined by a strictly positive definite matrix C_k and one may take $M_k = C_k^{1/2}$ or just as well a Cholesky factor but C_k may be non-strictly positive definite and M_k may be a rectangular factor; precise conditions on the available latitude will be derived later.). Similarly, the cost of an input u_k will be $u_k' N_k' N_k u_k$, and we assume N_k square non-singular (so that arbitrarily large inputs will not be possible). We assume, in addition, that the system description is *minimal*, which, in this case, just means that all possible states x_k at any relevant index point k can be reached with an appropriate sequence of past inputs, a technical condition necessary to make the algorithms presented work, but about which we shall not worry in this first approach, because a non-minimal model can always be reduced to a minimal one by a standard procedure that we shall present in the chapters on system realization.

Note

One of the original motivations for considering a quadratic optimization model was the Apollo mission: how to get a rocket to the moon with minimal expenditure of fuel? It takes, of course, some work to reduce the Apollo mission problem to the simple model presented above. One must first find a potentially optimal trajectory that respects gravity laws and that requires a minimal nominal amount of fuel to reach the goal within a domain of feasibility defined by various limits in time and fuel needed. Once settled on such an optimal trajectory, the control problem is to keep the rocket close to the optimal trajectory with minimal expenditure of fuel, even though various inaccuracies may have occurred producing (stochastic) deviations. This is achieved by controlling the *deviation* of the state from the desired optimal, and using the control to bring the rocket closer to the nominal optimal trajectory without spending too much fuel. When sufficiently small, the deviation of the state will satisfy a linear differential model derived from the optimal trajectory. After discretization, a model of the type given above is obtained (in a much later chapter we shall study discretizations to some extent.).

Dynamic programming

The problem of optimal quadratic control in the given set up is solved by *dynamic programming*. The basic idea of dynamic programming is: *once the system has gotten to whatever state x_k at time index k, the trajectory* *has to be optimal from there on*—for, if that were not the case, there would be a better overall trajectory just by replacing the segment from k on by a less costly path. This means, in particular, that all optimal inputs and costs, when started from whatever x_k at time index k are only dependent on x_k (and not on previous states), given the model of course (or, to put it differently, all the future inputs have to be chosen so as to optimize the trajectory from index k on, and hence are only dependent on x_k and the model). Let us prove this assertion now for the quadratic error case (a general proof works the same way, but in the quadratic case we get a more specific result).

For this we introduce the *recursion hypothesis:*

> *the total optimal cost starting from any x_k in the remaining interval $[k, n+1]$ is given by a quadratic form $c_k' c_k := x_k' Y_{k-1}' Y_{k-1} x_k$ with $c_k := Y_{k-1} x_k$, in which Y_{k-1} is a to be computed $\eta_k \times \eta_k$ matrix.*

The recursive hypothesis will be verified if 1. it is valid at end point $k = n+1$ and 2. when valid for $k+1$ it is valid for k. Point 1. is obvious, because at the end point $n+1$ the cost is nothing but $x_{n+1}' M_{n+1}' M_{n+1} x_{n+1}$, so $Y_n = M_{n+1}$ and the recursion we derive now will verify point 2. (*remark*: the '-1' in Y_{k-1} is historically motivated).

Key to the method is the determination of a *cost model*: put the 'square roots' $Y_k x_{k+1}$, $M_k x_k$ and $N_k u_k$ of the cost terms as outputs in the model. Next, assume recursively that the optimal cost, from $k+1$ on, and for any x_{k+1}, is given by $x_{k+1}' Y_k' Y_k' x_{k+1}$, the cost model at k (shown in fig. 2.3) gives, after multiplication of the first block row with Y_k:

$$\left[\begin{array}{c} Y_k x_{k+1} \\ \hline M_k x_k \\ N_k u_k \end{array}\right] = \left[\begin{array}{c} Y_k A_k \\ \hline M_k \\ 0 \end{array}\right] x_k + \left[\begin{array}{c} Y_k B_k \\ \hline 0 \\ N_k \end{array}\right] u_k = \left[\begin{array}{c|c} Y_k A_k & Y_k B_k \\ \hline M_k & 0 \\ 0 & N_k \end{array}\right] \left[\begin{array}{c} x_k \\ u_k \end{array}\right] \tag{2.5}$$

and the optimization problem specializes to: *find the u_k that minimizes the cost $x_k' M_k' M_k x_k + u_k' N_k' N_k u_k + x_{k+1}' Y_k' Y_k' x_{k+1}$ (x_k is thought given and fixed in this phase of the recursion and its cost obviously cancels).* Notice: in this phase of the recursion x_k is the only remaining 'variable', the others, namely u_k and x_{k+1} will be 'optimized out' in function of x_k.

Eq. 2.5 is a linear (vector) equation of the form
<cost> = <known data> + <matrix T> times <driver> and the goal is to find a <driver> that minimizes the <cost> in a quadratic sense. The general

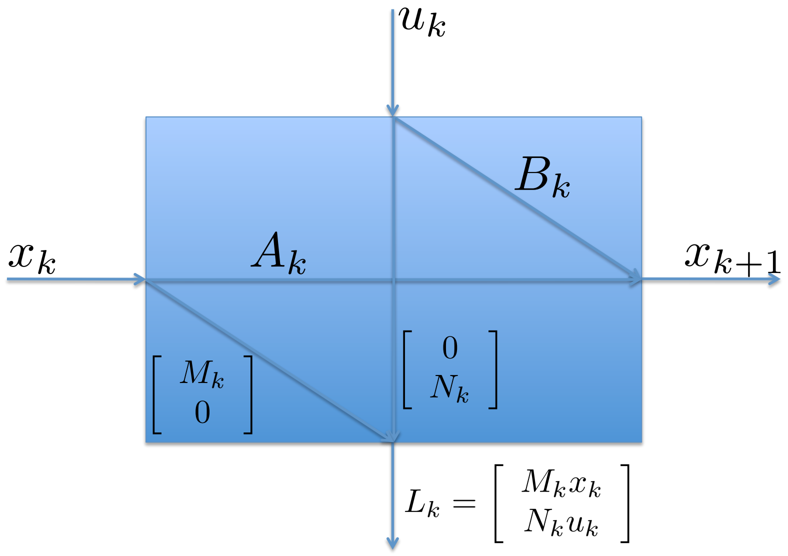

u_k
B_k
x_k
A_k
x_{k+1}
$\begin{bmatrix} M_k \\ 0 \end{bmatrix}$
$\begin{bmatrix} 0 \\ N_k \end{bmatrix}$
$L_k = \begin{bmatrix} M_k x_k \\ N_k u_k \end{bmatrix}$

Figure 2.3: The full local cost model for optimal quadratic control, including cost outputs

solution of this problem is given by '<cost> = -$T^\dagger$ <known data>', in which $T^\dagger$ is the *Moore-Penrose pseudo-inverse* of T (see the preliminary section!). Here we consider the common case in which T has independent columns, in which case the solution is unique and the Moore-Penrose pseudo-inverse is given by $T^\dagger = (T'T)^{-1}T'$. This is a good closed form expression, but our filtering strategy, to be developed next, will produce an attractive, recursive solution of low numerical complexity (for the definition of the Moore-Penrose inverse, see the first section of this chapter or the mathematical inroduction at the end of the book).

One effective way to find this pseudo-inverse and hence solve the problem is to execute a so-called *QL-factorization* on the full 'system model' given by the third term in eq. 2.5. 'Q' stands for an orthogonal transformation and 'L' for a lower-triangular matrix whose rows are linearly independent and have a 'staircase' aspect (often with irregular steps, but not so in the present case, because our assumptions will be such that the matrix to be factorized is assumed to have independent columns). The computation of an orthonormal basis for the columns (range) of a given matrix, is a classical problem in numerical analysis and can be done with a variety of methods that we shall discuss later (Givens or Jacobi rotations, SVD or even n-dimensional global rotations)—see, for a start, the section on implementation further on in this chapter. The result of the QL-factorization is summarized in the following formula, in which we have introduced appropriate (to be explained) symbols for all the elements that are computed in the operation, namely all the matrices and submatrices on the right hand side:

$$\left[\begin{array}{c|c} Y_kA_k & Y_kB_k \\ \hline M_k & 0 \\ 0 & N_k \end{array}\right] = Q_k \left[\begin{array}{c|c} 0 & 0 \\ \hline Y_{k-1} & 0 \\ C_{o,k} & D_{o,k} \end{array}\right] \tag{2.6}$$

Q_k is an orthogonal (or, in the complex case, unitary) matrix that produces the block staircase in the right factor, namely, as we shall show in the next paragraph, the new (recursive) value for Y_{k-1} as well as the non-singular bottom row $\left[\begin{array}{cc} C_{o,k} & D_{o,k} \end{array}\right]$ that contains the data needed for the optimal control, as we shall see soon. The computation of new matrices from a given matrix, as is done in eq. 2.6, has been termed *array processing* in the literature, for obvious reasons.

To proceed with ease, we need some further properties of Y_{k-1} and $D_{o,k}$. They will be square non-singular matrices when the left hand side of eq. 2.6 consists of independent columns. At this point we just assume that this is the case (the condition is necessary for a unique solution to exist).

From eq. 2.5, we see that the optimal u_k is given by

$$u_{\min,k} = -\left[\begin{array}{c} Y_kB_k \\ \hline 0 \\ N_k \end{array}\right]^{\dagger} \left[\begin{array}{c} Y_kA_k \\ \hline M_k \\ 0 \end{array}\right] x_k \tag{2.7}$$

in which $\left[\begin{array}{c} Y_kB_k \\ \hline 0 \\ N_k \end{array}\right]^{\dagger}$ is the Moore-Penrose pseudo-inverse of the coefficient matrix of u_k and, by eq. 2.6, we may infer:

$$u_{\min,k} = -D_{o,k}^{-1} C_{o,k} x_k \tag{2.8}$$

because

$$\left[\begin{array}{c} Y_kB_k \\ \hline 0 \\ N_k \end{array}\right] = Q_k \left[\begin{array}{c} 0 \\ 0 \\ D_{o,k} \end{array}\right] \text{ and } \left[\begin{array}{c} Y_kA_k \\ \hline M_k \\ 0 \end{array}\right] = Q_k \left[\begin{array}{c} 0 \\ Y_{k-1} \\ C_{o,k} \end{array}\right]. \tag{2.9}$$

Hence $\begin{bmatrix} Y_k B_k \\ \hline 0 \\ N_k \end{bmatrix}^\dagger = \begin{bmatrix} 0 & 0 & D_{o,k}^{-1} \end{bmatrix} Q_k'$. The orthogonality $Q_k' Q_k = I$ then produces the result. Next, filling in the optimal control in the cost equation immediately produces the minimal cost $c_{\min,k} := Q_k \begin{bmatrix} 0 \\ Y_{k-1} \\ 0 \end{bmatrix} x_k$ and hence $c_{\min,k}' c_{\min,k} = x_k' Y_{k-1}' Y_{k-1} x_k$ as claimed by the recursive hypothesis.

A simple QL-factorization gives the complete recursive solution of the quadratic tracking problem, whereby the orthogonal Q-factor filters the input data to produce both the optimal control and the new global cost function, all in function of the actual state x_k. This local factorization leads to the product decomposition of the model filter shown in fig. 12.3 known as an *inner-outer factorization*—a topic that we shall discuss in great detail in a later chapter.

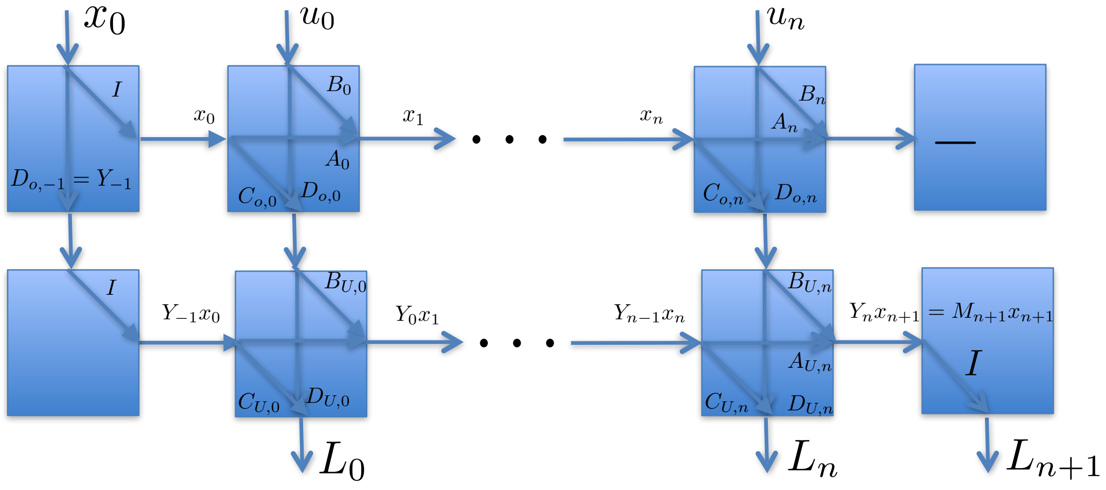


Figure 2.4: The recursive factorization of the Bellman model filter in an inner (bottom chain) and an outer (top chain) filter. The picture shows the signal propagation in the factorization as well.

In the optimal case, the input to the inner filter (see fig. 2.4) is seen to be zero (from the control equation, eq. 2.8), except for the initial input x_0, and the inner filter shows the distribution of the minimal cost across the system (fig. 2.5). The inverse of the outer filter with input zero gives the control

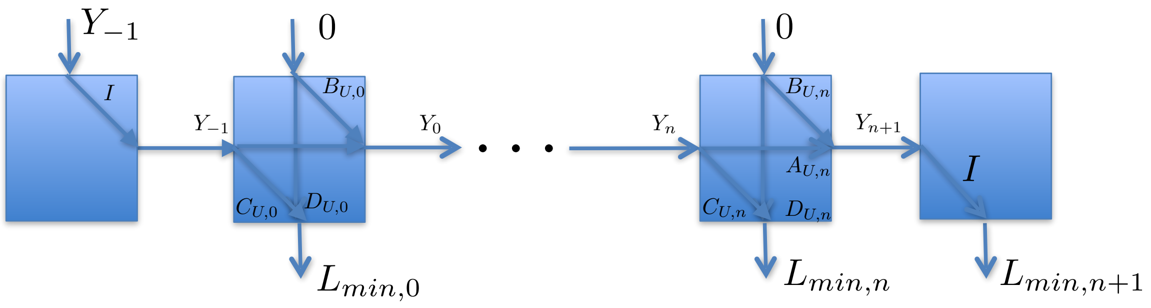


Figure 2.5: The signal propagation in the inner filter in the minimal case.

for the optimal case with the adopted costs, shown in fig. 2.6. However, that is not all. Both the inner filter and the outer filter are invertible. The inverse of the inner filter is anti-causal, while the inverse of the outer filter is causal. The inverses allow one to modify the cost at specific locations and

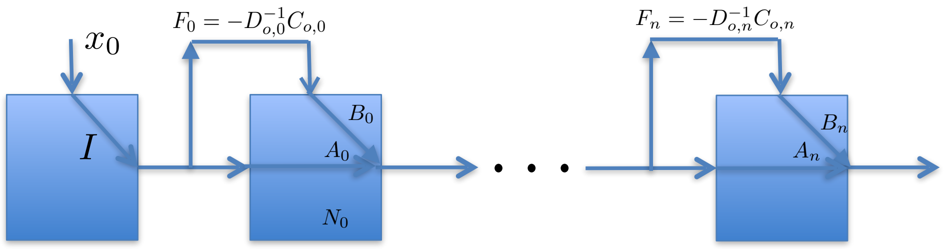


Figure 2.6: The optimal controller is simply the inverse of the outer filter.

then compute the input needed to achieve the new performance (this design issue leads beyond the scope of the present discussion.).

2.3 A toy example: row, row, row your boat*

A starred section may be skipped without impairing the continuity of the development.

Suppose you want to cross a river with a rowing boat. The current in the river has variable velocities depending on the distance from shore. You can let your boat drift, and with careful handling of the rudder or the oars, you can reach the other side without any effort on your part. This will drift you off too much, so, instead, you shall row against the current with the aim to reach a destination close to your starting point, at the other side. You will try to do a best possible job, by minimizing the effort you have to exert, while trying to get close to your destination. We make a simplified model of the situation, for discussion purposes. Here are the assumptions (see fig. 2.7):
- we subdivide the river in four segments $0:3$, each segment having a uniform speed of water $v_{0:3}$. We let the current flow in the (vertical) x-direction
- the 'natural drift' in each segment (i.e., the drift with no effort) is given by δ_i, $i \in 0:3$. E.g., δ_i proportional to v_i with some constant;
- rowing provides for an improvement on the drift of $d_i > 0$ in segment i, whereby the rowing effort is pegged at $n_i^2 d_i^2$ for some n_i solely dependent on v_i (perhaps proportional, assuming the force to be overcome proportional to the speed of the water that hits the boat). This is motivated by the following, admittedly somewhat flimsy, consideration: the force to gain d_i against the current speed v_i is proportional to d_i. So is also the displacement you have to cover with this force. So, the total energy your rowing has to produce is proportional to d_i^2 with some constant n_i^2 (other considerations would lead to a non-quadratic cost, which would complicate matters considerably mathematically, without impairing the principle of dynamic programming);
- the total cost to be optimized becomes hence

$$C_4 = \sum_{i=0}^{3} n_i^2 d_i^2 + m^2 x_4^2, \tag{2.10}$$

in which the offset at destination x_4 is penalized as $m^2 x_4^2$ for some m to be chosen. All the 'modeling quantities' n_i and m are assumed known (this is the big physical work!).

The local propagation model is very simple in this case:

$$x_{k+1} = x_k + \delta_k - d_k. \tag{2.11}$$

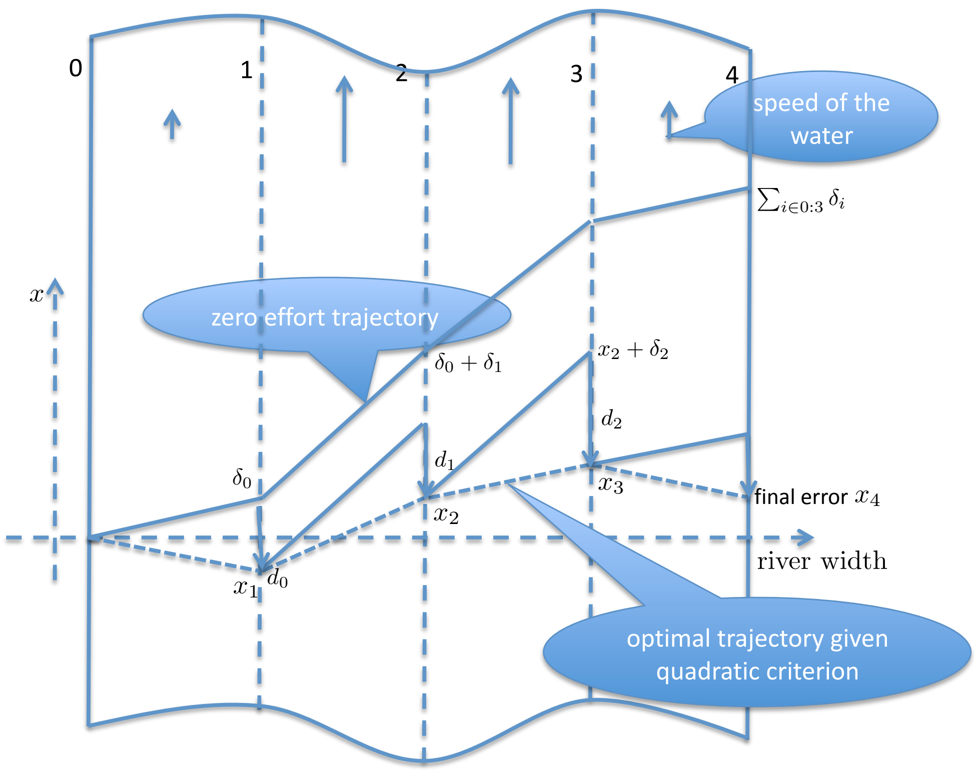


Figure 2.7: Optimal cost trajectory to row over a river with variable water speed.

Notice that the model is not linear: it is *affine* because of the drift term, but we shall soon see that it can be handled with linear methods just as well (thereby generalizing the development in the previous section!).

Our optimization strategy now consists in writing down the *cost model*, with the terms appearing squared in the cost function as outputs. We have, for the global cost model:

$$\left[\begin{array}{c} y_{0:3} \\ \hline y_4 \end{array}\right] = \left[\begin{array}{ccc} n_0 & & \\ & \ddots & \\ & & n_3 \\ \hline -m & \cdots & -m \end{array}\right] \begin{bmatrix} d_0 \\ \vdots \\ d_3 \end{bmatrix} + \left[\begin{array}{c} 0 \\ \vdots \\ 0 \\ \hline m\delta_t \end{array}\right] \tag{2.12}$$

in which $\delta_t = \sum_{i=0}^{3} \delta_i$ is the total drift and the $\|y_i\|^2$ are the local contributions to the overall cost. Let us define $N := \text{diag}[n_i]$ and $E = \text{col}\left[\begin{array}{cc} 1 & \cdots 1 \end{array}\right]$ a column vector of 1's, then the Moore-Penrose inverse of the non-singular *system matrix* $S := \left[\begin{array}{c} N \\ \hline -mE' \end{array}\right]$ is

$$S^{\dagger} = (N^2 + m^2 EE')^{-1} \left[\begin{array}{cc} N & -mE \end{array}\right] \tag{2.13}$$

and the solution of the optimization problem is given by

$$\widehat{d}_{0:3} := (N^2 + m^2 EE')^{-1} m^2 E \delta_t \tag{2.14}$$

This expression can be computed explicitly, using the inversion rule for a low rank perturbation of a non-singular matrix (sometimes called the 'Sherman-Morrison formula': suppose that low dimensional (rectangular) matrices A and B of same dimensions are such that $I + B'A$ is non-singular, then $(I + AB')^{-1} = I - A(I + B'A)^{-1}B'$; proof is by direct verification; the simplest possible case is when A and B are just vectors.). We leave details to the interested reader. The result is

$$\widehat{d}_i = \left(\frac{m^2 \frac{1}{n_i^2}}{1 + m^2 \sum(\frac{1}{n_i^2})} \right) \delta_t \tag{2.15}$$

which is a forced, a priori control (not a state dependent control), forced by the parameters n_i and the total drift δ_t. Notice that $\widehat{d}_i = \frac{a_i^2}{\sum a_i^2} \delta_t$ with $a_i := \frac{1}{n_i}$ when $m \to \infty$, so the effort to be spent at each step becomes inversely proportional to the local speed squared, assuming n_i proportional to the local speed (which is not unreasonable altogether: you distribute the energy to be exerted evenly over the sections. Notice also that in this limiting case $\sum \widehat{d}_i = \delta_t$, forcing the rower to get at the destination point exactly.).

The global character may be seen as a problem with the global solution. Many things can happen when one is underway, and it pays to figure out recursive solutions that can adapt to the perspective from a local state one may have reached. The global solution can of course easily be converted to a local solution, just by adapting the parameters to the state reached (making them a function of that local state). But there is another advantage to the local solution (given the global validity of the model of course), and that is that at any local position only information on the next move is needed.

In other words: at each local position, control is only dependent on that position. This fact is what makes dynamic programming so efficient.

Let us therefore follow how the recursive optimization happens in this case and how the control law depending on the local state is derived. Using subsequent Givens rotations of the type $\begin{bmatrix} c & -s \\ s & c \end{bmatrix}$, we have for the first reduction to lower triangular, starting on the right bottom of S

$$\begin{bmatrix} c_3 & -s_3 \\ s_3 & c_3 \end{bmatrix} \begin{bmatrix} n_3 \\ -m \end{bmatrix} = \begin{bmatrix} 0 \\ -\sqrt{n_3^2+m^2} \end{bmatrix} \tag{2.16}$$

for $c_3 = m/\sqrt{n_3^2+m^2}$ and $s_3 = -n_3/\sqrt{n_3^2+m^2}$. This first step reduces S to an intermediate

$$\left[\begin{array}{ccc|c} n_0 & & & \\ & n_1 & & \\ & & n_2 & \\ Y_2 & Y_2 & Y_2 & 0 \\ \hline C_{o3} & C_{o3} & C_{o3} & D_{o3} \end{array}\right] \tag{2.17}$$

in which $D_{o3} := -\sqrt{n_3^2+m^2}$, $Y_2 := -s_3 m$ and $C_{o3} := -c_3 m$. The recursion now moves one step up the diagonal, and applying a new Givens rotation to the new right bottom:

$$\begin{bmatrix} c_2 & -s_2 \\ s_2 & c_2 \end{bmatrix} \begin{bmatrix} n_2 \\ Y_2 \end{bmatrix} = \begin{bmatrix} 0 \\ \sqrt{n_2^2+Y_2^2} \end{bmatrix} \tag{2.18}$$

with $c_2 := Y_2/\sqrt{n_2^2+Y_2^2}$ and $s_2 := n_2\ \sqrt{n_2^2+Y_2^2}$, producing the next step in the reduction of S:

$$\left[\begin{array}{cc|cc} n_0 & & & \\ & n_1 & & \\ Y_1 & Y_1 & & \\ \hline C_{o2} & C_{o2} & D_{o2} & \\ C_{o3} & C_{o3} & D_{o3} & D_{o3} \end{array}\right] \tag{2.19}$$

in which $D_{o2} := \sqrt{n_2^2+Y_2^2}$, $Y_1 := -s_2 Y_2$ and $C_{o2} := c_2 Y_2$. Continuing recursively one finally gets

$$S = Q \left[\begin{array}{cccc} 0 & 0 & 0 & 0 \\ \hline D_{o0} & & & \\ C_{o1} & D_{o1} & & \\ C_{o2} & C_{o2} & D_{o2} & \\ C_{o3} & C_{o3} & D_{o3} & D_{o3} \end{array}\right] \tag{2.20}$$

and Q is a product of four elementary orthogonal rotations adequately embedded in a 5×5 orthogonal transformation $Q = Q_3 Q_2 Q_1 Q_0$, in which, e.g.,

$$Q_1 = \begin{bmatrix} 1 & & & & \\ & c_1 & s_1 & & \\ & -s_1 & c_1 & & \\ & & & 1 & \\ & & & & 1 \end{bmatrix}. \tag{2.21}$$

It is somewhat awkward to express the global Q in matrix form, because the matrix form hides the simple structure of the product. A more effective way is to exhibit the computational structure explicitly. We discuss this in the next paragraphs. Important is to see that both the recursive Q and R factors have an intrinsic low complexity, due to the recursive, dynamic system or evolutionary structure of the original problem. To exhibit and utilize this structure will be a main issue in the system theory that we shall develop in this book.

So far for the global computation. The computation of the total cost quickly becomes unwieldy, even in this extremely simple case! However, the local, recursive computation, based on dynamic programming, remains simple and is worth pursuing. In the present case, we can even greatly simplify the local computation by some further modelling considerations (which turn out to be very worthwhile, the whole art of the game is good modelling!). Suppose you have arrived at stage k of the computation, and you are regressing backwards from stage $k+1$ to stage k using dynamic programming. Considering the optimization at stage k, what you need is
1. the optimal cost $\widehat{C}(x_{k+1})$ for each (potential) x_{k+1}, and
2. the *cost model* at stage k, which consists of the local cost for the transit from x_k to x_{k+1}, assuming x_k given, as well as the optimal cost for all the x_{k+1} you might reach in the transit from k to $k+1$.
Concerning the optimal cost $\widehat{C}(x_{k+1})$, we observe 1. that the cost is zero if $x_{k+1} = -\delta_{t,k+1}$, where $\delta_{t,k+1} = \sum_{i=k+1}^{n} \delta_i$, because in that case we reach the ideal destination with zero effort, and 2. the optimal cost expression in function of x_{k+1} will very likely be a quadratic expression (we shall have to prove this hypothesis recursively), which then necessarily would have the form $m_{k+1}^2 (x_{k+1} + \delta_{t,k+1})^2$, with m_{k+1} some coefficient to be determined recursively. This local model at stage k is shown in fig. 2.8.

This cost model is affine in the steering (input) vector d_k, and the cost

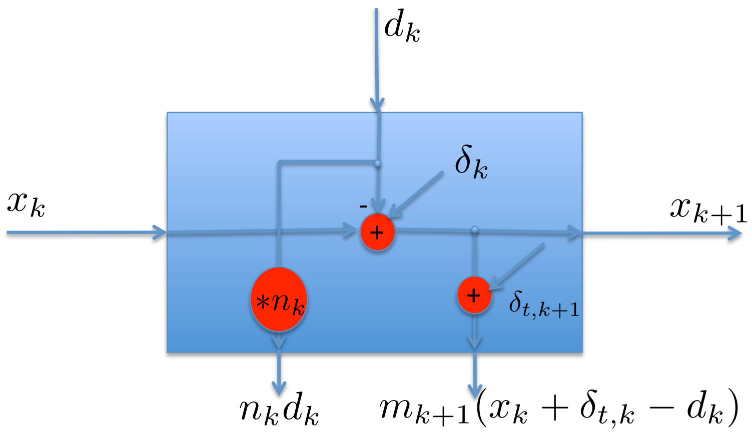


Figure 2.8: The local cost model for our rowing situation: it is the quadratic norm of the output vector.

vector (cost being its norm squared) is then

$$\begin{bmatrix} m_{k+1}(x_{k+1} + \delta_{t,k+1}) \\ n_k d_k \end{bmatrix} = \begin{bmatrix} -m_{k+1} \\ n_k \end{bmatrix} d_k + \begin{bmatrix} m_{k+1}(x_k + \delta_{t,k}) \\ 0 \end{bmatrix} \tag{2.22}$$

By the Moore-Penrose theory, it follows that

$$\widehat{d}_k = - \begin{bmatrix} -m_{k+1} \\ n_k \end{bmatrix}^{\dagger} \begin{bmatrix} m_{k+1}(x_k + \delta_{t,k}) \\ 0 \end{bmatrix} = \frac{m_{k+1}^2}{n_k^2 + m_{k+1}^2}(x_k + \delta_{t,k}). \tag{2.23}$$

This is an affine control, partly proportional to x_k with a constant $\frac{m_{k+1}^2}{n_k^2+m_{k+1}^2}$ *plus* an a priori driving term). For the cost we find, after a small calculation,

$$\widehat{C}(x_k) = \frac{n_k^2 m_{k+1}^2}{n_k^2 + m_{k+1}^2}(x_k + \delta_{t,k})^2 \tag{2.24}$$

thereby proving the recursive hypothesis, with $m_k = \frac{n_k m_{k+1}}{\sqrt{n_k^2+m_{k+1}^2}}$. That's all!

Remarks

- The simplicity of the recursive solution should be obvious, but it requires some additional modeling effort to get it. It also has a great advantage, not mentioned so far, and that is that the optimization criteria may be modified adaptively as one proceeds (e.g., when one gets better estimates of future costs). This is an issue we shall not address here, but which may come up when we discuss optimization problems in later chapters.

- In case one has to reach the destination exactly ($x_{n+1} = 0$), then one may let m tend to infinity with some care. This will not change much in the derivation, except at the last stretch. It is a good exercise to do!

- From the control formula, it is clear that the effort to be performed at any stage k has n_k^2 in the denominator, meaning that the stronger the current the less one should row against it, and this considerably! Going against the current is of course not restricted to the rowing case, and the wisdom to profit from the least resistance or the low hanging fruit is clear. The general physical principle of least action has some of this flavor as well.

- It should be clear that the Moore-Penrose optimization method only works on linear or affine models and costs of quadratic type. When the cost function or the model is more complex, then the whole procedure becomes considerably more complex as well, but the principle of dynamic programming often still holds, and recursive computations remain a method of choice. It is instructive to analize when the principle breaks down (we leave this point as a thinking exercise!)

- Our example was restricted to $n = 4$, but the treatment was perfectly general (provided the model is valid of course), and could also be utilized for a continuous time situation after discretization, or, conversely, to derive the continuous time treatment from the discretized (which is what is often done in the literature).

- It is easy to see that the recursive solution is the same as the global one. A good exercize!

- It is not easy to find a simple direct example for a purely quadratic optimization problem on a linear model. Most such problems are of the type ‘tracking a non-linear trajectory to counteract stochastic disturbances’. We shall discuss such problems in a later chapter.

2.4 Implementation

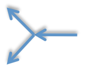
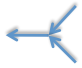
The filtering algorithm we have been discussing so far boils down to the determination of an orthonormal basis for a recursively computed set of vectors, for which an algorithm like QL or RQ (same as QL but done on the transpose) is appropriate. The most elementary (but singularly effective) way of doing this is by systematically using Jacobi/Givens transformations. Here we consider only this method, leaving other methods for later chapters, and this to illustrate the technique we shall use to represent computations.

The data model

To describe computational or time-variant systems comfortably, we need a data model that is more flexible than the traditional signal flow diagrams, because the computing nodes we use may have more than one function and we wish to have a representation that is both close to functional algebra and computer processing. For this we propose a simple *functional representation*, defined as a directed graph consisting of nodes and edges with the following properties (semantics):
- nodes execute functions and data is transmitted along edges;
- a node has input ports (indicated by incoming arrows) and output ports (indicated by outgoing arrows);
- a node executes a sequence of computing cycles; at the beginning of a computing cycle of a given node, a specific function is installed by the node controller, the function reads (and consumes) the input data it needs from its relevant input ports, does the computation, puts (pushes) the output data on the relevant output ports (these ports may differ per function) and installs the next function;
- input and output data accumulate on the edges in a first-in, first-out (FIFO) fashion; any function installed in a node waits until its relevant input ports have obtained the data the function needs, and pushes the output data it produces on its relevant output ports.

This means, in particular, that the edges must be equipped with the necessary memory and that the execution proceeds in a Petri-net fashion. Each node executes a *trace* of functions in the course of its history[1]. We may draw

[1]This model is close, but not identical, to the Kahn network formalism. Rather, it adheres closely to the algebraic practice of composing functions and arguments. It also extends the signal flow diagrams in a straightforward way.

functional diagrams from right to left, in accordance with mathematical practice (example: in $y = Au$, u is fed in from the right and y gets out at the left side, and we draw $y \leftarrow A \leftarrow u$.). A split node  and a join  typically act as controlled switches (in the diagrams presented here, there are other possibilities of course, but our conventions respect global losslessness or unitarity, when the individual nodes are unitary).

Simplified convention

When there is no confusion possible, we retreat to the classical signal flow graph convention, where an edge is seen as a multiplication by a coefficient or a matrix marked on top of it, like in $\xleftarrow{A}$, which according to the functional data model should be denoted '$\leftarrow A \leftarrow$', since multiplication by A is an operation, to be executed, and a join is actually a node in which matrix addition is performed. (In our datamodel, the trace representing the sequence of data transmitted is often put on top of the arrow. The applicable situation is to be derived from the context!)

Realizations using Jacobi/Givens rotors

The simplest possible components for orthogonal transformations (often called *orthogonal filtering*) are elementary real or complex *rotors* (the elementary rotor goes back to Jacobi, but in numerical algebra is known as a 'Givens rotation'). Only the real case is considered here, but the rotors can easily be extended to the complex case.

Rotors

A rotor comes in two (switchable) versions: the *vectoring* mode, where a rotation matrix is computed from the input data and applied, and the *rotating* mode, where a given rotation matrix is applied on new data—see fig. 2.9. In the *vectoring mode (v)* of the single rotor, a vector $\begin{bmatrix} b & a \end{bmatrix}$ is inputed, the rotor calculates $n = \sqrt{a^2 + b^2}$, $c = a/n$ and $s = b/n$, and outputs a 'control' output $\begin{bmatrix} s & c \end{bmatrix}$ and a 'data' output $\begin{bmatrix} 0 & n \end{bmatrix}$ so that

$$\begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{bmatrix} b \\ a \end{bmatrix} = \begin{bmatrix} 0 \\ n \end{bmatrix}. \tag{2.25}$$

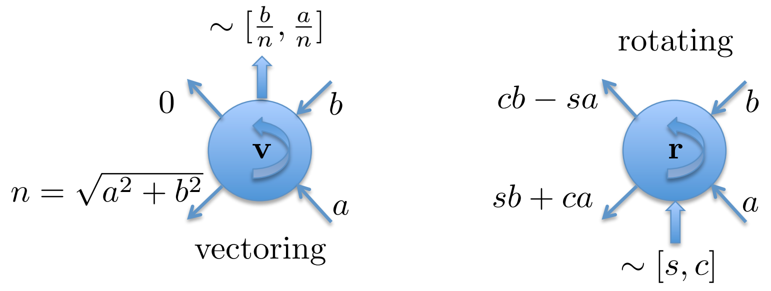


Figure 2.9: The single Jacobi/Givens rotor in two modes: vectoring and rotating. In the vectoring mode, angle information is produced, labeled here as $[s, c]$, while in the rotating mode the angle information is used to perform the rotation. In many cases the angle information remains resident in the node and is not shown. Signal propagation is from SE to NW.

In the *rotating mode (r)*, a control input $\begin{bmatrix} s & c \end{bmatrix}$ is given and applied to an input vector $\begin{bmatrix} b & a \end{bmatrix}$ to produce an output vector $\begin{bmatrix} d & c \end{bmatrix}$ so that

$$\begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{bmatrix} b \\ a \end{bmatrix} = \begin{bmatrix} d \\ c \end{bmatrix}. \tag{2.26}$$

Figure 2.10 shows a simple application in which the vectoring mode is used recursively to produce the norm of a vector (in this example, the recursive edge is initialized to 0 and the trace is vvv $\cdots$). There is an electronic device called CORDIC that implements the rotor using elementary (bitwise) rotations and that is used in high frequency applications[2]. The Jacobi/Givens matrix is in principle unitary, but in finite arithmetic, the unitarity cannot be exact. A conservative solution to this problem insures that the actual matrix is slightly contractive (as also used in standard wave digital filtering), so that no extra energy that could lead to instabilities is generated.

In the typical inner-outer applications, the initial data at the beginning of a step is given as $\begin{bmatrix} AY & B \\ CY & D \end{bmatrix}$ (step index k dropped for convenience) which the orthogonal filter with transfer matrix Q' transforms to $\begin{bmatrix} 0 & 0 \\ Y_n & 0 \\ C_o & D_o \end{bmatrix}$ starting from the last column and moving forward to the first, compressing rows towards the bottom (Y_n is the 'Y' for the next recursion!). As one needs

[2]See e.g., www.actel.com/ipdocs/CoreCORDIC_HB.pdf.

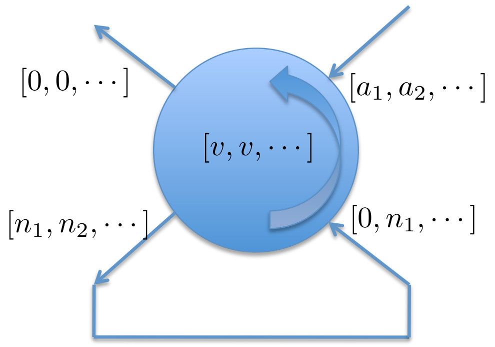


Figure 2.10: The single rotor used as a norm generating filter (traces shown).

to input the last column first, one can best use the *involution constructor*, denoted 'i' on the columns to represent this reversal (the involution 'i' can also be thought as a generic matrix with '1's on the anti-diagonal, which reverses the order of the columns when applied to the right of a matrix, or rows when applied to the left) to compute

$$\begin{bmatrix} 0 & 0 \\ Y_n & 0 \\ C_o & D_o \end{bmatrix} i \leftarrow Q' \leftarrow \begin{bmatrix} AY & B \\ CY & D \end{bmatrix} i. \qquad (2.27)$$

The block operation can be refined further by first acting with an orthogonal transformation Q_1' on $\begin{bmatrix} B \\ D \end{bmatrix} i$ to produce $\begin{bmatrix} 0 \\ D_o \end{bmatrix} i$, next using the recently computed Q_1' on $\begin{bmatrix} AY \\ CY \end{bmatrix} i$ to produce an intermediate result $\begin{bmatrix} X_1 \\ X_2 \end{bmatrix} i$, whereupon a (reduced) orthogonal transformation $Q_2' := \begin{bmatrix} \widehat{Q}_2' & \\ & I \end{bmatrix}$ produces the final result:

$$\begin{bmatrix} 0 & 0 \\ 0 & Y_n i \\ D_o i & C_o i \end{bmatrix} \leftarrow \begin{bmatrix} \widehat{Q}_2' & \\ & I \end{bmatrix} \longleftarrow Q_1' \leftarrow \begin{bmatrix} Bi & AYi \\ Di & CVi \end{bmatrix}. \qquad (2.28)$$

More details of the operations go all the way down to individual rotors that constitute the product $Q_2'Q_1'$ shown for the 3×2 case with D scalar, A of dimension 2×2 and Y of dimension 2×1 in fig. 2.11 (in this example $A = \begin{bmatrix} 1 & -2 \\ 2 & 1 \end{bmatrix}$, $B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, $C = \begin{bmatrix} 2 & -1 \end{bmatrix}$, $D = 1$ and $Y = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ gives

$D_o = \sqrt{2}$, $C_o = 2\sqrt{2}$ and the new $Y_n = \sqrt{3}$). In this typical orthogonal filter

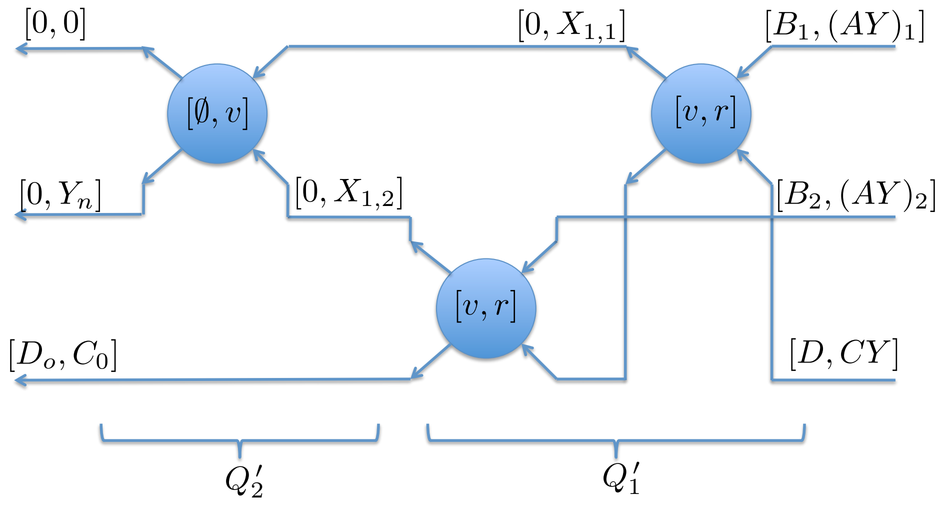


Figure 2.11: A 3×2 example of a step in an inner-outer factorization. In this example, D is assumed scalar, A a 2×2 matrix and Y a 2×1 vector. The traces on the rotors and edges are shown ($\emptyset$ stands for 'no operation'). Rotation angles remain internal to the rotors and are not shown, the last one computed in a vectoring operation is used in the next rotation.

configuration, the information on the angles remains resident in the rotor nodes and the outer factor is outputed (C_o and D_o), together with the data needed for the next recursion (Y_n). A similar filter can then be used in the next iteration after inputing the new data (dimensions may and often will change!). Some further observations:
- The Q matrix in this case has dimensions 3×3 with 8 non-zero entries, but is characterized by only three rotors. Even applying those elementary rotations in matrix form would be economical rather than multiplying with the matrix (and more so in higher dimensions). In the present example we have

$$Q = \left[\begin{array}{c|c|c} B_W & A_U & B_U \\ \hline D_W & C_U & D_U \end{array}\right] = \left[\begin{array}{c|c|c} \sqrt{2/3} & -\sqrt{1/3} & 0 \\ \sqrt{1/6} & \sqrt{1/3} & \sqrt{1/2} \\ \hline -\sqrt{1/6} & -\sqrt{1/3} & \sqrt{1/2} \end{array}\right] \tag{2.29}$$

(remark that in this case A_U is not square, because the dimensions of the

new 'Y' are different from the previous.)
- The 3×2 case is perhaps the simplest relevant case. In many cases the D matrix is (essentially) not scalar and the connecting matrix Y square and non-singular. In some important cases the Y matrix has to be rectangular, but it will always be chosen non-singular.

2.5 A question to be researched

Thinking about optimizing the behavior of a system evolving in time, under what conditions would dynamic programming be possible? Or, negatively, when is dynamic programming for sure not possible?

2.6 Notes

1. Dynamic programming is only one example of the use of system theory. There are many more examples, many of which will be discussed in this and the following books. We mention: state estimation (Kalman filtering), control theory, system modeling and model reduction, approximation and interpolation of matrices, efficient computations with some types of structured matrices, data filter design a.s.o. The value of dynamic programming as a prime example is the intimate connection it exhibits between the dynamic model and the recursion that leads to the optimal control. Nice further examples of interesting optimization problems on systems can be found in the book of Luenberger [27]. A classical textbook on linear optimization is [1]. Nonetheless, the straight connection between quadratic tracking and inner-outer factorization has only recently been made.

2. There is a host of other methods one can use to orthogonalize a set of commensurable vectors, to wit:
   - global multi-dimensional rotations (to be preferred) or Householder transformations;
   - Gram-Schmidt orthogonalization;
   - bi-orthogonalization using hyperbolic transformations [9, 8].

   These methods will be discussed in further chapters, when they come up, and in the appendix on Linear Algebra.

3. The functional representation and data model described in this chapter was first proposed and analyzed in [3]. It allows for an unbiased transformation of mathematical operations to a computer architecture at the functional level. It hides organizational details like the conditional sequencing of functions, the partitioning, storage and transfer of data, in such a way that this information can easily be generated once further architectural decisions like the localization of data in memories and the assignment of functions to processors has been done (such design phases will not concern us in this book). In particular, the model easily accommodates hierarchical representations and parallel processing. In this book we shall only worry about practical implementation aspects for as far as numerical properties (numerical accuracy and complexity) are concerned.

Chapter 3

Dynamical Systems: a narrative

This chapter brings a semi-philosophical description of the notions 'system' and 'dynamical system', together with an introduction to the main concepts characterizing the latter: state, behavior, reachability and observability, in non-technical terms—it brings what people call a 'narrative' that aims at describing the background ideas of system theory. In the remainder of the book the ideas presented are taken up again in more concrete situations, showing that not much more is needed conceptually to solve most system related problems, provided one succeeds in characterizing these notions mathematically. The chapter can be skipped at first reading, but it also introduces a way of thinking that motivates the general modern approach to dynamical system theory and can thus be seen as an introductory narrative as well.

Reachability and Observability

Third course
Behaviour

Dessert
Discussion Topics

3.1 What is a (dynamical) system?

The Oxford dictionary says:

system
noun
1 a set of things working together as parts of a mechanism or an interconnecting network; a complex whole: the state railway system || fluid is pushed through a system of pipes or channels.
· Physiology a set of organs in the body with a common structure or function: the digestive system.
· the human or animal body as a whole: you need to get the cholesterol out of your system.
· Computing a group of related hardware units or programs or both, especially when dedicated to a single application.
· Geology (in chronostratigraphy) a major range of strata that corresponds to a period in time, subdivided into series. the Devonian system.
· Astronomy a group of celestial objects connected by their mutual attractive forces, especially moving in orbits about a centre: the system of bright stars known as the Gould Belt.
· short for crystal system.
2 a set of principles or procedures according to which something is done; an organized scheme or method: a multiparty system of government — the public-school system.
· a set of rules used in measurement or classification: the metric system.
· [mass noun] organized planning or behaviour; orderliness: there was no system at all in the company.
· a method of choosing one's procedure in gambling.

In short: a *system* is an "assembly of interconnected and interacting entities which together achieve a specific behaviour". Hence, it consists of (1) entities, (2) interconnections, (3) interactions and (4) an identifiable behaviour. Such a definition is very general and as a consequence there are many types of systems. A discipline called *System Theory* would then attempt to find some communality between all these types, or at least discover classes of systems that can be described meaningfully. As it would be impossible to have descriptions that fit all, engineers and scientists who want to understand how a system functions or how to build one, wisely restrict themselves to classes on which they have some grip, i.e., types of systems they can either manufacture, describe mathematically or both. The combination of mathematical description and ability to manufacture is what allows engineers to design a system. However that may be, systems do occur naturally and there are many types of extremely interesting systems that we cannot manufacture — yet can use to combine them with man-made subsystems. Some can be described mathematically pretty well, think e.g., of the sun-and-planets system, others are so complex or complicated that they defy unified mathematical descriptions, think e.g., about a cellular organism or the human brain. *Identifying* such a system from its behaviour is then a major issue.

As soon as one observes a system, one discovers that it moves and evolves. So, often one shall be interested not only in how a system is put together, but also in its evolution. When this evolution is the prime interest, one would call it a *dynamical system* in contrast to a purely static system, e.g., the architecture of a building. Our main interest in this book shall go to dynamical systems. The very first dynamical system ever described mathematically is the system of planets evolving around our sun. Isaac Newton deserves the tribute that he discovered the basic dynamics of this system: the sun and each planet is given one static variable, their mass, and six characterizing dynamic variables, three positions and three velocities, while the future evolution is given by the law of gravity, that defines the derivatives of these 'dynamic' variables (Newton discovered the notion of 'derivative' exactly for the purpose). All the variables that characterize the dynamical entities of the system are called its *state variables*, and the evolution is then given by writing out equations that describe how these variables change with time, e.g., an ordinary differential equation relating the derivatives of the state variables to their actual values and the forces of gravity to which they are subjected.

3.2 Discrete-time systems

In the present age of computers controlled by a clock, we often encounter situations where there is discrete time evolution: the 'time' is actually an index, and system evolution then moves from some time index — say k — to the next time index $k+1$ producing the dynamic state equations

$$\left\{ \begin{array}{lcl} x(k+1) & = & f_k(x(k), u(k)) \\ y(k) & = & g_k(x(k), u(k)) \end{array} \right. . \tag{3.1}$$

In this expression, we have chosen to index the functions f and g, a convention that is often used. We could have written $f(x(k), u(k), k)$, the important and highly limiting restriction being the direct dependence on states and inputs at index k.

A discrete time system will also result from the discretization of a continuous time system. This appears to be a necessary condition for numerical simulation of physical systems. The discretization of physical systems with continuous time often results in an accurate description when carefully done, so that we can concentrate in first instance on discrete time systems to obtain results that are also valid for the continuous time case. It turns out that most basic notions of dynamical system theory can indeed be covered in this way.

A further discussion leads us to a number of central notions characteristic of dynamical systems. We consider them in this chapter, concentrating on the basic concepts and leaving specifics for further chapters.

3.3 The State

What characterizes the notion of "state"? Over the years the following insight has crystallized (see [26]):

> A state at a given time t (or index k when discrete time) is sufficient information on the system at that given time point t (or k) to determine, given future inputs, its future evolution.

This means, among other things, that for systems with a finite state vector the whole past evolution of the system up to time t is, as far as the future is concerned, fully accounted for in the state vector at time t. We don't need more information on the past to assess the future evolution, given future inputs, or, to put it more negatively, the system forgets everything from its past except what is contained in its state. This is somewhat illustrated in fig. 3.1.

The definition given leaves a somewhat arbitrary choice dangling: where is the 'cut' between the past and the future with respect to a time point t or a time index k? The traditional choice, in the wake of [26], is to let the past run to just before t (often written as $t-$) or $k-1$ for discrete-time systems, and to let the 'future' start with the present t (or k) — this is already assumed in the equations given so far. In statistics often the opposite choice is made,

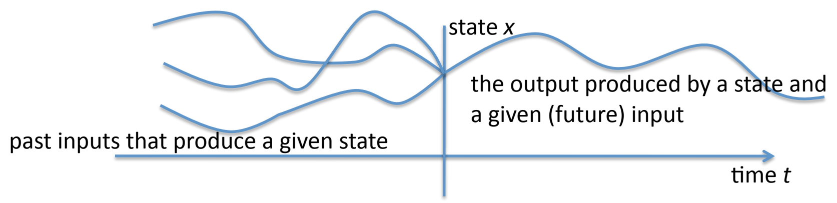

state x
the output produced by a state and a given (future) input
past inputs that produce a given state
time t

Figure 3.1: The state as the link between past and future

and sometimes one sees mixed choices for inputs and outputs. The problem arises in the case of discontinuities and has to be considered carefully once it occurs.

3.4 Basic system characteristics

State minimality

In the state description just given, the state is described as 'sufficient' information. Minimality would actually require also 'necessary' information. A computer memory is usually filled with all sorts of information that is not relevant to the problem at hand. A state is called *minimal* when no data in the state vector at time t can be left out without potentially affecting future outputs, at least for some well chosen inputs. A more precise formulation is: so that no future input exists for which the system's evolution would be different. This leads to the notion of *Nerode equivalence*. We say that two past inputs up to time '$t-$' are 'Nerode equivalent', if there is no future input (i.e., starting at time t) that will produce different future outputs.

It should be immediately obvious that the notion of minimality depends very much on the output equation as well. It may be that part of the system internals will never be visible just observing the output, in which case they could be deemed superfluous. However, a completely autonomous system has neither inputs, nor outputs. Hence, a minimal description for it would be empty. This unpleasant situation can be remedied by the convention that in the case of an autonomous system, one can observe the state directly, so that the output equation simply becomes $y(t) = x(t)$. Since there are no inputs any more in this case, the minimal state determines the output evolution uniquely. (The great breakthrough of Newtonian mechanics was the realization that the velocity of the planets belonged to the state, in contrast to what the pre-Newtonian Western world thought, which limited the state just to position data; they thought that a change of position directly depended on (was proportional to) instantaneous forces.)

An important further consequence of the notion of a minimal state is that its quantities (components of the vector) are *algebraically independent*. They can be assigned arbitrary values (of course within the number system used, in our case they will be either real or complex numbers), independently from each other. Here also, there is a potential unwarranted generalization. It is conceivable that independent state variables may only take limited sets of values. That is e.g., the case in a computer, which does not allow any size of number, but also real life systems are limited by ranges of relevant variables. The mathematical formalism conveniently ignores these contingencies, hoping that the theory shall be able to handle them when actually necessary.

Reachability

Returning to the characterization of a non-minimal state, it may be that some states at some time point t cannot be *reached*, i.e., there is no past input up to time t that is able to produce the state. States that can be generated by at least one past input function are called *reachable*.

Observability

It may also happen that some states at a given time point t will produce the same output in the future, no matter which future input is applied. The distinction between such states is called *unobservable*: whether the system has one or the other state at time t cannot be determined by observing the evolution from t on, using whatever input from then on. We call the state *observable*, when each such equivalence class of states that all produce the same future output for whatever future input consists of only one element. Actually, every equivalence class of states whose distinction is unobservable can be represented by just one state.

The behaviour of a system

We use the term 'behaviour of a person' to indicate how a person's actions and reactions appear to us, not how the person actually produces them physically. The notion involves both the person and how the environment perceives them[1]. In the case of a system it would be "how the world perceives the system globally". As the external world has access to the system solely via its inputs and outputs, one could say that the system behaviour is "how the system attaches outputs to inputs over all time". In mathematical terms, this would be formulated as a relation between input over all time and corresponding output over all time. At issue could be whether a system behaves deterministically: one could conceive a case whereby within a system there are some hidden actors that cause the system to move one way or the other independently from the actual inputs. Needless to say, such a viewpoint would greatly complicate formal descriptions. In the section on estimation theory we shall allow such interferences, but shall be careful to assign additional inputs to them, so that the erratic behaviour is taken out of the system's internals. That turns out to be a good strategy in many cases, at least where engineering is concerned. Therefore, many treatments of dynamical systems will assume that the system, given an initial state at some initial time t_0, defines maps from the input starting at t_0 and running to some final time $t_1 > t_0$ to (1) the state at t_1 and (2) the output from t_0 to t_1. This map is then the *functional behaviour* or *transfer map* of the system.

From the point of view of an observer of the system, the system may produce some other types of behaviour that are worth noting, and that we would catch under the term 'behaviour' as well. For example, it may heat up, or it may age, or it may develop some other physical characteristics that are not directly related to how it produces an output given an input, such as a change of size, color or form (these might even be taken among the outputs.). Certainly, the term 'behaviour' would always relate to something in the system that changes over time and can be observed. In engineering practice, one always selects a limited number of variables for which one wants to define the dynamics (i.e., the state variables), relegating all other (often long term changes) to evolution of the system's structure over time, calling it a (maybe slowly) 'time-variant' system. Although one could actually assign states to such characteristics, it turns out to be more practical to leave them as system parameters that change over time — as you can see, the assign-

[1]We use a plural for non-gender specific reference.

ment of state variables is often in some sense arbitrary (e.g., if you want to control an airplane, you do not bother about changes in loads, which you conveniently summarize as one parameter: actual weight).

The late Jan Willems defines the *behaviour* of a system as [44]

> 'A mathematical model is a subset of a set of a priori possibilities. This subset is the behaviour of the model. For a dynamical system, the behaviour consists of the time trajectories that the model declares possible.'

Willems attaches the notion 'behaviour' to an intrinsic mathematical model of a system. He thereby excludes inputs and outputs to keep the definition untainted from outside influences. Such a definition has advantages and disadvantages. An important advantage is a precise definition of scope. A disadvantage is that the definition is 'mathematical', in that differs from the normal, non-mathematical usage of 'behaviour'. The term behaviour is also used in computer science with a somewhat different meaning, where how the system interacts with the outside world is at stake, not how it achieves its actions internally. We have defined behaviour in the more common sense, and have included inputs and outputs in the definition. The disadvantage of our definition is that it is more remote from the problem of mathematically modeling systems in a physically sound way than Willems's definition, but the advantage that it leans towards synthesis, signal processing and design engineering, where behaviour is seen as a goal to be achieved (namely a desired input-output behaviour) rather than the characterization of how a given mathematical system model is thought to evolve internally. To conform with Willems's terminology, we could call our usage 'Input-Output behaviour'. Nonetheless, the methods we use, and in particular inner-outer factorization, come very close to the treatment Willems gives based on his notion of behaviour. The two notions can be brought together with a bit of good will both sides. Fully correct mathematical modeling of a system is never possible since every model involves abstractions and approximations, and correct modeling necessitates observation of the system, hence outside influence. Willems recognizes this, but reserves the term behaviour to properties of the mathematical model, while in our usage it refers to an observed or desired input-output relation. Whatever usage is utilized should be made explicit from the start.

3.5 Concluding remarks

All these concepts: state, behaviour, reachability, observability, play an exceedingly important role in the development of dynamical system theory. Almost all the properties of a system can be derived from them. A central thesis of this book is that these few basic notions are able to produce major mathematical leverage and thereby solve central problems in estimation theory, control theory, numerical linear algebra and circuit theory. To conclude this introductory discussion, here is a quick run down of some typical dynamical systems and their state spaces.

Mechanical system:	position and velocity
Computer:	various types of memory
Automaton:	control states, routing states
Airplane:	position, velocity, roll, yaw and pitch angles
Process plant:	pressure, temperature, concentrations
Brains:	synapses

Discussion items

1. Given the non-minimal state set of a system, how could one derive a minimal set?
2. Propose more examples of dynamical systems, and logical state sets for them.
3. Given a system, one can create a system to observe it. What would be the state of the latter?
4. Systems can be build from other systems. How would that work? What would be the state of the overall system? How about its minimality?
5. An electrical circuit is a dynamical system. What is a reasonable state for it? The same question holds for a living cell.
6. State equivalence: given a state set for some system, could one derive new state sets for it purely formally?
7. To analyze a new dynamical system one would have to discover a state set for it. How could one do that?

8. The dynamical systems considered in this chapter have states that evolve either through a differential equation (for continuous time) or through a difference equation (for discrete time). What could be more general types of systems?

Chapter 4

Types of dynamical systems*

Before developing the theory for a our selected prototype of dynamical system, let us make an inventory of the various types one encounters and how one may describe them mathematically. In the previous chapter, we have identified basic notions that are characteristic for the notion of dynamical systems: inputs and outputs, the state, the state evolution, reachability and observability. These are characteristics that all types of dynamical systems will share, and the issue we consider in this chapter is how these notions appear in a concrete mathematical description, and how such descriptions relate to each other. The chapter also serves as a motivation for our choice of prototype system environment: discrete time, linear, time variant systems, because, from a computational point of view, many system problems can and will be brought to this LTV set up.

Menu

Hors d'oeuvre
Inputs and outputs

First course
The state

Second course
State evolution

Third course
Behaviour

Causality

Dessert
Generalizations and Discussion Topics

4.1 Inputs and outputs

Inputs and outputs are *quantities* that vary in *time*[1]. To start with time, it is customary to assume a unique time variable—often labeled t—valid for the full system, including state variables. The simplest distinction was already given in the previous chapter: continuous-time and discrete-time. Continuous-time would be a real number (often called t), and it would typically run from $-\infty$ to $+\infty$, by which we mean that it has no fixed beginning and no fixed end. Clearly if a system had a fixed beginning or end, this could be embedded in the previous: embedding is a strategy we shall often adopt, so we shall not consider 'sub-cases' separately. It turns out that this will not lead to major difficulties. Next, *discrete-time* would be just an index (an integer) also running from $-\infty$ to $+\infty$. Here the situation is a bit more delicate: 1., one could think of these indices to represent time points that are spread regularly over the time axis with constant intervals (this would be the normal discrete-time case) or 2., they could be representative of moments in time when something happens, not necessarily regularly spaced (like what happens in a waiting line), in which case we would talk of a *discrete event signal.* There are even situations where in one given system, several time scales are present, e.g., in a sampling system where the input would be continuous-time and the output discrete-time, or in a system in which different sampling rates are present. What also often happen in practice is that the system has an overall regular clock, but that many of its subsystems have their own clocks. The strategy to be followed in such cases will probably be case-dependent, but an effective strategy is to make all these various timing signals dependent on one general time (continuous or discrete when possible), so that one does not loose track of how they relate to each other.

In this book, our starting point will be discrete-time, equally spaced tim-

[1]In more advanced theories, there might be 'space-time'. Or various parts of a system may have different times. For example: various computers with different clocks communicating to each other. There are many more possibilities, but they will not concern us here.

ing points, but as we go further we shall venture in some other timing systems, continuous-time and even some discrete event systems. It will turn out that there is a relatively easy way of connecting (regular) discrete time with continuous-time algebraically, allowing the transposition of important results from one domain to another.

Concerning the *values* inputs and outputs may have: here also a great variety is possible. Inputs and outputs are function of time, that issue we just discussed. The next question is: to what range do they belong? In the simplest case, they would be one-dimensional real ($\mathcal{R}$) or complex valued ($\mathcal{C}$). Let us first consider the discrete-time case. What comes to mind is how a computer operates: at regular time intervals it takes in new data from its input devices, does some computations on it and then outputs to output devices, after which it renews the cycle. In each such cycle, the computer may take in data as needed from a variety of sources and output data to various data repositories. So we would typically assume that 1., the data has a vectorial character, and 2., the dimension of the vectors taken in or outputed may change from one event point to the next—hence inputs would belong to $\mathcal{R}^{m_k}$ or $\mathcal{C}^{m_k}$ where m_k is the input dimension at index point k, and outputs would have the form $\mathcal{R}^{n_k}$ or $\mathcal{C}^{n_k}$, with n_k the dimension of the output data vector. This means that in such systems, the input is an irregular sequence of vectors that are either real or complex, and likewise for the output. *This is the point of view that we shall mostly adopt in this book.*

For continuous-time systems, the situation is more tricky. The traditional approach is to assume that the inputs and outputs form some function of time to a real or complex vector space of fixed dimension (hence $\mathcal{R}^m$ respect. $\mathcal{R}^n$ for some fixed positive integers m and n in the real case.). Traditionally, one also assumes the continuous-time system to be time-invariant, but that is a massive reduction of the field of interest, often justified, but actually not really necessary in many cases. When one is not satisfied with these restrictions, one would have to conceive continuous-time systems in which the dimensions of inputs and outputs may vary when time evolves. That would make such a system already automatically a combination of discrete events and continuous-time evolution between events. In some later chapter we shall consider that case to some extent.

But why should the collection of inputs and outputs be discrete (vectorial)? It is of course easily conceivable to have systems whose inputs and/or outputs are themselves function of another variable besides time, e.g., a space variable. Think e.g., of a boat in the waves (waves impinging on its full length) or, similarly, a near field antenna in an electromagnetic wave. Often, such cases can be treated by discretizing the continuously parametrized input (or output), so we shall not pay special attention to them.

Finally, why should an input or output be a real or complex number? In our computer age, we could just as well assume them to be bits, or bytes or to belong to some other field, as happens in coding theory. Although we shall not consider the case of such *digital systems* in this book, let us mention that all the basic notions that we are considering do apply to them as well (e.g., reachability and observability play an important role in digital optimization and testability of digital systems), and that the conversion of real and complex data to digital data requires consideration. Many problems can be dealt with by careful development of algorithms that are robust for rounding—and this is an issue that we shall consider.

4.2 The state

The state of a system is also a function of time, and hence will be subject to a similar taxonomy as the inputs and outputs. So, at a given point in time, it can be a real or complex vector (i.e., a function of an index) or a function of some continuous parameter (e.g., the length of the rod of a cantilever). In a discrete-time system, the dimension of the state may change from one time-index to the next. In a continuous-time system it is common to fix the dimension of the state over all time, unless one considers the already mentioned combination of discrete events and continuous-time evolution between events (after all a situation that can easily occur in practice). In most of the chapters of this book, we shall assume a state that evolves in discrete time, and whose dimension can vary from one time point to the next. This is in line with what a state can be in a computer memory: evolving in time stepwise but whereby the size of the required memory may change. At some point, we shall also develop arguments to show that in many cases a time-variant but continuous-time system can be accurately discretized.

There are of course many cases in which the state of a system is not a finite dimensional vector. It may also be that it is finite dimensional, but that the vector structure is not really appropriate for it, its natural structure would rather be e.g., a matrix or a tensor or some other entity with structure. Such additional structures are very interesting, in particular as they may lead to efficient algorithms, but their extensive study would lead us too far afield, although we shall consider some interesting cases of states with structure, in particular matrices or tensors. The treatment of state variables that are continuously dependent on some parameters (e.g., space) is beyond the scope of this book—suffice it to say that in many cases careful discretization of the continuous parameter may solve the problem.

4.3 The evolution of the state

As there are many possible types of states, there shall be many possible ways in which a state can evolve. However, since our main focus is on discrete-time systems (typically mapped on a computer), we shall generally assume the state evolution to be an index-varying map at each index point k, which maps the vectorial state $x(k)$ and the vectorial state $u(k)$ to the next state $x(k{+}1)$, as shown in eq. 3.1. Here the state *transition function* f_k is a general non-linear function. (One may assume without too much extra difficulty that the state-evolution is restricted to a (finite dimensional) manifold.)

Most of the theory we shall develop at first makes a sweeping further assumption, namely that the transition function is actually linear, which means that both the state evolution and the output equations are described by a set of linear difference equations:

$$\left\{ \begin{array}{lcl} x(k+1) & = & A_k x(k) + B_k u(k) \\ y(k) & = & C_k x(k) + D_k u(k) \end{array} \right. \tag{4.1}$$

in which A_k, B_k, C_k and D_k are matrices of appropriate dimensions, which may vary with the index k. There are two main motivations for this choice:
1. linear systems do occur a lot in practice, as matrix computations do occur a lot;
2. many properties can be derived from studying *system variations* i.e., differentials, which turn out to be linear but time-variant.

This works as follows: the state of a non-linear system—say $x(k)$—follows a *trajectory* function of k, imposed by an initial state and an input sequence $u(k)$. Under some continuity requirements valid for many real-life systems, one may assume that a close-by initial state and a close-by input sequence generate close-by trajectories. Let the variational operator be denoted by δ so that a new, close-by trajectory is written as $x(k){+}\delta x(k)$ and the new input as $u(k) + \delta u(k)$—this actually defines the operator $\delta x(k) := x_{\text{new}}(k) - x(k)$ and likewise for $u(k)$, then, and assuming the differences to be very small, we shall have to a first order[2]

$$\left\{ \begin{array}{lcl} \delta x(k+1) & = & A_k \delta x(k) + B_k \delta u(k) \\ \delta y(k) & = & C_k \delta x(k) + D_k \delta u(k) \end{array} \right. \tag{4.2}$$

whereby $A_k := \partial_x f_k$, $B_k := \partial_u f_k$, $C_k := \partial_x g_k$ and $D_k := \partial_u g_k$. In other words: the variation of the trajectory is a time-variant linear system. The difficulty with this approach is the fact that the resulting linear variational system is trajectory dependent. To obtain properties of the original system from this, an integration has to be performed from the differential system to the original. We shall spend a chapter later in this book to discuss this issue.

The same strategy can of course be followed in the continuous-time case, with the additional difficulty that it may interfere with discretization—discretization in both time and in the state space has to be done consistently. This can be achieved elegantly for an important class of physical systems, namely those described by a Lagrangian—again a topic for a later chapter.

Some classes of systems have received much more attention historically than others. To be mentioned are both continuous-time and discrete-time, linear time-invariant systems (so called *LTI systems*) with a finite dimensional real or complex state space. No doubt, they are important as they cover important application domains (elementary electrical circuits and elementary control systems), but, surprisingly, they are less elementary than the time-variant variety that form the backbone of this book. The reason for this is that time-invariance imposes global constraints that are often hard to fulfill. For example: the control of an airplane is very much dependent on the local speed, pressure and perhaps also temperature, which change all the time. From our general and elementary time-variant treatment we shall in a few chapters specialize to LTI systems and derive the additional properties needed. Here we suffice to indicate the state evolution of the two types.

In the continuous-time case, the state evolution of an LTI system with finite dimensional state space is given by a set of ordinary differential equations:

$$\left\{ \begin{array}{lcl} \dot{x}(t) & = & Ax(t) + Bu(t) \\ y(t) & = & Cx(t) + Du(t) \end{array} \right. \tag{4.3}$$

in which the dot indicates time-derivation and the matrices A, B, C, D are constant (in the time-variant version they would be time dependent.). In the

[2]We shall use the expressions to come only much later, in Part II, so the detailed form of the differential matrices A_k etc. is not of importance at this point of the discussion.

discrete-time variant, the state evolution is given by a difference equation and the state-space equations become:

$$\begin{cases} x(k+1) & = & Ax(k)+Bu(k) \\ y(k) & = & Cx(k)+Du(k) \end{cases} \tag{4.4}$$

There is a substantial distinction between the two cases. In the continuous-time case, the A matrix is a *generator* of a state evolution 'semi-group', while the zero-input state transition is properly described by an operator of the type e^{At}, while in the discrete-time case, the A matrix properly transfers the state from time-point k to $k+1$, assuming the input $u(k) = 0$. The distinction will become much clearer when we discuss discretization in chapter **??**.

In the wake of the extensive historical development of LTI system theory, people have attempted to extend the theory to more complex cases than just covered by finite dimensional state spaces. A case in point are *delay differential systems*, which are systems in which a finite number of delays of the state may influence the state evolution. Strictly speaking, such systems have infinite state spaces, but they have a structure that allows for a description with finite matrices. Examples are circuits that contain transmission lines or control systems in which interactions between components are delayed, as would happen if a control signal has to be transmitted over a communication link. Needless to say, the additional structure can, on the one hand, be exploited to simplify treatment, but, on the other, will result in a much higher theoretical complexity. In many cases, the additional structure can be accounted for by an adequate (discrete) computer model in which the transition matrices have some interesting structure (like multiple band). As we already indicated, we shall only scantily touch on this topic.

Concluding, a good balance for a textbook on the relation between dynamical system theory and computational algebra is provided by discrete-time, time-variant linear systems, as most more complex cases end up for practical treatment in that category.

4.4 Behaviour

We defined behaviour as *how a system is seen from the external world*, specifically what is called its 'functional behaviour': how inputs and outputs are globally related to each other mathematically. The concrete mathematical description of the system's behaviour will depend on the type of system, including the type of inputs, outputs, state and state evolution, but one thing is common to all: *the functional behaviour is the map from the inputs to the outputs after elimination of the state*, at least when such a map exists (otherwise one considers the relation). To define such a map properly, a host of sometimes nasty mathematical assumptions have to be made, mainly because the allowable sets of inputs and outputs have to be defined properly for the behaviour to make sense. For example: it does not make sense to allow any possible input sequence from $-\infty$ to $+\infty$ as input and require the system to produce a reasonable output consistent with the state equations, what if e.g., the input values for large negative times tend to infinity?

Another issue related to behaviour, is what to do with starting up the system? It is definitely impractical to attach a specific 'creation time' to a system and a starting value of the state at that time—this turns out to be unnecessary. In a time-variant system, this situation is easily circumvented (as we shall soon see), by allowing an indeterminate start up time before which the state (and all inputs and outputs) are simply empty. The issue is more a problem in LTI systems, where 'start up' actually destroys time-invariance. Traditionally, one circumvents the problem in two non-compatible different ways: 1., by allowing as inputs only signals that have a limited carrier for large negative times (i.e., that are zero for any t smaller than some signal dependent t_{start}) and whereby the system is initially in the zero-state, or 2., by assuming from the start that the behaviour is a map between normed spaces of inputs and outputs consistent with the state evolution, i.e., where there exists a state evolution, usually also normed in some sense, that is consistent with the map.

Which choice is actually made will depend both on physical considerations and mathematical consistency. Many physical signals and systems have energy constraints, so one can often require the inputs to belong to some L_2 (in the continuous-time case) or ℓ_2 space (for discrete-time), and then require the output to belong to such a space as well, which will put constraints on the system description. For example, it may be so that the system permanently adds energy to the output, making the map from input to output unbounded. This is the case of an integrator (in the continuous-time case) or an adder (in the discrete-time case): for the latter case, suppose the system starts adding at $k = 0$ then at some integer $K > 0$, we shall have $y(K) = \sum_{k=0}^{K} u(k)$. Even when the series $u(k)$ is bounded in energy ($\sum_{k=0}^{K} |u(k)|^2 < E$ for some maximum energy input E), such a bound will not exist for the resulting series $y(k)$): the behaviour is unstable in the norms chosen. However, that does not mean that the system is ill defined: it is merely behaviourally unbounded. It will depend on the application domain what type of input-output description is appropriate, and how the (external) behaviour relates to the (internal) state description. Such considerations appear to have a great impact on the mathematical development of system theory in specific mathematical environments (it is maybe unfortunate that there does not seem to exist a unique framework covering all instances!).

So, let us concentrate on what to do in the discrete-time LTV case with finite state vectors (the central case in this book). Also here we have two choices: either force the system and its inputs to start at a certain (not necessarily known) point in time, and thereby allow 'unstable' behaviour, or force the system to achieve a bounded map between normed input and output spaces. The first approach would often be used in control applications, where instability and the control thereof plays a major role. The second approach is more appropriate for signal processing or numerical analysis, where numerical stability and boundedness plays an important role. As we shall soon see, the two approaches are not compatible with each other, although the relation between the two can be profitably studied. It turns out that the mathematical analysis for the second case is easier than for the first, because the properties the system under consideration may have are mathematically more restricted.

To make the last point more precise, let us explore the differences between the two approaches in our LTV case. The state description then has the form given in eq. 4.1. Assuming all states and inputs zero or empty before some index k_0, we find, by eliminating the states, that the output for $k \geq k_0$ is given by (see chapter 5 for more details)

$$y(k) = D_k u(k) + \sum_{i=k_0}^{k-1} C_k A_{k-1} \cdots A_{i+1} B_i u(i) \tag{4.5}$$

where the *continuous product* $A_{k-1} \cdots A_{i+1}$ (sometimes denoted as $A^{>}_{k,i}$) only exists for $i < k-1$ and is taken to be I for $i = k-1$. These equations describe the relation between input and output, and are perfectly well defined because the summation is in all circumstances finite. However, when k increases, the sum may get out of bounds, and this will especially happen when the continuous product diverges. In the LTI case $A^{>}_{k,k_0}$ has the form A^{k-k_0-1}

and divergence will happen when A has eigenvalues greater than 1, that is, when the system is unstable. Moreover, the instability is exponential, which in numerical computations is definitely a disaster (we shall explore this question further in the chapters on inversion.)

The second approach uses norms on the input and output sequences and requires the output series $y(k)$ to be (uniformly) bounded when the input series is. The traditional approach in numerical analysis is to require the time series in the input and output spaces to be bounded in energy, i.e., to be of ℓ_2-type. Once this is agreed on, one can drop the requirement of having a starting time because both the maps from input to state and from input to output map will be well-defined, even if one allows the signals and the system to run for all times. The details on how all this is done is relegated to chapter 5, let us suffice here to state that the approach greatly simplifies the mathematics at the cost of introduction of some elementary Hilbert space theory. In particular, the input-output map gets the simple form $T = D + C(I - ZA)^{-1}ZB$ in which A, B, C and D are diagonal operators, Z a simple shift operator and the inverse is guaranteed to exist as a bounded operator.

Connecting the properties of the behavioural or input-output map to the internals of the system is a major issue in dynamical system theory, known as *system identification* and we devote chapter 6 to it for the case of LTV systems. A central role in this question is played by what we shall call the *Hankel operator*, which by definition is the operator that describes how the system maps (strict) past inputs to its future outputs, excepting or modulo the contributions of present and future inputs, and this at each time point (this is explained in detail in chapter 6. The Hankel operator characterizes the contribution of the minimal state of the system at each time point needed to uniquely determine future outputs from that time point on given (present and) future inputs. It is a sub-operator of the input-output map, which is instrumental in singling out what a minimal state has to contribute, and it factors into the composition of a *reachability operator*, which produces the state from strict past inputs and an *observability operator*, which then describes the contribution of the state in the output, and this at every time point. These operators form the gist of dynamical system theory. They exist in all known types of dynamical systems, with more or less the same properties in each case. The Hankel operators play a central role in dynamical system theory!

4.5 Causality

Dynamical system theory evolved with the progression of time. Just like number theory and algebra evolved: after the positive integers came 'zero' and then the negative integers, the rational, the real, the complex numbers etc., and this to great benefit. There is no reason to stick with just positive time evolution. Why not consider a new type of time that evolves negatively? That means: the system's dynamics evolves with decreasing time—a new kind of system evolution very much 'dual' to the original. To be able to talk about the distinctions, let us call a system that evolves with positive time, as before *causal*, a system that evolves with decreasing time we would call *anti-causal*.

Restricting ourselves to systems described by ordinary differential or difference equations, how would an anti-causal system be described? Looking first at the differential equation:

$$\left\{ \begin{array}{lcl} \dot{x}(t) & = & A(t)x(t) + B(t)u(t) \\ y(t) & = & C(t)x(t) + D(t)u(t) \end{array} \right. \tag{4.6}$$

there is no other difference than that the integration would be executed in reverse order, for negative times. The time reversal happens completely outside the system, it is in the eye of the beholder. (We know that Newtonian systems are time reversible!)

However, with discrete-time systems, the situation is different: a causal system would move from index point k to index point $k+1$, while an anti-causal system moves from index point k to index point $k-1$—this is substantially different, as the evolution may not be reversible in this case. This being as it may (we shall exploit it extensively further on), the convention we shall adopt is to always take the state $x(k)$ as the input of stage k, so that the description of an anti-causal time-discrete system becomes:

$$\left\{ \begin{array}{lcl} x(k-1) & = & f_k(x(k), u(k)) \\ y(k) & = & g_k(x(k), u(k)) \end{array} \right. \tag{4.7}$$

for some functions f_k and g_k. In this convention, which shall appear to be consistent with matrix calculus, the incoming state $x(k)$ inputs at a different position in the causal system than in the anti-causal (do you see that: it inputs at the index $k+1$ of the forward system?).

The indexing conventions we adopt for discrete-time systems are somewhat arbitrary; other conventions appear in the literature. It should become clear in chapter 5 that they are the most natural for translating matrix calculus to dynamical system operations.

The questions immediately arise whether the causal and the anti-causal descriptions can be mixed with each other and whether a causal description can be inverted to an anti-causal one with the same behaviour, i.e., the same overall relation between inputs and outputs. These questions are closely related to system inversion and will be treated extensively in this book, in particular in chapters 9, 10 and **??**, so we do not discuss the question further at this point.

4.6 Generalizations

In this book, we restrict ourselves strictly to a monotonous time, either increasing or decreasing. In other words: time forms a completely ordered (countable) lattice. Nothing prevents one from considering more general 'time'-lattices. One would be just an ordered lattice with origin, i.e., a lattice in which two elements have at least one common ancestor (and hence also a minimal one). More general lattices have been considered as well, in particular regular 2D or 3D lattices as they occur in image or video processing, i.e., partial orders in space. Needless to say, system theory on such structures becomes much more complicated, because, in more general lattices, there are many ways in which evolution can take place and descriptions hence inherit a large measure of arbitrariness. In this book, we take the view that time is a fully ordered lattice, and it can only be increasing or decreasing, while all other ordering principles (e.g., in space) are 'local' at any given time point (and may even change between time points.). In first instance, the only local order we consider is the ordering in the vector representation, but in some later chapters we shall consider more extensive local orders, e.g., multi-band transition matrices or transition matrices that themselves have a 'local' state space structure.

Also the discussion on behaviour can be made more general than we have done. One very fruitful approach is based on 'nest algebras' [4]: starting from the definition of unilateral time as a totally ordered set, to each point in time t one may attach all acceptable inputs up to that time point, calling it a set $\mathcal{U}_t$—containing all the inputs that run from $-\infty$ to and including t. Clearly when $t_1 \leq t_2$ one shall have $\mathcal{U}_{t_1} \subset \mathcal{U}_{t_2}$, the collection $\mathcal{U}_t$ forms what is called a *nest algebra*. Usually one shall impose some continuity on that algebra as well, requiring e.g., $\mathcal{U}_t = \cap_{\tau > t} \mathcal{U}_\tau$. This then allows to define $\mathcal{U}_{t-} := \cup_{\tau < t} \mathcal{U}_\tau$, thereby identifying what happens exactly at a continuous time point t. Also the output space can be given a similar structure based on the same time set, and a map would correspond to a causal system when, at each time point t, all inputs that coincide up to (and not including) t will map to the same partial output up to, and including t. Most of the important system notions (e.g., Hankel operator, inner-outer factorization) extend to such systems defined on nest algebras. However, it would lead us too far afield to consider this highly interesting and relevant generalization, given our aim to keep all treatments matrix computational.

4.7 Concluding remarks

The discussion in this chapter focusses on some very basic assumptions and considerations, that are often overlooked or considered 'trivial'. However, the more basic the assumption, the more consequences it has for the further development of the theory, so it pays to put enough care to develop them. We did not do that fully as of yet, for that we need a better vista on these very consequences, but they will soon appear when we proceed. Some are technical, and determine how easy or complicated the resulting algebra will become. It has been thought for a long time that the time-continuous, time-invariant case is the simplest, but it will appear that this assessment does not pan out, although one can treat some of the estimation and control problems elegantly, with simple expressions, in that case.

However, from the point of view of the simplest possible algebra, the discrete-time, time-variant case stands out, because it relates directly to matrix algebra. It allows for the simplest possible treatment of all the basic system theoretical ideas that we already met schematically in this chapter (state, reachability, observability), and this with an impressive generality, de facto capable of handling completely general matrices (while the time invariant case is restricted to matrices with a lot of structure: infinite dimensional 'block Toeplitz matrices' for transfer matrices and (infinite dimensional) 'block Hankel matrices' for what we have called Hankel operators).

Some assumptions appear to be fundamental in the sense that they have a great influence on the type of theory that follows. One such is whether one chooses to consider bounded systems on normed input and output spaces, or whether one would allow the system to start at some finite point in the past and then let it evolve unconstrained. In the first case, a 'dichotomy' appears, which allows for new system properties to appear, that do not reveal themselves easily in the second case. The complementarity of the two cases can sometimes be exploited, as we shall see when we deal with system inversion.

4.8 Item for discussion

How would one set up a dynamical system theory for the discrete event case?

Chapter 5

LTV (semi-separable) systems

In this chapter we specialize the type of dynamical systems to ‘linear, time-variant, discrete time’ (and then further to time-invariant systems). Although such systems may not be the first type studied historically, they are the systems that allow for the most straightforward mathematical and numerical treatment. Because of the prevalence of modern computers, they are also the type that best meshes with computer calculations and numerical linear algebra, where they are often called ‘semi-separable systems’ or sometimes ‘quasi-separable systems’ (see the notes at the end of the chapter on this.). This connection works in two directions: systems that compute can be considered dynamical systems and vice-versa. Our starting assumptions are: the time is discrete and represented simply by an index that runs from $-\infty$ to $+\infty$. The chapter then develops our formalism for LTV systems, concentrating on straightforward, easy to use generic matrix representations for the state space evolution and the input-output map, both for a causal and an anti-causal system. We introduce block-diagonal representations, the generic shift and the generic transfer function representation for the LTV case. Next we discuss closely related issues such as the notions of causality, anti-causality, stability and duality. Further important system theoretic notions are postponed to the next chapters; the aim of the present chapter being to develop familiarity with the relation between systems and matrix algebra.

Menu

Hors d'oeuvre

Why linear time-variant (LTV) and discrete-time systems?

First course

The formalism for causal systems

Second course

Uniform exponential stability

Third course

The formalism for anti-causal systems

Duallity

Fourth course

The diagonal notation

Dessert

The LTI case

5.1 The formalism for causal systems

Let us first consider systems that run forward in time, using only past inputs (causal systems). At index point k, the system will have a state x_k of dimension η_k and will receive an input u_k of dimension m_k. Stepping from k to $k+1$, it determines a new state x_{k+1} of (potentially different) dimension η_{k+1} and produces an output y_k of dimension n_k. In other words, there is a *local transition map* $(x_k, u_k) \mapsto (x_{k+1}, y_k)$ happening at 'clock index k' producing the next state x_{k+1}, a local output y_k and moving the system to the next index point $k+1$. The local transition map would in general be non-linear, but we start out by assuming that it is linear, leaving the generalization to non-linear for later (in Part II).

For mathematical precision, one would normally specify to which type of vector space the various vectors one is considering belong. In this book, we compute with real or with complex numbers ($\mathcal{R}$ or $\mathcal{C}$)—even though many properties would also work in a finite field (like p-adic numbers), not considered here. So we write: $x_k \in \mathcal{R}^{\eta_k}$ or $x_k \in \mathcal{C}^{\eta_k}$. Actually, it does not matter much whether the numbers are real or complex for the theory to work, and, moreover, the reals can be considered a special case of complex numbers. We shall define our notation in such a way that things work out independently of which case is being considered (in many cases the computations will just be real).

A linear map $a : \mathcal{R}^m \to \mathcal{R}^n : u \mapsto y = a(u)$ (often just written $y = au$) is represented by an $n \times m$ real matrix when the natural bases in the input and output spaces are used, let us call it A, so that $y = Au$, or with indices $y_k = \sum_{i=1}^{m} A_{k,i} u_i$. This is an example of matrix-column multiplication, that assumes vectors to be represented by columns and the linear map by matrix multiplication. A row-based system would work equally well, and we shall sometimes use it, but most of the book will use the column representation for vectors (in modern differential geometry people use an "Einstein index notation" and do not really specify which type or matrix representation they use.).

It is often useful to subdivide vectors into sub-vectors, or to assemble vectors into new, larger vectors. Such operations are very common in e.g., MATLAB, and we shall adopt MATLAB's notation conventions to construct new vectors (we shall even extent the conventions, see further.). Column vectors can be concatenated: suppose e.g., that u_1 is a column vector of dimension n_1 and u_2 a column vector of dimension n_2, then one could define a new column vector u of dimension $n_1 + n_2$, which we could denote as $\mathrm{col}(u_1, u_2)$, using a *constructor* 'col'[1]. If $n_1 = n_2 = n$, then one could also stack the two vectors and create an $n \times 2$ matrix $\begin{bmatrix} u_1 & u_2 \end{bmatrix}$. Many such constructs are possible (we shall explain what is happening either in detail when needed, or else assume that it is clear from the context.). The distinctive characteristic of a matrix is that it is a rectangular block of data with precise dimensions. When assembling matrices, or do operations with them, dimensions should always match properly. E.g., a matrix A of dimensions $n \times m$ can only multiply a matrix B of dimensions $k \times \ell$ to the left when $m = k$. Similarly, a sub-division of the matrix $T = \begin{bmatrix} A & B \\ C & D \end{bmatrix}$ requires the number of rows of A and B as well those of C and D to be equal and the number of columns of A and C as well as those of B and D.

With the conventions so far, our discrete-time, linear and time-variant system would then satisfy a transition equation at each index point k of the

[1]One often sees expressions like $\mathbf{u} = \begin{bmatrix} u_1^T & u_2^T \end{bmatrix}^T$, which are unnecessarily cumbersome. We prefer $u = \mathrm{col}(u_1, u_2)$, much simpler and effective, especially when one wants to make more complex constructs as we shall be doing.

type:

$$\left\{\begin{array}{lcl} x_{k+1} & = & A_k x_k + B_k u_k \\ y_k & = & C_k x_k + D_k u_k \end{array}\right. \tag{5.1}$$

in which A_k has dimensions $\eta_{k+1}\times\eta_k$, B_k dimensions $\eta_{k+1}\times m_k$, C_k dimensions $n_k\times\eta_k$ and D_k dimensions $n_k\times m_k$. A_k is called the 'state transition map', B_k is a map from the input at index k to the next state, C_k is a state-to-output map and D_k is a so called 'feed-through', it connects the input at index k directly to the output at index k (from a computational point of view there will be a slight delay, which is neglected, assuming a computer to calculate at infinite speed. If one does not like the feed-through term, one may put it zero! However, keeping the term simplifies many calculations and enhances generality. It also plays an essential role in the Kalman filter, see chapter 11.).

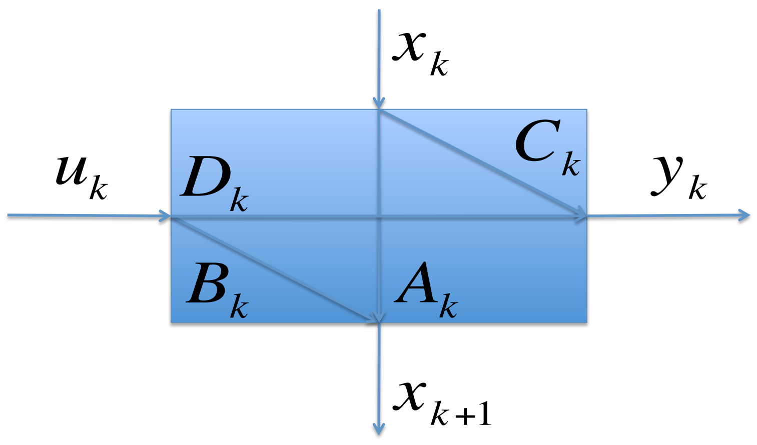


Figure 5.1: Signal flow diagram of a discrete-time, linear and time-varying causal system.

5.2 Input-output behavior

Suppose now we were doing a computation as follows: the calculation starts at index 1 with no input state (x_1 is empty). It takes in a first input u_1 of dimension m_1, computes $x_2 = B_1 u_1$ of dimension η_2 and $y_1 = D_1 u_1$ of dimension n_1, using some matrices B_1 and D_1 of appropriate dimensions. Then it moves to index 2, takes in u_2 of dimension m_2, computes $x_3 = A_2x_2 + B_2u_2$ of dimension η_3 and $y_2 = C_2x_2 + D_2u_2$ of dimension n_2. Next, it moves to index 3, takes in u_3 and computes x_4 and y_3 etc... up to a last index ℓ, whereby the last computation ends with an empty state $x_{\ell+1}$ and $y_\ell = C_\ell x_\ell + D_\ell u_\ell$.

What would be the *behavior*, i.e., the overall map such a system has computed? Clearly, each state x_k remains internal to the system, for an external observer, only the input-output map, called the functional behavior would matter. We obtain this behavior when we eliminate the state. E.g., we have $y_2 = C_2x_2 + D_2u_2$ and hence $y_2 = C_2B_1u_1 + D_2u_2$, and likewise $y_3 = C_3A_2B_1u_1 + C_3B_2u_2 + D_3u_3$ etc..., summarizing in matrix form:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \end{bmatrix} = \begin{bmatrix} D_1 & 0 & 0 & \cdots \\ C_2B_1 & D_2 & 0 & \ddots \\ C_3A_2B_1 & C_3B_2 & D_3 & \ddots \\ \vdots & \ddots & \ddots & \ddots \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \end{bmatrix}. \tag{5.2}$$

The matrix in the middle, let us call it T, has dimensions $(n_1 + n_2 + n_3 + \cdots) \times (m_1 + m_2 + m_3 + \cdots)$. It is a matrix consisting of subblocks $T_{i,j}$ with dimensions $n_i \times m_j$. It is also lower block triangular, reflecting the fact that an output y_i is only dependent on inputs u_j with $j \leq i$. Its general element, assuming $j < i - 1$ is $C_iA_{i-1} \cdots A_{j+1}B_j$, in the literature sometimes denoted by $C_iA^{>}_{i,j}B_j$, where $A^{>}_{i,j}$ is the *continuing product* $A_{i-1} \cdots A_{j+1}$ for $i > j$ and with the convention that $A^{>}_{i,i} = I$. The regularity in the construction of the entries is pretty apparent, so it will prove useful to introduce a more compact notation that summarizes the essentials.

To start with this, let us first look at vectors. If we do not want to see indices in detail, we put $u := \mathrm{col}(u_j)_{j=1}^{n}$. But why restrict ourselves to starting times at index 1 or stopping times at n? If we do not want to bother about the starting time, we can just pad u with empty entries for all index points less than one and larger than n. Let us put $u_j := -$ (i.e., *dash*) for all $j < 1$ and $j > n$, and we can write $u := \mathrm{col}(u_j)$ where the index range from $-\infty$ to $+\infty$ is simply understood. The entry annotated by '$-$' has dimensions 0×1, and it is compatible with the typical dimension of a vector, e.g., $m \times 1$, now with $m = 0$. One can always think of such dummies stacked on top and at the bottom of a vector. They play the role of "place holders", with the meaning: "there is an index, but no entry". Such place holders

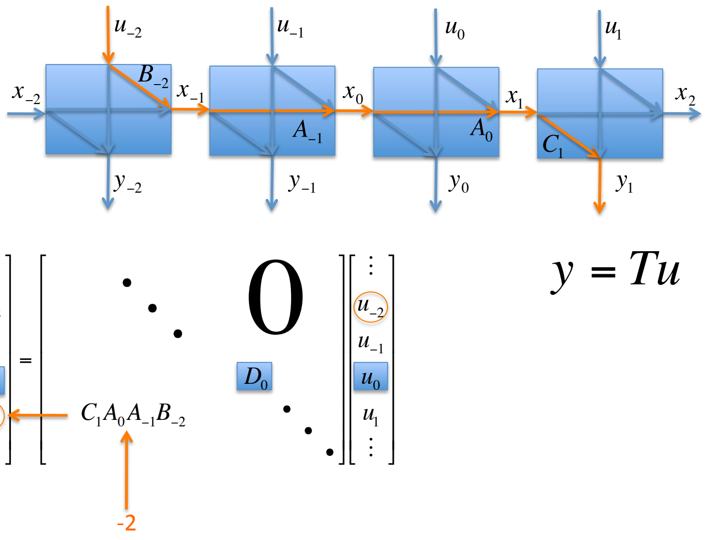


Figure 5.2: Example of an overall causal system build from local computations.

play an essential role in computer languages (in that context they are often denoted by a "perp" ($\perp$)), and they arise e.g., when a stack becomes empty after processing, or, as here, when at some indices in an indexed list there is no entry. In our case, the definition should produce elements that are dimensionally compatible with other elements in the same vector.

Likewise, we could have a case where at some index points there are no input entries, while there is still a computation going on, or where the computation does not produce an output entry. So it makes sense to generalize our notion of 'empty entry' further. We may have a matrix element of dimension 1×0, which we might indicate by a vertical dash ($|$), an element of dimensions 0×0 that we annotate with a simple dot $(\cdot)$ and all these may be stacked as well in vectors, provided dimensions remain compatible. Clearly, an extension of matrix calculus is needed to allow us to do computations with such elements. Here are a few simple rules that keep everything compatible. Let us use the "$*$" to represent matrix multiplication explicitly, and a any number:

operation	result
$- * \vert$	$\cdot$
$\vert * -$	$[0]$
$\cdot * -$	$-$
$\vert * \cdot$	$\vert$
$\cdot * \vert$	illegal
$- * [a]$	$-$
$[a] * \vert$	$\vert$
$\begin{bmatrix} - & - \end{bmatrix} \begin{bmatrix} \vert \\ \vert \end{bmatrix}$	$\cdot$
etc...	

(Notice that we do not distinguish between | and [|] etc..., for brevity—brackets are just "sugar coating".)

With such rules we can still write for any input-output map $y = Tu$, in which all vectors and matrices are properly padded, and zero dimensions are allowed for the inputs, the states and the outputs, while all time series run from $-\infty$ to $+\infty$.

However, there is one issue with this approach that should already be mentioned here: as soon as vectors and matrices are allowed to have infinite indices, the matrix-vector product or matrix-matrix product may produce infinite sums that do not converge. We shall discuss in further chapters how one can deal with such a situation, let us at this point assume that either it does not occur (because all summations are de facto finite), or that convergence is properly taken care of. The fact that systems evolve with unconstrained times has great technical consequences. So it makes sense to bring in that fact from the start. Another point of technical importance is that of orientation in an infinitely indexed vector or matrix. For this we single out the element at index 0 and surround it with box. Hence we have $u = \mathrm{col}(\cdots, u_{-1}, \boxed{u_0}, u_1, \cdots)$ etc... (this element may of course be empty.).

The next step is to represent the various actions in the system in a compact fashion. We start by remarking that each elementary computation takes place at a specific index point. Let us explore the global consequence of this simple fact. Concatenating the states, let's simply write

$$x := \mathrm{col}(\cdots, x_{-1}, \boxed{x_0}, x_1, \cdots), \tag{5.3}$$

then we observe that the operators A_k together not only map x to a backward shifted version of itself (namely $\mathrm{col}(\cdots, x_0, \boxed{x_1}, x_2, \cdots)$), but also so that each x_k only maps to its x_{k+1} and nothing else. To capture this globally, we define a *forward or causal shift operator* Z: given any infinitely indexed vector (say x), let Zx be defined as the vector $Zx := \mathrm{col}(\cdots, x_{-2}, \boxed{x_{-1}}, x_0, \cdots)$, i.e., it just postpones x_k one index point. This operator admits of course an inverse, the *backward or anti-causal shift*, which we write as Z^{-1}.

For the operators A_k that are just locally active, we next define a (doubly infinitely indexed) block diagonal matrix, using a constructor that produces a diagonal matrix from its arguments, $A := \mathrm{diag}(\cdots, A_{-1}, \boxed{A_0}, A_1, \cdots)$. Similarly, $B := \mathrm{diag}(B_k)_{k=-\infty}^{+\infty}$, $C := \mathrm{diag}(C_k)_{k=-\infty}^{+\infty}$ and $D := \mathrm{diag}(D_k)_{k=-\infty}^{+\infty}$ will globally characterize the input-to-state, state-to-output and feed-through maps that act locally at each index point, and the global state-space equations become

$$\left\{ \begin{array}{rcl} x & = & Z(Ax + Bu) \\ y & = & Cx + Du \end{array} \right. \tag{5.4}$$

This is probably as compact as one can get these equations.Whether such a compact representation is indeed useful has to be demonstrated, and we hope that the rest of the book will be convincing. It surely allows us to derive an equally compact representation for the state of the system, at least formally. Eliminating the state x, we obtain in sequence:

$$x = (I - ZA)^{-1}ZBu \tag{5.5}$$

and $y = Tu$ with

$$T = D + C(I - ZA)^{-1}ZB. \tag{5.6}$$

Very well, but what is $(I - ZA)^{-1}$?

5.3 Uniform exponential stability

Now the issue has become: is inverting $(I - ZA)^{-1}$ meaningful and what is it then? Remarkably, there are several answers possible to this question, each with its own significance (and history). First of all, suppose that the sequence x is known to begin somewhere (e.g., at index 1 in our previous example and is empty for indices less that 1), then one could write $w := (I - ZA)^{-1}x = (I + ZA + ZAZA + (ZA)^3 + \cdots)x$, remarking that the term $(ZA)^k x$ involves k subsequent forward shifts, since of course $[(ZA)^k x]_j = A_{j-1} \cdots A_{j-k+1} A_{j-k} x_{j-k}$, because $[(ZA)^k x]_j = [A(ZA)^{k-1}]_{j-1} = A_{j-1}[(ZA)^{k-1}]_{j-1}$ etc. Now fix some j and look at the sum of all the terms in the j^{th} component of the sum. When $j = 1$, only one term is non-trivial:

the first one $w_1 = x_1$. When $j = 2$, we find $w_2 = x_2 + A_1x_1$, and then $w_3 = x_3 + A_2x_2 + A_2A_1x_1$ etc... and when $j < 1$, w_j is empty. The sum contains less than j terms for every element in $w = (I - ZA)^{-1}x$ of index less than j. When $j \leq k$, then x_{j-k} is empty ($= -$), and hence the whole product is empty as well. This actually reflects the fact that the state space equation is a forward recursion, whereby x_k is only dependent on previous values, which is just a finite set. However, the dependence grows with growing k, and so it may happen that for larger k's, these entries blow up. E.g., suppose all $A_k = 2$ for $k \geq 1$, then one would have $[(ZA)^kx]_j = 2^{j-k}x_{j-k}$ for all $j > k$.

It is not difficult to write down the matrix for $(I - ZA)^{-1}$ for the $n \times m$ block matrix we started out with in this chapter. It is

$$\begin{bmatrix} \boxed{\cdot} & - & - & - & \cdots & \cdots \\ | & I & 0 & 0 & \cdots & \cdots \\ | & A_2 & I & 0 & 0 & \ddots \\ | & A_3A_2 & A_3 & I & 0 & \ddots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots \\ | & A_{n-1}A_{n-2}\cdots A_2 & \cdots & A_{n-1}A_{n-2} & A_{n-1} & I \end{bmatrix} \tag{5.7}$$

and it has dimensions $(\eta_1 + \eta_2 + \cdots + \eta_{n-1})^2$ (it is a square matrix, with unit matrices on the main diagonal.). We see that if the A_k's are too large, elements further away from the main diagonal might blow up even exponentially, making the resulting matrix numerically unstable. Such an instability must often be avoided. Although $(I - ZA)^{-1}$ remains well defined in the positive one-sided case just described and leads to finite arithmetic, it is often necessary to require boundedness[2]. This can be done as described next, where we consider the more general case that the system's indices extend potentially to either $-\infty$, $+\infty$, or both, as would e.g., always be the case for time-invariant systems.

One way to impose some modus of stability, at least in the infinitely indexed case (but also often needed for finite matrices), is to assume that there exists a positive number $\sigma < 1$, such that for any (small) positive ϵ

[2]Systems that have to be controlled can be unstable, the purpose of the controller being to achieve stability. For such problems, unstable system descriptions are necessary. On the other hand, instabilities in data processing applications have to be avoided, making a different approach necessary. We shall analyze this issue further in chapter 9.

with $\sigma+\epsilon<1$, there is an ℓ such that for all k (i.e., uniformly), $\|A^{>}_{k+\ell+1,k}\|<(\sigma+\epsilon)^{\ell}$, where $\|M\|$ is by definition the Euclidean norm of any matrix M. The smallest such σ is called the *spectral radius* of ZA. When such a property holds, then A is said to be *uniformly exponentially stable*, which amounts to say that the entries in the matrix $(I-AZ)^{-1}$ *eventually* die out exponentially with rate σ uniformly, when you move away sufficiently far from the main diagonal. This condition is of course automatically fulfilled when $(I-AZ)$ is a finite matrix, since a lower triangular block matrix with units on the main diagonal will always be invertible, but the result may still have an undesirably large condition number, a situation that then would have to be dealt with in each concrete case. We shall analyze this crucial situation in detail in chapter 10.

5.4 Diagonal shifts

Block diagonal matrices, responsible for local calculations, play a major role in the theory of LTV systems, much like constant matrices do in the theory of linear time-invariant systems. We introduce therefore a special type of shift for them. Suppose A is such a matrix, then consider ZAZ^{-1}. The left shift operator Z will shift the rows of A one notch down, while the right shift operator Z^{-1} will shift the columns one notch to the right. The net result is a shift along the main diagonal in the South-East direction. We write $A^{<+1>}:=ZAZ^{-1}$. Similarly, a diagonal shift in the North-West direction is $A^{<-1>}:=Z^{-1}AZ$. Equivalently, we have $ZA=A^{<+1>}Z$ and $AZ=ZA^{<-1>}$. Although Z does not commute with other operators, there is a kind of weak commutativity, involving a shifted version. As we shall see in further chapters, this is enough for most of LTV system theory to generalize what happens in the time-invariant theory. It also follows that $(ZA)^{\ell}=A^{<+1>}A^{<+2>}\cdots A^{<+\ell>}Z^{\ell}=Z^{\ell}A^{<-(\ell-1)>}\cdots A^{<-1>}A$: a global expression for the *continuous product*.

5.5 Anti-causal Systems

We call lower triangular block matrices *causal*. A state x_k or output y_k of a system whose transfer operator is (block) lower triangular depends only on inputs in the past up to index k (i.e., a causal system). Dually, we can consider systems that run backward in time, where y_k would only be dependent on inputs u_j with $j \geq k$. Consider the transpose $Z' = Z^{-1}$ of Z: it defines a backward shift on vectors. Now consider the backwards running system

$$\left\{ \begin{array}{lcl} x_{k-1} & = & A_k x_k + B_k u_k \\ y_k & = & C_k x_k + D_k u_k \end{array} \right. \tag{5.8}$$

resulting, similarly as before, in an *anti-causal* transfer operator $T = D + C(I - Z'A)^{-1}Z'B$. The corresponding matrix will then be upper-triangular, in the case of a traditional (block) matrix with a state space description starting at index n and running backward to 1 it looks as follows:

$$\begin{bmatrix} D_1 & C_1B_2 & C_1A_2B_3 & \cdots & C_1A_2\cdots A_{n-1}B_n \\ 0 & D_2 & C_2B_3 & \ddots & C_2A_3\cdots A_{n-1}B_n \\ 0 & 0 & D_3 & \ddots & \vdots \\ \ddots & \ddots & \ddots & \ddots & \vdots \\ \cdots & \cdots & \cdots & 0 & D_n \end{bmatrix} \tag{5.9}$$

with general term $T_{i,j} = C_iA_{i+1}\cdots A_{j-1}B_j$ for $i + 1 < j$. The notion of uniform exponential stability applies as before.

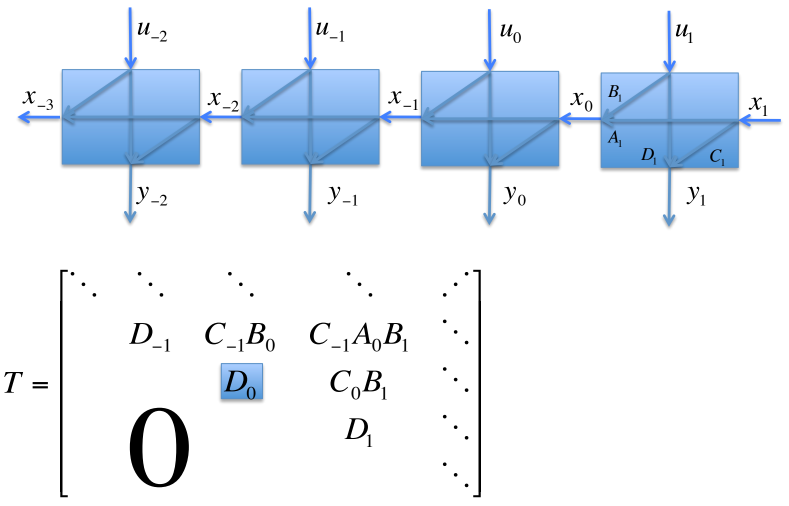


Figure 5.3: Schema for a general anti-causal LTV system.

5.6 Duality

Given a causal state space system $T = D + C(I - ZA)^{-1}ZB$, and using the regular matrix transposition ($'$), we may define the dual or transposed system $T' = D' + B'Z'(I - A'Z')^{-1}C' = D' + B'(I - Z'A')^{-1}Z'C'$. We see that the causal transfer operator $\begin{bmatrix} A & B \\ C & D \end{bmatrix}$ just changes into the anti-causal transfer operator $\begin{bmatrix} A' & C' \\ B' & D' \end{bmatrix}$. Time is reversed, the role of inputs and outputs are interchanged, the incoming state x_k comes out of the previous step which is now located at index $k+1$ and the outgoing state will feed into stage $k-1$. Although this convention may seem a bit strange at first (the k^{th} state x_k is at a different location in the causal than in the anti-causal case), it turns out to be consistent with the normal indexing in the input-output matrices. (It may be remarked that other conventions can be found in the literature, in particular x_k can be taken as the output state in the causal filter, which then takes x_{k-1} as its input. This is of course immaterial, the convention adopted here seems the most convenient.)

5.7 Example: a simple banded matrix and its inverse.

Let's find a simple realization for

$$T = \begin{bmatrix} d_1 & 0 & 0 & 0 \\ a_2 & d_2 & 0 & 0 \\ 0 & a_3 & d_3 & 0 \\ 0 & 0 & a_4 & d_4 \end{bmatrix} \tag{5.10}$$

Let's retain the traditional matrix indexing scheme (i.e., starting at index 1). It follows that stage 0 is empty. At stage 1 we take in u_1 and immediately produce $y_1 = d_1u_1$. Next we have the choice to either immediately compute a_2u_1, or take u_1 as a state and feed it to the next stage. Let's take that latter path and put $x_2 = u_1$. Hence $\begin{bmatrix} A_1 & B_1 \\ C_1 & D_1 \end{bmatrix} = \begin{bmatrix} | & 1 \\ | & d_1 \end{bmatrix}$. The next stage now takes in x_2 and u_2, and we compute immediately $y_2 = a_2x_2 + d_2u_2$. At this point, $u_1 = x_2$ is not needed anymore further on and can be discarded, but u_2 will be needed in the next computation. Hence we put $x_3 = u_2$ and have

$\begin{bmatrix} A_2 & B_2 \\ C_2 & D_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ a_2 & d_2 \end{bmatrix}$. The next stage is not much different and we find $\begin{bmatrix} A_3 & B_3 \\ C_3 & D_3 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ a_3 & d_3 \end{bmatrix}$. Finally, the fourth stage takes in $x_4 = u_3$ and u_4, and computes $y_4 = a_4 x_4 + d_4 u_4$ so that $\begin{bmatrix} A_4 & B_4 \\ C_4 & D_4 \end{bmatrix} = \begin{bmatrix} - & - \\ a_4 & d_4 \end{bmatrix}$. The following steps are empty again.

Anticipating the section on matrix inversion, let us observe that if all the D_i's are invertible, one obtains an easy realization of the inverse of a system (forgetting about stability), by 'arrow reversal': reversing the feed-through arrow from input to output (see fig. 5.4). This procedure can be applied

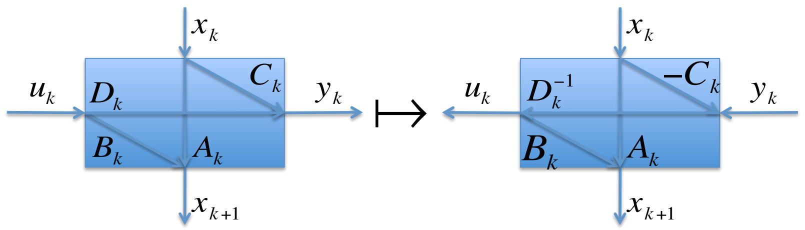


Figure 5.4: Finding a formal (potentially unstable) inverse by arrow reversal.

directly on the state space description:

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \mapsto \begin{bmatrix} A - BD^{-1}C & BD^{-1} \\ -D^{-1}C & D^{-1} \end{bmatrix} \tag{5.11}$$

from which a realization for the inverse is easily found, in sequence: $\begin{bmatrix} | & d_1^{-1} \\ | & d_1^{-1} \end{bmatrix}$, $\begin{bmatrix} -d_2^{-1}a_2 & d_2^{-1} \\ -d_2^{-1}a_2 & d_2^{-1} \end{bmatrix}$, $\begin{bmatrix} -d_3^{-1}a_3 & d_3^{-1} \\ -d_3^{-1}a_3 & d_3^{-1} \end{bmatrix}$, $\begin{bmatrix} - & - \\ -d_4^{-1}a_4 & d_4^{-1} \end{bmatrix}$. Not only does this allow for a quick and easy calculation of the inverse matrix (which would actually not be too hard by back-substitution), it shows that the inverse system has a realization whose state dimension equals the original, hence can perform computations (solving equations) with the same computational complexity as the original, although the matrix now has a full triangular lower part.

5.8 Block matrices and the diagonal notation.

Suppose A, B, C, D are (block) diagonal matrices representing a realization, then we could write down a block matrix $\begin{bmatrix} A & B \\ C & D \end{bmatrix}$ having these block-diagonal matrices as components. More graphically, this representation looks as follows:

$$\left[\begin{array}{ccccc|ccccc} \ddots &&&&& \ddots &&&& \\ & A_{-1} &&&&& B_{-1} &&& \\ && \boxed{A_0} &&&&& \boxed{B_0} && \\ &&& A_1 &&&&& B_1 & \\ &&&& \ddots &&&&& \ddots \\ \hline \ddots &&&&& \ddots &&&& \\ & C_{-1} &&&&& D_{-1} &&& \\ && \boxed{C_0} &&&&& \boxed{D_0} && \\ &&& C_1 &&&&& D_1 & \\ &&&& \ddots &&&&& \ddots \end{array}\right] \tag{5.12}$$

Alternatively, one could write the same realization as a single block diagonal consisting of local block matrices, as follows:

$$\begin{bmatrix} \ddots &&&& \\ & \begin{bmatrix} A_{-1} & B_{-1} \\ C_{-1} & D_{-1} \end{bmatrix} &&& \\ && \boxed{\begin{bmatrix} A_0 & B_0 \\ C_0 & D_0 \end{bmatrix}} && \\ &&& \begin{bmatrix} A_1 & B_1 \\ C_1 & D_1 \end{bmatrix} & \\ &&&& \ddots \end{bmatrix} \tag{5.13}$$

Both representations are, of course, fully equivalent: via an obvious permutation of appropriate rows and columns. One could develop a formal equivalence theory for such structures, but we shall not do so, assuming that the type of representation used will always be clear from the context, and thereby avoiding an unnecessary cluttering of symbols. How this type of liberty functions is seen in the following, hopefully obvious example; when e.g.,

$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$ is a matrix of block diagonals, then a correct expression for ZA is:

$$ZA = \begin{bmatrix} ZA_{11} & ZA_{12} \\ ZA_{21} & ZA_{22} \end{bmatrix} \tag{5.14}$$

because "Z" is nothing else but a shift operator, which applies equally on the indices of all rows (when applied left) or on the indices of all the columns (when applied right). One could actually state the formal equivalence

$$\begin{bmatrix} \mathrm{diag}A_k & \mathrm{diag}B_k \\ \mathrm{diag}C_k & \mathrm{diag}D_k \end{bmatrix} \equiv \mathrm{diag} \begin{bmatrix} A_k & B_k \\ C_k & D_k \end{bmatrix} \tag{5.15}$$

showing that the matrix constructor commutes in a sense with the diagonal constructor (provided dimensions of the arguments agree, of course). However, the matrices both sides of this expression have a different order.

5.9 Example: the realization of a series.

With the ideas just expounded, it is easy to generalize the previous example and produce realizations in 'companion form' as in the classical case. Consider the global realization

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \left[\begin{array}{ccc|c} 0 & I & 0 & 0 \\ 0 & 0 & I & 0 \\ 0 & 0 & 0 & I \\ \hline T_3 & T_2 & T_1 & T_0 \end{array}\right] \tag{5.16}$$

in which the "I's" are (finite or infinite) unit matrices, the "0's" (finite or infinite) zero matrices and the T_i block diagonals, all with matching dimensions as needed by the matrix, of course. Then this realizes $T = T_0 + T_1 Z + T_2 Z^2 + T_3 Z^3$ as is easily verified from

$$(I - ZA)^{-1} ZB = \begin{bmatrix} I & -Z & 0 \\ 0 & I & -Z \\ 0 & 0 & I \end{bmatrix}^{-1} ZB = \begin{bmatrix} Z & Z^2 & Z^3 \\ 0 & Z & Z^2 \\ 0 & 0 & Z \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ I \end{bmatrix}. \tag{5.17}$$

The method generalizes easily further to larger expansions. In the case of the smaller Example 1, we have:

$$T = \begin{bmatrix} d_1 & & & \\ & d_2 & & \\ & & d_3 & \\ & & & d_4 \end{bmatrix} + \begin{bmatrix} | & & & \\ & a_2 & & \\ & & a_3 & \\ & & & a_4 \end{bmatrix} \begin{bmatrix} - & & & \\ 1 & 0 & & \\ & 1 & 0 & \\ & & 1 & 0 \end{bmatrix} \tag{5.18}$$

and the realization

$$\left[\begin{array}{cccc|cccc} | & & & & 1 & & & \\ & 0 & & & & 1 & & \\ & & 0 & & & & 1 & \\ & & & - & & & & - \\ \hline | & & & & d_1 & & & \\ & a_2 & & & & d_2 & & \\ & & a_3 & & & & d_3 & \\ & & & a_4 & & & & d_4 \end{array}\right] \tag{5.19}$$

(it is worthwhile checking this out carefully!)

5.10 Discrete time, linear, time-invariant systems*

A time-invariant system runs from arbitrarily small negative indices to arbitrary large positive ones—it is by definition an infinitely indexed system. That mere fact makes a number of issues connected to such systems rather peculiar. Historically, such systems were studied first, and it was soon thought that time-variant systems would be more complicated, which, in the light of modern time-variant theories appears not to be true at all. Some attempts to generalize results for time-invariant systems to the time-variant case failed miserably (we shall discuss some in further chapters), and it took a long time to understand that the algebraic principles underlying general, time-variant dynamical systems were at the same time simpler, more general and more intuitive than in the LTI case. Much is somewhat similar to the move from continuous-time to discrete-time systems, and in particular from the Laplace transform to the z-transform. Discrete-time systems are basically much simpler than continuous-time systems. The modern approach that we advocate turns things around: one discusses the discrete-time time-variant case first and then moves on to continuous-time and time invariance.

Concerning the notion of time invariance, one may adopt an internal and an external or behavioral view. The internal view is that none of the constituents of the system change with time. If the system has a 'natural' state space description, time invariance would mean that this description is the same at all times. In the case of discrete-time, linear systems with finite dimensional states, that means that the same transition maps $\begin{bmatrix} A & B \\ C & D \end{bmatrix}$ is valid for all k running from $-\infty$ to $+\infty$. In that case, the corresponding global block diagonal matrices A, B, C and D are diagonal *Toeplitz*, meaning that their diagonal entries all have the same value. In that case, the shift Z commutes with the operators and can simply be written as a "scalar" z, as we have $zA = Az$. The time-invariant transfer transfer function can then be written as

$$T = D + C(I - zA)^{-1}zB. \tag{5.20}$$

At this point, z is just a forward shift (in much of the engineering literature, the forward shift is written z^{-1}, with z the backward shift. We do not follow that convention, mainly to avoid formulas with lots of z^{-1}'s—which of course look very impressive.). The next step in the analysis of such systems is to interpret z as a complex variable, and to drop the interpretation of A as a block diagonal Toeplitz matrix, replacing it simply with the original transition matrix A (and similarly for the other matrices B, etc.)[3]. This is useful, because then $(I - zA)^{-1}$ can be interpreted as a rational matrix:

$$(I - zA)^{-1} = \frac{1}{\zeta_A(z)} M_A(z) \tag{5.21}$$

in which $\zeta_A(z) = \det(I - zA)$ and $M_A(z)$ is the matrix of minors of $(I - zA)$ (if $\chi_A(\lambda) = \det(\lambda - A)$ is the characteristic polynomial of A of dimension $\eta \times \eta$, then $\zeta_A(z) = z^\eta \chi_A(z^{-1})$. $\zeta_A(z)$ is a polynomial of degree η, and $M_A(z)$ is a matrix of polynomials of degree at most $\eta - 1$. Hence the transfer function becomes

$$T(z) = \frac{1}{\zeta_A(z)} [D\zeta_A(z) + zCM_A(z)B] \tag{5.22}$$

[3]Although we shall not pursue this path formally, there is an 'isomorphism' involved here, where a calculus of (doubly infinitely indexed) Toeplitz matrices are made to correspond with infinite matrix series in the variable z. E.g., the input-output causal Toeplitz operator T corresponds to the *transfer function* $T(z) = T_0 + zT_1 + z^2T_2 + \cdots$. See the appendix, chapter 14 for further information.

Much of discrete-time LTI system theory consists in studying this expression, using various properties from linear algebra and analysis. In particular, the zeros of $\zeta_A(z)$ are the poles of $T(z)$ and $T(z)$ will be 'stable', if all these lie strictly outside the unit disc $\mathbf{D}$ of the complex plane. It may be seen that this condition is equivalent to the more general condition of 'uniform exponential stability' that we discussed earlier. Suppose that $a \in \mathbf{D}$ with $a \neq 0$ is an eigenvalue of A, then $1/a$ is a pole of $T(z)$, and it lies outside the unit disc. It would be called a 'stable pole'. Suppose that $a \neq 0$ is an eigenvalue of order ℓ of A, then one can show that the corresponding time series decays as $\ell k|a|^k$ for indices k running to $+\infty$ when this pole is excited, which will always be majorized eventually by $(|a| + \epsilon)^k$ for any arbitrarily small $\epsilon > 0$. One way to study these phenomena is to convert the state transition matrix A to a Jordan canonically form; in further chapters we shall hint at some of those. In particular, when all the eigenvalues of A are zero, we shall have $\zeta_A(z) = I$, A nilpotent, and $T(z)$ purely polynomial, called a 'moving average filter'.

Conversely, suppose that a is an eigenvalue of A with $|a| > 1$, and that the corresponding pole $1/a$ is not cancelled out by all entries in $M_A(z)$, then the system will be guaranteed unstable. A somewhat dubious case arises when the eigenvalue a has modulus $|a| = 1$, resulting in a kind of borderline stability, at least when the eigenvalue is single (when the eigenvalue is multiple it is unstable). However, there are good reasons to even call the case of a single boundary eigenvalue unstable, which is what is usually done in the literature.

5.11 Discussion issues

- Given two LTV systems, one could combine them in various ways. Since each represents an input-output operator, one could e.g. add them when they have the same inputs and outputs ($T = T_1 + T_2$) or multiply them, in case the output of one can be taken as input to the next ($T = T_2T_1$). Suppose each has an appropriate realization, what would be a realization of the sum or the product? One could, of course, consider more general, networked cases, how could a more general theory look like formally?

- The various construction mechanisms for block- and block-diagonal matrices seem to be only partially compatible with each other. Would there be a more logical system, that is at the same time simple, MATLAB-like and allows for arbitrary, matrix-compatible constructs?

- With respect to the computation of $(I - ZA)^{-1}$ in section 5.3 and example 1: an easy way to see how signals are put together (in particular the state at a certain index) is by drawing the data flow diagram, much in the taste of figs. 5.2 and 5.3. You can easily check the correctness of the formulas given that way. Which properties should such a diagram have so that it corresponds to an executable computation?

- Stability as we defined it, whether u.e.s. in the time-variant case of the location of poles of the transfer function, is based on the resulting evolution of the state or the response of the system when time increases. Or, to put it differently: it describes whether the effect of a single disturbance dies down when the system progresses in time. Another notion that is often used is 'BIBO-stability', or 'Bounded Input Bounded Output stability'. This notion requires the definition of a norm in the spaces of input and output time-series, often taken to be an overall quadratic norm. BIBO-stability then means that the map from input space to output space induced by the transfer function is bounded. Would there be some relationship between the various notions?

5.12 Notes

- The large interest in linear time-variant systems started most likely with the ground breaking work of R.E. Kalman on state estimation theory, in the early sixties of the past century [25]. For a nice overview of much of the early work, see [21]. The gist of the new movement in system theory was the introduction of state space descriptions, so that system properties could be studied intimately via the properties of the evolution of the state rather than purely from an input-output point of view, as was the standard until then. Notions such as reachability, controllability and observability became the central concepts. They allowed the development of new approaches not only to estimation theory, but also to control theory and network theory. Even though many of these were very successful, a big gap remained between the properties of time-variant and time-invariant systems, because in the latter, full use could be made of transform theory via the properties of poles and zeros of the transfer function or the eigenvalues of the state-transition matrix. Only in the beginning of the 90's a bridging concept was discovered that allowed many, if not most of the properties of time-invariant systems to be generalized. The present chapter introduces that concept for the case of discrete-time systems, namely the globalization of the system description via instantaneous diagonal operators and the shift operator Z. In the remainder of this book we shall study the emerging basic concepts in detail. We just mention at this point that 'inner-outer' factorization will turn out to be the central method. From a mathematical point of view, inner-outer factorization is exemplary of what happens in what has been termed 'nest algebras' by Ringrose [30] and Arveson [4], which provides for the basic theoretical framework common to both time-invariant and time-variant systems. The case of discrete time, time-variant systems using diagonal calculus is a specialization of the nest algebra approach that makes the treatment of time-variant and time-invariant look very much alike, and was first proposed in [11], see [10] for a full account.

- One of the salient features of using diagonals as basic entities, is that (block-)diagonals act as the 'scalars' or basic building blocks of the theory, very much like real or complex numbers do in elementary one-input one-output system theory, or $m \times n$ real or complex matrices in multiport theory, with m the dimension of the input vectors and n that of the output vectors. In other words: only the character of the basic state operators $\{A, B, C, D\}$ changes, but not, in a large part, their algebra. They do not commute any more with shifts, but the structure remains rich enough to allow for most if not all the basic notions and operations. This will become apparent in the following chapters and is also the main motivation why the theory is developed in this way.

Chapter 6

System identification

In chapter 5 we discovered matrix representations for the input-output behavior of a causal dynamical systems for which there is a state space realization—at least in the discrete time, linear case. System identification takes the opposite path: it starts from an input-output description, and then goes on to figure out a state space realization of a causal dynamical system that would produce the specified behavior. This is in general a tricky undertaking in general, like trying to discover what is inside a black box from experiments outside. Luckily, some a priori structural knowledge about the system helps. For example, that it is a discrete-time and linear system, the case we consider here. Even in this already restricted class, we shall distinguish several different but related cases: the finite matrix case (the simplest one), the time-invariant case and the general linear discrete-time and time-variant case. Further cases, beyond discrete time LTV are relegated to later chapters.

In this chapter we work primarily on causal systems—the anti-causal case being dual and hence equivalent algebraically. Hence, we assume the transfer matrices describing the input-output behavior to be block lower triangular for the finite matrix case, or, in the LTI-case and infinitely indexed case, to be stable as well as causal. It soon turns out that for systems with relatively small state space description, the full input-output description is highly redundant, and the identification can be done with a limited amount of well-chosen data. In another direction, we could try to match limited system descriptions to a limited number of experiments, a point that we shall only touch. The procedures generalize to mixed causal-anticausal systems as well.

Menu

Hors d'oeuvre
The matrix case

Intermezzo
Specializing to time-invariance*

Main course
Realizations with partial data

Dessert
The case with infinite indices

6.1 The matrix case

We assume that we are given a lower block-triangular matrix T, and a realization (a state space description) for it is desired. From chapter 5, we already know that just finding a non-minimal realization is relatively easy: just decompose T in its block-diagonals and a standard 'companion form' realization would follow. This is not really what is desired, because the dimensions of such a realization quickly gets out of bounds. From a numerical point of view, the whole attraction of using state space descriptions is that many systems possess a low order system description, which can be used to execute efficient computations, e.g., for matrix-vector multiplication or system inversion—to be treated in following chapters. Hence, what is needed is a realization that is as small as possible in the dimensions of the subsequent state spaces. We shall soon see that there are indeed realizations with the smallest possible dimension at each index point, and that these are uniquely determined by the input-output matrix—a very strong result.

At this point, our basic insights from Chapter 3 come to the rescue. The state of the system at a given index k is *what it has to know from the past, to allow precise determination of the future evolution given future inputs*—see figure (3.1). To put it differently: two past inputs may produce the same state when the observation of any future system evolution cannot distinguish between them, using new test inputs (we then say: these inputs are 'Nerode equivalent'.). In the case of linear systems (we shall discuss non-linear systems in a later chapter), the situation greatly simplifies, at least algebraically (it turns out that a clean and general non-linear theory is possible, but gets to be too abstract for a first approach.). The reason for this is that in the case of a linear system, testing with any future input turns out to be as good as with any other, and in particular with the zero input: we shall soon see that if one specific future input is able to distinguish between two different states, then any other would do it as well.

From the definition of 'state', it follows that the state is what connects the past of the system to its future. Algebraically, this is best characterized by what we have defined as the *Hankel map*, namely the map that maps strict past inputs to future outputs, with zero as the future input (since, as we shall soon see, any other input would be just as good). Let's make this precise. Consistent with our notation so far and positioning ourselves at some index point k, we can subdivide any input sequence $u = u_{p_k} + u_{f_k}$ where u_{p_k} is the input up to and including index $k-1$, and u_{f_k} the input from k on up to infinity. Hence, the subdivision w.r. to index k divides the input into its strict past and its future, whereby the 'present' k is included in the future. We can of course also define a projection operator π_{k-} so that $u_{p_k} = \pi_{k-}u$ and $u_{f_k} = (I - \pi_{k-})u$. Likewise for y. Doing so will decompose the (causal) input-output operator as follows:

$$y = Tu \mapsto \begin{bmatrix} y_{p_k} \\ y_{f_k} \end{bmatrix} = \begin{bmatrix} T_{k-} & 0 \\ H_k & T_{k+} \end{bmatrix} \begin{bmatrix} u_{p_k} \\ u_{f_k} \end{bmatrix} \tag{6.1}$$

In MATLAB notation[1] we have $T_{k-} := T_{:(k-1),:(k-1)}$, $H_k := T_{k:,:(k-1)}$ and $T_{k+} := T_{k:,k:}$ see fig. 6.1. T_{k-} maps strict past to strict past, T_{k+} maps future (including present) to future (including present), while the crucial Hankel operator H_k maps strict past to future at index point k.

From the relation $y_{f_k} = H_k u_{p_k} + T_{k+}u_{f_k}$, we see that two inputs in the past will produce the same output in the future when their difference belongs to the kernel of H_k, no matter what the future input u_{f_k} is. Conversely, if, for some future input, two past inputs produce different future outputs, then

[1]In MATLAB, ranges of indices are indicated as follows: '1:5' means 'from 1 to 5 inclusive', '1:' runs from 1 to infinity, and multiple ranges are separated by comma's. Hence, a doubly infinite matrix with rows running from $-\infty$ to -1 and columns from 0 to ∞ has ranges ': $-1, 0$:'.

$$T = \begin{bmatrix} T_{k-} & 0 \\ H_k & T_{k+} \end{bmatrix} = \left[\begin{array}{cccc|cccc} \ddots & & & & & & & \\ \ddots & T_{k-3,k-3} & & & & & & \\ \ddots & T_{k-2,k-3} & T_{k-2,k-2} & & & \mathbf{0} & & \\ \cdots & T_{k-1,k-3} & T_{k-1,k-2} & T_{k-1,k-1} & & & & \\ \hline \cdots & T_{k,k-3} & T_{k,k-2} & T_{k,k-1} & T_{k,k} & & & \\ \ddots & T_{k+1,k-3} & T_{k+1,k-2} & T_{k+1,k-1} & T_{k+1,k} & T_{k+1,k+1} & & \\ \ddots & T_{k+2,k-3} & T_{k+2,k-2} & T_{k+2,k-1} & T_{k+2,k} & T_{k+2,k+1} & T_{k+2,k+2} & \\ \iddots & \ddots & \ddots & \vdots & \vdots & \ddots & \ddots & \ddots \end{array}\right]$$

Figure 6.1: The decomposition of T at index point k into strict past and future.

the difference between these two past inputs cannot be in the kernel of H_k. It follows that the matrix rank of H_k has to be the minimal state dimension. We show now constructively that this is indeed the case, and in the process obtain a realization for T at index k.

Our strategy is now as follows: first we suppose that we have indeed found a realization $\{A_k, B_k, C_k, D_k\}$ at every index k for each H_k and look at the consequences. Next, we use the knowledge so gained to turn tables (this is a common strategy in algebra: assume a solution, find its properties, and if these turn out to be sufficient, use them to determine the solution.). From eq. 5.2, we find:

$$H_k = \begin{bmatrix} \cdots & C_k A_{k-1} A_{k-2} B_{k-3} & C_k A_{k-1} B_{k-2} & C_k B_{k-1} \\ \ddots & C_{k+1} A_k A_{k-1} A_{k-2} B_{k-3} & C_{k+1} A_k A_{k-1} B_{k-2} & C_{k+1} A_k B_{k-1} \\ \ddots & C_{k+2} A_{k+1} A_k A_{k-1} A_{k-2} B_{k-3} & C_{k+2} A_{k+1} A_k A_{k-1} B_{k-2} & C_{k+2} A_{k+1} A_k B_{k-1} \\ \iddots & \ddots & \ddots & \vdots \end{bmatrix} \tag{6.2}$$

and we see immediately that H_k factors:

$$H_k = \begin{bmatrix} C_k \\ C_{k+1}A_k \\ C_{k+2}A_{k+1}A_k \\ \vdots \end{bmatrix} \begin{bmatrix} \cdots & A_{k-1}A_{k-2}B_{k-3} & A_{k-1}B_{k-2} & B_{k-1} \end{bmatrix} := \mathbf{O}_k\mathbf{R}_k \tag{6.3}$$

Interpreting this decomposition by applying u_{p_k} on the right, we see that $\mathbf{R}_k u_{p_k} = \begin{bmatrix} \cdots & A_{k-1}A_{k-2}B_{k-3} & A_{k-1}B_{k-1} & B_{k-1} \end{bmatrix} u_{p_k} = x_k$, the state for this realization, and, in a sense dually, $\mathbf{O}_k x_k = y_{f_k}$, when u_{f_k} is zero. Referring back to chapter 3, we realize that we have discovered the *reachability operator* $\mathbf{R}_k$ at index k, and, dually, the *observability operator* $\mathbf{O}_k$ at the same index.

Before 'turning the tables', we have to derive important (sufficient) properties of the decomposition of the H_k, given a realization. First, we see that $C_k = [\mathbf{O}_k]_k$ and $B_{k-1} = [\mathbf{R}_k]_{k-1}$. Next, considering $[\mathbf{O}_k]_{(k+1):}$ (the beheaded $\mathbf{O}_k$, which we also write as $\mathbf{O}_k^{\uparrow}$), we see that $[\mathbf{O}_k]_{(k+1):} = \mathbf{O}_{k+1}A_k$. So, if $\mathbf{O}_{k+1}$ is left invertible, i.e., if there exists a *pseudo-inverse* $\mathbf{O}_{k+1}^{\dagger}$ such that $\mathbf{O}_{k+1}^{\dagger}\mathbf{O}_{k+1} = I$, then $A_k = \mathbf{O}_{k+1}^{\dagger}[\mathbf{O}_k]_{(k+1):}$[2]. Dually, $A_k[\mathbf{R}_k] = [\mathbf{R}_{k+1}]_{:(k-1)}$, and hence $A_k = [\mathbf{R}_{k+1}]_{:(k-1)}\mathbf{R}_k^{\dagger}$, provided $\mathbf{R}_k$ is right invertible. These observations give rise to a few definitions:

Definition 1 *We say that a realization is reachable iff all $\mathbf{R}_k$ are right invertible. Equivalently, the rows of each $\mathbf{R}_k$ are linearly independent.*

Definition 2 *We say that a realization is observable iff all $\mathbf{O}_k$ are left invertible. Equivalently, the columns of $\mathbf{O}_k$ are linearly independent.*

Definition 3 *We say that a realization is minimal iff it is both reachable and observable.*

When a realization is minimal, then the corresponding factorizations of the H_k are minimal as well, and conversely, when all factorizations of H_k are minimal then the resulting realization is minimal. That means that, in the minimal case, the columns of $\mathbf{O}_k$ form a basis for the range of H_k, while at

[2]Often $X^{\dagger}$ is used as notation for the Moore-Penrose inverse of X. One may also allow more general pseudo-inverses here, which may be computationally less demanding.

the same token, the rows of $\mathbf{R}_k$ form a basis for the co-range of H_k (the range of H_k'), as happens with any minimal factorization of a matrix.

The gist of realization theory is now that, conversely, factorizations of H_k produce realizations, at least in the minimal case. We formulate this as a theorem (this is maybe the most important theorem in system theory, which we could call the "generalized Kronecker theorem"—see the notes at the end of the chapter on this.).

Theorem 1 *Let, for each k, $H_k := \mathbf{O}_k\mathbf{R}_k$ be a minimal factorization of H_k, then a corresponding minimal realization is given by*

$$\left\{\begin{array}{lcl} A_k & = & \mathbf{O}_{k+1}^{\dagger}[\mathbf{O}_k]_{(k+1):} \\ B_k & = & [\mathbf{R}_{k+1}]_k \\ C_k & = & [\mathbf{O}_k]_k \\ D_k & = & T_{k,k} \end{array}\right. \tag{6.4}$$

In addition, $A_k = [\mathbf{R}_{k+1}]_{:(k-1)}\mathbf{R}_k^{\dagger}$ as well.

Proof

We have first to show that the realization so defined reproduces the entries in the subsequent Hankel operators H_k. This we do by first showing that the realization as given reproduces all the left factors $\mathbf{O}_k$. This is clear by definition for the first entries given by the subsequent C_k. For the remaining entries $[\mathbf{O}_k]_{(k+1):}$, we observe that, with the definitions given $\mathbf{O}_{k+1}A_k = \mathbf{O}_{k+1}\mathbf{O}_{k+1}^{\dagger}[\mathbf{O}_k]_{(k+1):}$. $\Pi_{k+1} := \mathbf{O}_{k+1}\mathbf{O}_{k+1}^{\dagger}$ is a (perhaps skew)[3] projection operator on the range of H_{k+1}. But, *the columns of $[\mathbf{O}_k]_{(k+1):}$ belong to the range of H_{k+1} since we assumed a minimal factorization*, so Π_{k+1} projects them on themselves, and hence $[\mathbf{O}_k]_{(k+1):} = \mathbf{O}_{k+1}A_k$. It follows that the $\{A_k, C_k\}$ define all the subsequent $\mathbf{O}_k$, starting from the highest relevant value of k. A dual reasoning works for the $\mathbf{R}_k$, now with the definition $A_k = [\mathbf{R}_{k+1}]_{:(k-1)}\mathbf{R}_k^{\dagger}$. Hence remains to be shown that the two definitions for A_k are equivalent.

From the way the Hankel operators are intertwined, we have $[H_k]_{(k+1):,:} = [H_{k+1}]_{:,:(k+1)}$ and hence $[\mathbf{O}_k]_{(k+1):}\mathbf{R}_k = \mathbf{O}_{k+1}[\mathbf{R}_{k+1}]_{:(k-1)}]$. Pre- and post multiplication with respect. $\mathbf{O}_{k+1}^{\dagger}$ and $\mathbf{R}_k^{\dagger}$ produces the equality of the definitions of A_k. QED

[3]Depending on the pseudo-inverse used. In case the Moore-Penrose inverse is used, then the projection is orthogonal.

Figure (6.2) illustrates the intertwining property of the Hankel operators. Concerning the pseudo-inverses, up to this point we have not supposed them to be Moore-Penrose inverses, so actually there is a large collection of pseudo-inverses possible, which, however, would all produce the same results given the minimal factorizations. Also with the choice of bases for the columns or rows of each H_k there is a lot of freedom possible, which we shall soon exploit.

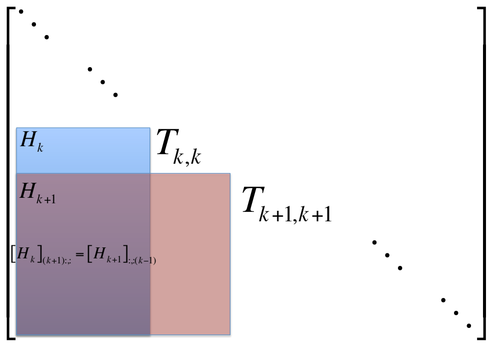


Figure 6.2: How the Hankel operators at k and $k+1$ are intertwined.

Example

Consider

$$T = \begin{bmatrix} 1 & & & \\ 0 & 1 & & \\ 0 & 0 & 1 & \\ 1 & 0 & 0 & 1 \end{bmatrix} \tag{6.5}$$

with the normal matrix indexing schema. We have $H_1 = \begin{bmatrix} | \\ | \\ | \\ | \end{bmatrix}$ empty. Next:

$$H_2 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, H_3 = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \end{bmatrix}, H_4 = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \tag{6.6}$$

and hence $C_1 = |$ is empty, $C_2 = 0$, $C_3 = 0$ and $C_4 = 1$. Likewise, $B_1 = 1$, $B_2 = 0$, $B_3 = 0$, B_4 is empty. We can take $\mathbf{O}_2^\dagger = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$, $\mathbf{O}_3^\dagger = \begin{bmatrix} 0 & 1 \end{bmatrix}$ and $\mathbf{O}_4^\dagger = [1]$, so that A_4 is empty, $A_3 = 1 * 1 = 1$, $A_2 = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = 1$ and A_1 again empty. The result is shown in fig. (6.3). Remark that in this

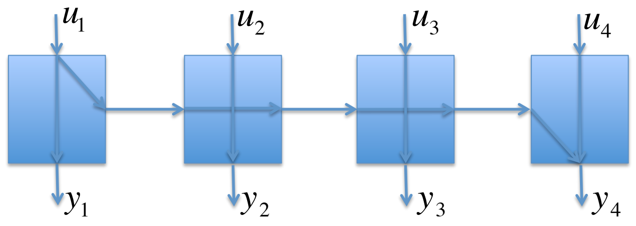


Figure 6.3: A realization example.

example, other values for e.g., $\mathbf{O}_2^\dagger$ would be $\begin{bmatrix} x & y & 1 \end{bmatrix}$ with arbitrary values for x and y, and that this would not influence the result. Other minimal realizations would be found by factoring differently, e..g., one could write $H_3 = \begin{bmatrix} 0 \\ 2 \end{bmatrix} \begin{bmatrix} 1/2 & 0 \end{bmatrix}$ and one would have obtained a somewhat different realization (in this simple example the differences are not that big.).

6.2 The time-invariant case*

The time-invariant case is in essence not much different from the matrix case just treated, except that the matrices involved are now infinite dimensional. A fully responsible treatment necessitates the introduction of a mathematical framework that allows for such matrices, because matrix-vector products are not guaranteed to converge any more. We do this in a separate chapter (chapter **??**). Because of time-invariance, all Hankel operators are now equal, let's look at H_0, and let $T(z) = T_0 + zT_1 + z^2T_2 + \cdots$, then

$$H_0 = \begin{bmatrix} \cdots & T_3 & T_2 & T_1 \\ \ddots & T_4 & T_3 & T_2 \\ \ddots & T_5 & T_4 & T_3 \\ \iddots & \ddots & \ddots & \vdots \end{bmatrix}. \tag{6.7}$$

This is a famous 'block Hankel matrix', which is normally written as

$$H := \begin{bmatrix} T_1 & T_2 & T_3 & \cdots \\ T_2 & T_3 & T_4 & \iddots \\ T_3 & T_4 & T_5 & \iddots \\ \vdots & \iddots & \iddots & \ddots \end{bmatrix} \tag{6.8}$$

after reordering of the columns (corresponding to reversing the input order). The Hankel matrix is highly regular (it has identical blocks on anti-diagonals), and it is handy to introduce a constructor, that produces a Hankel matrix from a series of blocks (notice a different definition of H_k in this section!):

$$H_k := \text{Han}(T_1, T_2, \cdots, T_k) := \begin{bmatrix} T_1 & T_2 & \cdots & T_k \\ T_2 & T_3 & \iddots & T_{k+1} \\ T_3 & T_4 & \iddots & T_{k+2} \\ \vdots & \iddots & \iddots & \vdots \\ T_k & T_{k+1} & \cdots & T_{2k-1} \end{bmatrix} \tag{6.9}$$

The famous result (originally due to Kronecker in the scalar case) is now that $T(z)$ is a rational transfer function, if and only if H is finite dimensional (i.e., its rows and columns span finite dimensional spaces). This somewhat delicate question (because the T_k might not be bounded when k increases, producing unbounded H_k's) can luckily be dealt with without recourse to infinite matrices. Suppose that for a given Hankel matrix we can ascertain that there is an index k so that $\text{Han}(T_1 \cdots T_\ell)$ has the same rank δ for all $\ell \geq k$, then this can be used as a criterion for finite-dimensionality of H, even when H turns out not to be bounded in any reasonable metric. Actually, one can even take k as the minimal such value, in which case one would call it the *order* of the system, with the rank δ as its *degree*. Taking a finite H_ℓ with $\ell > k$ and assuming the H_ℓ to be block-indexed starting at index 1, let us factorize it minimally as $H_\ell = \mathbf{O}_\ell \mathbf{R}_\ell$, where $\mathbf{O}_\ell$ has δ columns and $\mathbf{R}_\ell$, δ rows. One can now see easily that $H_{\ell-1}$ has a minimal factorization derived from the previous, as $H_{\ell-1} = [\mathbf{O}_\ell]_{1:(\ell-1)}[\mathbf{R}_\ell]_{1:(\ell-1)}$, also of rank δ. The construction of the previous section applies here as well:

$$\begin{cases} A & := & [\mathbf{O}_\ell]^\dagger_{1:(\ell-1)}[\mathbf{O}_\ell]_{2:}(= [\mathbf{R}_\ell]_{2:}[\mathbf{R}_\ell]^\dagger_{1:(\ell-1)}) \\ B & := & \mathbf{R}_1 \\ C & := & \mathbf{O}_1 \\ D & := & T_0 \end{cases} \tag{6.10}$$

and the proof runs precisely like in theorem 1.

Once the full rank δ is reached at block index k, all the subsequent finite Hankel matrices can actually be constructed from H_k because the additional rows and columns are all linearly dependent on those in H_k, and contain sufficient blocks that are already known, which determine the further coefficients in the factorization. This *partial realization* method generalizes actually to the general LTV case—see the next section.

6.3 Partial realizations in the matrix case

Reverting back to the matrix LTV case and the notation of section 6.1, suppose that for some reason we know upper limits to the sizes $a_k > k$ and $b_k < k$ needed for each partial $[H_k]_{k:a_k,b_k:k}$ to reach the full rank δ_k, with, in addition, $a_{k+1} \geq a_k$ and $b_{k+1} \geq b_k$. This would e.g., be the case when the matrix is block-banded, the inverse of block-banded, or else because it is known that elements decay rapidly away from the diagonal, so that at some point one can assume them to contribute little (notice: this can be a dangerous assumption in the case of very large matrices!). In matrices from modeling problems such knowledge is often the case. It turns out that minimal factorizations of somewhat enlarged sub-matrices of H_k are then sufficient to find a realization.

Consider therefore minimal factorizations of $[H_k]_{k:a_k,b_k:k} := \hat{\mathbf{O}}_k\hat{\mathbf{R}}_k$, a partial Hankel operator as shown in fig. (6.4). Then, assuming that such a factorization is compatible with a realization, it would follow that

$$\left\{ \begin{array}{rcl} A_k & = & \hat{\mathbf{O}}^{\dagger}_{k+1} H_{k+1:a_{k+1},b_k:k} \hat{\mathbf{R}}^{\dagger}_k \\ B_k & = & [\hat{\mathbf{R}}_{k+1}]_k \\ C_k & = & [\hat{\mathbf{O}}_k]_k \\ D_k & = & T_{k,k} \end{array} \right. \tag{6.11}$$

very much as before. The reason is that factorizations of the somewhat larger partial Hankels produce factorizations of smaller ones, and can be extended to factorizations of larger ones. For a full proof, see [10], but it is not hard to see that

$$H_{k+1:a_{k+1},b_k:k} = \hat{\mathbf{O}}_{k+1} A_k \hat{\mathbf{R}}_k \tag{6.12}$$

which is the main relation used in the proof. Besides the partial Hankel operators, knowledge of the orange colored sub-matrix is needed as well. We leave it to the reader to investigate how the ideas of theorem 1 can be refined to handle the present case.

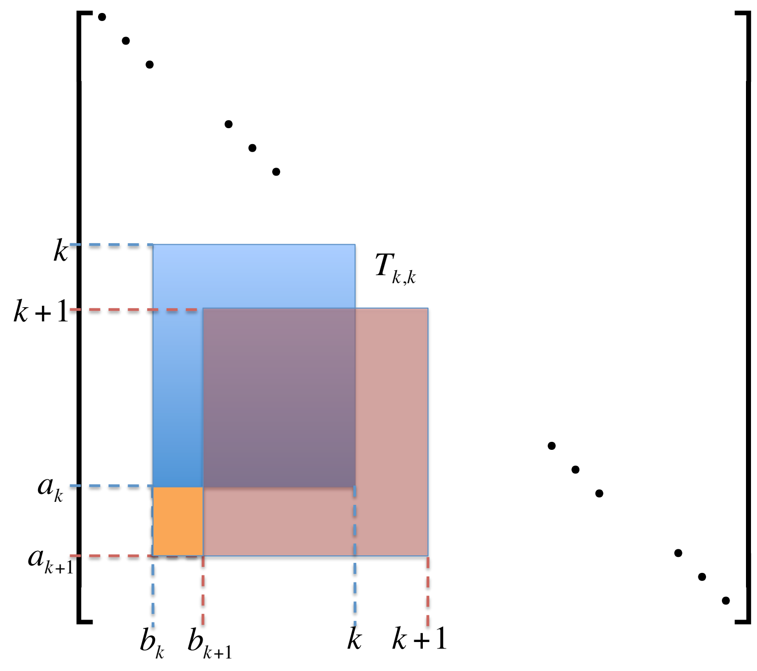


Figure 6.4: Realization with partial Hankel operators.

6.4 Identifying a running system*

Up to this point, we assumed the input-output matrix given, and the problem was to find a minimal realization. However, direct access to all the information needed, i.e., all the entries of the matrix (i.e., all the impulse responses at each index point), may not be available, and the realization has to be made from indirect measurements, or it may be that only partial information is available. We already encountered that case in the previous section, where we were able to construct a realization from smaller submatrices of the Hankel operators, provided some additional information on the system was there, in that case: the maximum delay needed to reach the dimension of the state at each index point. A further question would be: can one identify the system just based on observing inputs and outputs?

Let us first, for exploration sake, look at systems with one input and one output. Suppose the input is u and the output y, and we know already that the system is causal. Can we specify the matrix from that information? Clearly, one has to bring in all the extra information one has and find input-output matrices that are compatible with it. For example: time-invariance. This can be fully accounted for by observing that if $u \mapsto y$, then also $Zu \mapsto Zy$ etc... a full set of input-output maps becomes available, merely by shifting (assuming u starts at some point):

$$\begin{bmatrix} u_0 & 0 & 0 & \cdots \\ u_1 & u_0 & 0 & \ddots \\ u_2 & u_1 & u_0 & \ddots \\ \vdots & \ddots & \ddots & \ddots \end{bmatrix} \mapsto \begin{bmatrix} y_0 & 0 & 0 & \cdots \\ y_1 & y_0 & 0 & \ddots \\ y_2 & y_1 & y_0 & \ddots \\ \vdots & \ddots & \ddots & \ddots \end{bmatrix} \tag{6.13}$$

Suppose now that u_0 is neither unreasonably small nor large, then this map, when restricted to dimension n, specifies the first n entires of the input-output map:

$$\begin{bmatrix} T_0 & 0 & 0 & \cdots \\ T_1 & T_0 & 0 & \ddots \\ T_2 & T_1 & T_0 & \ddots \\ \vdots & \ddots & \ddots & \ddots \end{bmatrix} = \begin{bmatrix} y_0 & 0 & 0 & \cdots \\ y_1 & y_0 & 0 & \ddots \\ y_2 & y_1 & y_0 & \ddots \\ \vdots & \ddots & \ddots & \ddots \end{bmatrix} \begin{bmatrix} u_0 & 0 & 0 & \cdots \\ u_1 & u_0 & 0 & \ddots \\ u_2 & u_1 & u_0 & \ddots \\ \vdots & \ddots & \ddots & \ddots \end{bmatrix}^{-1} \tag{6.14}$$

(observe: the result will automatically be finite Toeplitz.). Hence, a partial representation of the I/O-map in the sense of the previous section has been obtained.

In the case of a time-variant system, this does not work, because we loose the shift property, and hence cannot construct a full, invertible input matrix that will determine the transfer function, at least partially. Not enough information is available to construct the full input-output map. Suppose, to begin with, that a single input-output pair (u, y) is the only information available, what is a system with minimal state dimension that we can construct that will produce it? First, the system we are identifying is known to be linear, so we should surely have $ua \mapsto ya$ for any number a. Next, to be a bit more specific, let us start at $k = 0$, and suppose we input just u_0, leaving all the next entries 0, then we should certainly have, because of assumed causality, the first output to be y_0, all the following entries being unknown (since we do not have the impulse response for a single input at $k = 0$ available, hence we end up with quite a bit of freedom). Next, a similar argument is valid for the inputs $\mathrm{col}[u_0, u_1]$ for which we only know that it must produce $\mathrm{col}[y_0, y_1]$, and so on for the further entries. It follows that the following map will hold (with unspecified and hence potentially free to choose entries marked '?'):

$$\begin{bmatrix} T_{0,0} & 0 & 0 & \cdots \\ T_{1,0} & T_{1,1} & 0 & \ddots \\ T_{2,0} & T_{2,1} & T_{2,2} & \ddots \\ \vdots & \ddots & \ddots & \ddots \end{bmatrix} \begin{bmatrix} u_0 & u_0 & u_0 & \cdots \\ 0 & u_1 & u_1 & \ddots \\ 0 & 0 & u_2 & \ddots \\ \vdots & \ddots & \ddots & \ddots \end{bmatrix} \mapsto \begin{bmatrix} y_0 & y_0 & y_0 & \cdots \\ ? & y_1 & y_1 & \ddots \\ ? & ? & y_2 & \ddots \\ \vdots & \ddots & \ddots & \ddots \end{bmatrix} \tag{6.15}$$

Let us now, for discussion's sake, make the rough assumption that all the entries u_k up to $k = n$ (supposing we have to go that far) are nicely invertible, then the u matrix above is invertible, and we may see that the row-ranks of the Hankel operators build on the strictly lower part of the T matrix will be the same as for the corresponding lower Hankel operators in the output matrix, all consisting of '?'s, because the inverse of the u matrix is again upper and the lower part of the product of the y matrix with the inverse of the u matrix does not depend on the upper part of the y matrix. Hence, if we want to find a realization that is minimal in the state dimensions, we should choose all these ranks to be as small as possible, the simplest would be to put all the '?' equal to zero. That would then simply yield a diagonal T with $T_{k,k} = y_k/u_k$, obviously a solution and the simplest possible—it is *algebraically minimal* as it contains exactly the same number of free parameters as the problem has.

What now when one entry $u_k = 0$? We see that in this case u_k has no influence on y_k (multiplying u_k with whatever constant produces zero): the value of y_k has to be generated, because of causality, from previous values of u, and relevant '?' cannot be zero any more (for example: if one wishes $\begin{bmatrix} T_{0,0} & 0 \\ T_{1,0} & T_{1,1} \end{bmatrix} \begin{bmatrix} u_0 & u_0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} y_0 & y_0 \\ ? & y_1 \end{bmatrix}$, one has no alternative but choosing $T_{1,0} = y_1/u_0$ and hence $? = y_1$.). In the more general case one can fill in the ? entries recursively so as to minimize the ranks of the subsequent Hankel operators, taking care of correct ranges as given by the right hand side (the y matrix)—we leave the details for the "Discussion items". So we see that in the LTV case we end up with realizations that have much smaller state dimensions than what would be the case for LTI systems, although they also are not without some constraints forced by linearity and causality. The Hankel theory gives us a good grasp on the situation.

Let us now briefly consider generalizations to multi-input multi-output systems (so called MIMO systems). In the LTI case, a single input-output pair will not suffice anymore to characterize the full transfer function, one would need a sufficient number of independent inputs, at least the same number as the number of input ports. This, together with shifted versions, would then provide a full characterization of the transfer function, by inversion of the input data as done in the one input case. For LTV systems the situation is even more complex: not only is information needed about inputs applied at all the input ports, but also at every time point. Again as before, such situations can be analyzed using, in particular, the Hankel theory that we have developed, but the situation quickly becomes quite complex and specific for the example at hand. See the notes for some literature references on the issues.

6.5 Working with infinite indices*

With the exploration done so far, it is not too hard to imagine what to do when we deal with LTV systems running for all times (from $-\infty$ to $+\infty$). Let us assume that we dispose of a full set of impulse responses at each index point, or, equivalently, of the matrix input/output operator T, which we assume to be causal, and that we have to compute a realization, assuming that the system has one. The most practicable way then is to assume that we can *compute* any $T_{i,k}$ $(i \geq k)$ or that any such is given if desired.

From our knowledge so far, we know that a realization with finite states will exist, iff each Hankel H_k (which is now an infinitely indexed matrix) has finite rank, by which is meant that the rank of each sub-Hankel $[H_k]_{\ell:k,:}$ of H_k will be finite, uniformly in ℓ. Indeed, assuming a minimal realization to exist, one has

$$\begin{aligned}
[H_k]_{\ell:k,:} &= \begin{bmatrix} C_k A_{k+1} \cdots A_{\ell-1} B_\ell & \cdots & C_k B_{k-1} \\ C_{k+1} A_k \cdots A_{\ell-1} B_\ell & \cdots & C_{k+1} A_k B_{k-1} \\ \vdots & \vdots & \vdots \end{bmatrix} \\
&= \begin{bmatrix} C_k \\ C_{k+1} A_k \\ \vdots \end{bmatrix} \begin{bmatrix} A_{k+1} \cdots A_{\ell-1} B_\ell & \cdots & B_{k-1} \end{bmatrix}
\end{aligned} \tag{6.16}$$

so the maximal rank of all these over ℓ will be the dimension of the state space at index point k. If δ_k is this dimension, then there will be an ℓ_k such that for all $\ell \geq \ell_k$ the rank of $[H_k]_{\ell:k,:}$ will be δ_k, and the factorization shown will be a minimal factorization of that $\ell \geq \ell_k$. Conversely, adopting the strategy of section 6.3, and finding such ℓ_k and ℓ_{k+1}, a realization at point k can be found by choosing compatible bases as explained in that section—we leave it to the reader to figure out details.

While the previous paragraph does produce a solution to the identification problem, it is somewhat unsatisfactory, in that quantities ℓ_k and ℓ_{k+1} have to be found that are dependent on unknown quantities, namely the degrees δ_k and δ_{k+1} of the state at index point k. One can of course determine the ranks of Hankel submatrices progressively, but that may amount to a lot of work without guarantee that one has gone out far enough to reach the maximal rank.

The situation is of course inherent to the identification problem: in some systems and especially time-variant systems, things may happen at unforeseen points in time, but they can be detected by the rank conditions developed so far. On the one hand, that is satisfying because it is a precise condition, on the other, it may require a lot of computation, which, in many systems would be unnecessary, as one can put a bound on potential time delays in the system's operation. If the latter is the case, then one can often state that (1) there is an upper limit to the state dimension over all k, and (2) there is an upper limit ℓ_m to ℓ_k and how big ℓ must be in $[H_k]_{\ell:k,k:\ell}$ to reach the state dimension, also uniformly over k, so that partial realization theory can be used on submatrices $[H_k]_{\ell_m:k,k:\ell_m}$. System's with this property are sometimes called *locally finite systems*.

6.6 Discussion items

1. An important issue, not covered in this chapter, is what happens when 'partial' identification is done: one uses data up to a certain order, does the identification and one obtains a realization. How does the realization obtained relate to the original system? [Some questions, e.g., what gets interpolated, may be easy to answer; other, e.g., how well does the partially identified system approximate the original seem much harder.]

2. There are other ways how to approximate or identify a system (or an approximation), given data or information on it. One is to use various kinds of 'interpolations' and even specific properties the system may be known to have. E.g., it may be known that the system is 'bounded' with respect to some norm, and that hence the approximation should satisfy the same constraint. We devote quite a few subsequent chapters to such issues.

3. In the time invariant case, it is known that approximating or identifying a system on the basis of either the impulse response or the resulting transfer function may not be the best thing to do. In particular, the frequency response of the system may be dependent on the given coefficients in a highly sensitive way (meaning: tiny variations in the value of the coefficients entails big variations in frequency response.). This brings in the issue of conditioning of a problem and its relation to numerical stability. This highly important question is given some considerations in the "Notes on Linear Algebra" at the end of the book.

6.7 Notes

- Needless to say, system identification is a central problem in system theory, and a lot of literature has been devoted to it. Far from giving a full account, we want to mention a few salient contributions. The first results of what later became identification theory are due to the German mathematician Kronecker, who showed that a Maclaurin series belongs to a rational function iff the Hankel matrix build on its coefficients has finite rank. Kronecker's work was easily extended to rational matrix functions and has become a standard fixture of, among other, matrix function and Hardy space theory [31].

- In the sixties of the previous century and in the wake of the development of state space theory, a search was on for methods of identifying the state space description (the so called 'realization') of a linear, discrete-time and time-variant system, and to derive both conditions and algorithms to ascertain that the system given by its (multiport) impulse response has a finite dimensional state space. Best known is probably the contribution of B.L. Ho and R.E. Kalman [19], but also L.M. Silverman and H.E. Meadows [33] solved the problem simultaneously. The method presented in this chapter, based on the global Hankel operator, was first presented by A.J. Van der Veen in his thesis [39] and subsumes much of the previous work.

- A nice observation is that system identification from the knowledge of the input-output transfer data (the transfer matrix or transfer operator) amounts to the determination of the range and the co-range of all the relevant Hankel operators derived from the transfer data. This is precisely the data that leads to minimal factorizations of the subsequent Hankels! This simple observation greatly simplifies both the methods for identification and the proofs.

- Identification using real life measurements is a more involved issue that we could only touch in this chapter, although it uses the same basic principles (how the system maps strict past inputs to the future). Original work on this topic and further references can be found in [41, 42, 40].

- Very often, the determination of a realization of one or the other transfer operator is part of a more general problem such as solving systems of equations, matrix factorization, spectral factorization or the approximation of a matrix with a low degree realization. In many of the following chapters we shall see that the methods presented in this chapter reappear as building blocks to solve those more elaborate issues.

Chapter 7

State equivalence, state reduction

The state of a system sits in its interior, it is often hidden (this property is sometimes called 'Markovian', especially when the dimension of the state is small). A consequence is that the state is not uniquely characterized by the input-output behavior of the system. On the one hand, there may be redundancy in the state (think of the memory of a computer: it may contain many items that are not relevant to a given problem). On the other hand, different minimal states may produce the same input-output behavior, they are equivalent from the point of view of the input-output map. In this section we deal with these two issues in the context of discrete-time, finitely indexed LTV-systems (the matrix case), ending the chapter with some considerations concerning infinitely indexed, discrete-time LTV and LTI systems. Changing the state description without changing the behavior can produce desirable effects, in particular, it can produce interesting canonical forms: the input or output normal forms and balanced realizations, all of which play important roles in further developments (approximation, inversion, estimation, control and synthesis).

Menu

Hors d'oeuvre

Introspection

First course
Equivalences of minimal states

Second course
Reduction to minimal
A first square-root algorithm

Third course
The LTI case

Dessert
The case of infinite indices

7.1 Equivalences of minimal LTV system realizations

In the previous chapter, we discovered that a minimal realization of a discrete-time, LTV system can be obtained through factorization of the Hankel operator H_k at each index point k of a system with input-output map T: $H_k = \mathbf{O}_k\mathbf{R}_k$. The reachability operator $\mathbf{R}_k$ maps the 'strict past' of the system to a (minimal) state, while the observability operator $\mathbf{O}_k$ maps the contribution of the state at index point k to the future (because of linearity, that contribution is a linear component of the output from k on.).

Factorizations of a matrix are by no means unique, even when they are minimal: choose for each k an otherwise arbitrary invertible matrix S_k and $H_k = \widehat{\mathbf{O}}_k\widehat{\mathbf{R}}_k$, with $\widehat{\mathbf{R}}_k := S_k\mathbf{R}_k$ and $\widehat{\mathbf{O}}_k := \mathbf{O}_kS_k^{-1}$ will likewise be a good minimal factorization. The new state becomes $\widehat{x}_k := S_kx_k$, and the new realization:

$$\left\{\begin{array}{lcl} \widehat{x}_{k+1} & = & \widehat{A}_k\widehat{x}_k + \widehat{B}_ku_k \\ y_k & = & \widehat{C}_k\widehat{x}_k + D_ku_k \end{array}\right. \tag{7.1}$$

with

$$\begin{bmatrix} \widehat{A}_k & \widehat{B}_k \\ \widehat{C}_k & D_k \end{bmatrix} := \begin{bmatrix} S_{k+1}A_kS_k^{-1} & S_{k+1}B_k \\ C_kS_k^{-1} & D_k \end{bmatrix}. \tag{7.2}$$

Such state transformations are commonly used to produce interesting 'canonical' forms for the realization—canonical means 'characterizing', often uniquely in a certain sense. The first idea we develop further is to choose an orthonormal basis for either the reachability spaces or the observability spaces.

Definition 4 *One says that the minimal realization of a system is in input normal form, when all its* $\begin{bmatrix} A_k & B_k \end{bmatrix}$ *are co-isometric (i.e.,* $A_kA_k' + B_kB_k' = I$*). Dually, one says that the realization of a minimal system is in output normal form, when all its* $\begin{bmatrix} A_k \\ C_k \end{bmatrix}$ *are isometric (i.e.,* $A_k'A_k + C_k'C_k = I$*).*

As we shall see, these choices amount to finding an orthonormal basis for either the reachability or the observability spaces of the system (i.e., the co-range, respect. the range, of the Hankel operator H_k at index k). To determine such bases from the transfer data is not immediately easy: we are dealing with infinite indices. So it makes sense to study how the result can be obtained with just finite matrix calculations. Suppose therefore, we are given a minimal, but otherwise arbitrary realization $\{A, B, C, D\}$, what does it take to put it in one or the other normal form? To start with the input normal form: we should find a transformation S_k at each index point, which is so that the resulting $\begin{bmatrix} \widehat{A}_k & \widehat{B}_k \end{bmatrix} := \begin{bmatrix} S_{k+1}A_kS_k^{-1} & S_{k+1}B_k \end{bmatrix}$ is co-isometric, or

$$S_{k+1}A_kS_k^{-1}(S_k^{-1})'A_k'S_{k+1}' + S_{k+1}B_kB_k'S_{k+1}' = I \tag{7.3}$$

for each index k. Let $M_k := (S_k'S_k)^{-1}$, then this reduces to the recursive equation

$$M_{k+1} = B_kB_k' + A_kM_kA_k'. \tag{7.4}$$

This latter equation is called a *Lyapunov-Stein equation* and it can be solved recursively, going from k to $k+1$, provided one knows an initial, starting matrix. In view of special form M_k must have, it must be strictly positive definite, so that its inverse can be factorized subsequently into $S_k'S_k$ with S_k square non-singular. Let us investigate further whether such a solution exists indeed and how it can be obtained.

To start, suppose that the system is at first empty and its first index point at which it becomes active is k_0. The state x_{k_0} is assumed empty, and $x_{k_0+1} = B_{k_0}u_{k_0}$. Hence, M_{k_0} will be empty, and $M_{k_0+1} = B_{k_0}B_{k_0}'$. Is the latter non-singular? Well, we assumed the realization to be minimal, and the first non-empty Hankel operator factorizes as $H_{k_0+1} = \mathcal{O}_{k_0+1}B_{k_0}$, which we assumed to be a minimal factorization. Hence B_{k_0} must have independent rows and M_{k_0+1} has to be non-singular. *M_{k_0+1} is actually the Gramian of the chosen row-basis of* $\mathbf{R}_{k_0+1}$ in the given realization[1].

This last property generalizes recursively. The next $M_{k_0+2} = B_{k_0+1}B'_{k_0+1} + A_{k_0+1}B_{k_0}B'_{k_0}A'_{k_0+1}$, and will again be non-singular (because $\mathbf{R}_{k_0+1} = \begin{bmatrix} A_{k_0+1}B_{k_0} & B_{k_0+1} \end{bmatrix}$ had been chosen non-singular in the realization) and positive definite (automatically). $M := \mathrm{diag}(M_k)$ can be interpreted as nothing else than the global Gramian of the reachability operator $\mathbf{R}$, but at this point we have not defined the necessary global constructs yet to do so—we shall do that in a specialized chapter devoted to this point, chapt. **??**. In the matrix case considered here, the recursion keeps on progressing until it terminates at the last index k_ℓ. All this will go well if the subsequent transition matrices A_k are properly bounded, actually their continuous product $A_kA_{k-1}\cdots A_{k_0+1}$ comes into play, which may blow up exponentially. This eventuality is of some concern in the infinitely indexed case and will be discussed to some extent later in this chapter.

For the output normal form, something similar happens dually (with also time reversed). Requiring $\begin{bmatrix} \widehat{A}_k \\ \widehat{C}_k \end{bmatrix} = \begin{bmatrix} S_{k+1}A_kS_k^{-1} \\ C_kS_k^{-1} \end{bmatrix}$ to be isometric now leads, with $N_k := S'_kS_k$, to the *backward* Lyapunov-Stein equation

$$N_k = A'_kN_{k+1}A_k + C'_kC_k \tag{7.5}$$

and N_k can be interpreted as the Gramian of the local observability basis at index point k—which because of minimality will be non-singular.

7.2 A square-root algorithm to compute normal forms

It is of course a pertinent question whether the S_k in the previous calculations of a normal form could not be determined directly (not via their gramian)—and they can, with great numerical benefits. Let us concentrate on the output normal form (the backward recursion), the case of the input normal form is

[1]If we also assume that the input u_{k_0} maps one-to-one to the first non-empty state x_{k_0+1}, then B_{k_0} will have to be square non-singular as well. This is an assumption that is usually made: inputs to be non-redundant. If they are not, then their dimension can be reduced to the co-range of B_{k_0}.

just dual (and with forward recursion). At stage k one assumes S_{k+1} known, and perform a QR-factorization of

$$\begin{bmatrix} S_{k+1}A_k \\ C_k \end{bmatrix} =: \begin{bmatrix} Q_{1,1} & Q_{1,2} \\ Q_{2,1} & Q_{2,2} \end{bmatrix} \begin{bmatrix} R_k \\ 0 \end{bmatrix} = \begin{bmatrix} Q_{1,1} \\ Q_{2,1} \end{bmatrix} R_k \tag{7.6}$$

then, premultiplying both sides with their complex conjugates, one gets the for this case relevant Lyapunov-Stein equation back, because of the unitarity of the Q matrix. Actually, one gets more. We know already that the left-hand side has to be non-singular, hence also R_k is non-singular, and from the Lyapunov-Stein equation we have that actually $R_k = S_k$, the desired state transformation at index k. It then follows directly that

$$\begin{bmatrix} \widehat{A}_k \\ \widehat{C}_k \end{bmatrix} = \begin{bmatrix} Q_{1,1} \\ Q_{2,1} \end{bmatrix} \tag{7.7}$$

and the QR-factorization has simply become

$$\begin{bmatrix} S_{k+1}A_k \\ C_k \end{bmatrix} =: \begin{bmatrix} \widehat{A}_k & Q_{1,2} \\ \widehat{B}_k & Q_{2,2} \end{bmatrix} \begin{bmatrix} S_k \\ 0 \end{bmatrix} = \begin{bmatrix} \widehat{A}_k \\ \widehat{B}_k \end{bmatrix} S_k \tag{7.8}$$

Thus, a simple recursive QR-factorization solves the Lyapunov-Stein equation in square-root form (S_k can be taken as an upper-triangular 'square root' of $N_k = S_k' S_k$.)[2]. In the next chapter (on canonical factorizations), also the components $Q_{1,2}$ and $Q_{2,2}$ will acquire significance, see there.

Why is solving a square-root equation preferable to solving the Lyapunov-Stein equation directly? First of all, finding S_k directly guarantees the positivity of $N_k = S_k' S_k$. However, there is more. One can show (see the mathematical introduction in the appendix), that the condition number of N_k is the square of the condition number of S_k. That means that one looses half of the significant bits in the direct computation of N_k as compared to the computation via S_k. The result is that only the square root equation will give results that are accurate within the range allowed by the precision offered by the computer, except in the most elementary cases. The square-root calculation must hence be preferred in almost all cases.

[2]Strictly speaking, the square root $N_k^{1/2}$ of N_k is a symmetric or hermitian matrix and is unique. However, for most applications any minimal size factorization $N_k = S_k' S_k$ works, in which case S_k is viewed as a kind of generalized square root.

7.3 Balanced realizations

Balanced realization theory plays an important role in model reduction theory—i.e., when one want to represent a system with a reduced but still accurate model. A balanced realization treats the input and output side of the realization equation equally in some sense (the sense is: the 'energy' conveyed from the input to the state equals the energy the state conveys to the output, where energy is measured in quadratic norm squared.). Be that as it may, here is how the balanced form is obtained at each index point k.

We start out with a minimal realization $\{A, B, C, D\}$ in input normal form (say). The local reachability Gramian $\mathbf{R}_k\mathbf{R}_k' = I$ is then unitary. Let the corresponding observability Gramian be $N_k := \mathbf{O}_k'\mathbf{O}_k$, and let's find its eigen decomposition: $N_k = U_k'\Sigma_k^2 U_k$, with U_k unitary and Σ_k diagonal strictly positive (that Σ_k is non-singular is of course a consequence of the minimality assumption.). Let us now define a new realization $\{\widehat{A}_k, \widehat{B}_k, \widehat{C}_k, D_k\} := \{S_{k+1}A_kS_k^{-1}, S_{k+1}B_k, C_kS_k^{-1}, D_k\}$ with $S_k := \Sigma_k^{1/2}U_k$. Then we shall have $\widehat{\mathbf{R}}_k = \Sigma_k^{1/2}U_k\mathbf{R}_k$ and $\widehat{\mathbf{O}}_k = \mathbf{O}_kU_k'\Sigma_k^{-1/2}$ (leaving H_k unchanged), and hence the resulting modified $\widehat{M}_k = \Sigma_k$ and $\widehat{N}_k = \Sigma_k$ as well. Both the reachability and controllability Gramians have become diagonal positive definite, and equal to boot.

It can pretty easily be shown that the realizations so obtained correspond to an SVD of each Hankel operator H_k—actually, this would be a way to derive them as well, be it not that the Hankel operators have infinite indices in general. In the case of finite matrices, the update of the Hankel matrices can be done recursively and combined with the generation of a realization, but that is still an operation with potentially high numerical complexity.

7.4 Reduction to a minimal realization

The next point we have to discuss, is how to reduce a non-minimal realization to a minimal one. Let $\{A, B, C, D\}$ a presumably non-minimal realization, how can it be reduced, without going back to the factorization of the Hankel operators? This question often occurs when systems are being combined with each other, e.g., when their transfer operators are being added, put in cascade or more generally, combined into a connected network—'cancellations' may occur between the systems, for example: when a system is cascaded with its inverse, just a direct feed-through results.

What is needed is a method to discover superfluous parts of the state space. There are two main ways the dimension of the state space may be too large: (1) some states (at some index k) cannot be generated from the input (they will be called 'unreachable'), and then (2) some states cannot be discriminated by future observations (their difference is 'unobservable'). These notions can be made more precise by looking at relevant gramians, which will be singular when the state dimension is too large (we already encountered gramians in the previous discussion.).

Let us start with reachability. The given realization will have reachability operators for each k

$$\mathbf{R}_k = \begin{bmatrix} \cdots & A_{k-1}A_{k-2}B_{k-3} & A_{k-1}B_{k-2} & B_{k-1} \end{bmatrix} \tag{7.9}$$

and related Gramian $M_k := \mathbf{R}_k\mathbf{R}_k'$. When the rows of $\mathbf{R}_k$ form a basis, and hence M_k is non-singular, then all states x_k are reachable with appropriate inputs running from the beginning point of the system to $k-1$. However, if M_k is singular, then there exists a (actually many) non-singular matrix R_k such that

$$M_k = \begin{bmatrix} R_{k,1} & R_{k,2} \end{bmatrix} \begin{bmatrix} \widehat{M}_k & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} R_{k,1}' \\ R_{k,2}' \end{bmatrix} \tag{7.10}$$

with $\widehat{M}_k$ square non-singular. For later use, let us be more precise. The columns of $R_{k,1}$ must form a basis for the range of M_k. It really does not matter what $R_{k,2}$ is, so long as it complements the basis generated by the columns of $R_{k,1}$. It may be useful, but not strictly necessary, to choose the columns of $R_{k,2}$ to form a basis for the kernel of M_k'. In that case one shall have that

$$\begin{bmatrix} R_{k,1} & R_{k,2} \end{bmatrix}^{-1} = \begin{bmatrix} R_{k,1}^{\dagger} \\ R_{k,2}^{\dagger} \end{bmatrix} \tag{7.11}$$

in which the dagger's stand for the respective Moore-Penrose inverses: $R_{k,1}^{\dagger} = (R_{k,1}'R_{k,1})^{-1}R_{k,1}'$ and similar for $R_{k,2}$. However, whatever the choice of an appropriate $R_{k.2}$, we can always have $\widehat{M}_k = (R_{k,1}^{+})'M_kR_{k,1}^{+}$ for any pseudo-inverse $R_{k,1}^{+}$ ($\widehat{M}_k$ is largely not unique, and can even be diagonal.).

Consider now, at each k, the transformation $\widehat{x}_k = R_k^{-1}x_k$. Then we have

$x_k = \left[\begin{array}{cc} R_{k,1} & R_{k,2} \end{array}\right] \left[\begin{array}{c} \widehat{x}_{k,1} \\ \widehat{x}_{k,2} \end{array}\right]$, and subsequently, with

$$\left[\begin{array}{cc} \widehat{A}_k & \widehat{B}_k \\ \widehat{C}_k & D_k \end{array}\right] = \left[\begin{array}{cc} R_{k+1}^{-1} A_k R_k & R_{k+1}^{-1} B_k \\ C_k R_k & D_k \end{array}\right] \tag{7.12}$$

and using the induced partitioning at each index point k

$$\left\{ \begin{array}{ccccccc} \left[\begin{array}{c} \widehat{x}_{k+1,1} \\ \widehat{x}_{k+1,2} \end{array}\right] & = & \left[\begin{array}{cc} \widehat{A}_{k;1,1} & \widehat{A}_{k;1,2} \\ \widehat{A}_{k;2,1} & \widehat{A}_{k;2,2} \end{array}\right] \left[\begin{array}{c} \widehat{x}_{k,1} \\ \widehat{x}_{k,2} \end{array}\right] & + & \left[\begin{array}{c} \widehat{B}_{k;1} \\ \widehat{B}_{k;2} \end{array}\right] u_k \\ y_k & = & \left[\begin{array}{cc} \widehat{C}_{k;1} & \widehat{C}_{k;2} \end{array}\right] \left[\begin{array}{c} \widehat{x}_{k,1} \\ \widehat{x}_{k,2} \end{array}\right] & + & D_k u_k \end{array} \right. \tag{7.13}$$

and, as the forward recursion on M_k is $M_{k+1} = B_k B_k' + A_k M_k A_k'$, we also have

$$\left[\begin{array}{cc} \widehat{M}_{k+1} & 0 \\ 0 & 0 \end{array}\right] = \left[\begin{array}{c} \widehat{B}_{k;1} \\ \widehat{B}_{k;2} \end{array}\right] \left[\begin{array}{cc} \widehat{B}_{k;1}' & \widehat{B}_{k;2}' \end{array}\right] + \left[\begin{array}{cc} \widehat{A}_{k;1,1} & \widehat{A}_{k;1,2} \\ \widehat{A}_{k;2,1} & \widehat{A}_{k;2,2} \end{array}\right] \left[\begin{array}{cc} \widehat{M}_k & 0 \\ 0 & 0 \end{array}\right] \left[\begin{array}{cc} \widehat{A}_{k;1,1}' & \widehat{A}_{k;2,1}' \\ \widehat{A}_{k;1,2}' & \widehat{A}_{k;2,2}' \end{array}\right]. \tag{7.14}$$

It immediately follows from the zero'ing of the (2,2) entry and the non-singularity of $\widehat{M_k}$ that $\widehat{B}_{k;2} = 0$ and $\widehat{A}_{k;2,1} = 0$, so that the hatted realization has the form

$$\left[\begin{array}{cc|c} \widehat{A}_{k;1,1} & \widehat{A}_{k;1,2} & \widehat{B}_{k;1} \\ 0 & \widehat{A}_{k;2,2} & 0 \\ \hline \widehat{C}_{k;1} & \widehat{C}_{k;2} & D_k \end{array}\right]. \tag{7.15}$$

The state $\widehat{x}_{k;2}$ (at each k) will always remain zero, because there is no input that can change it (at least when it started out zero in the first place: it could lead a life all by itself, of course.). Hence it can be cancelled out and we have found a reduced realization for the system, which is reachable:

$$\left[\begin{array}{c|c} \widehat{A}_{k;1,1} & \widehat{B}_{k;1} \\ \hline \widehat{C}_{k;1} & D_k \end{array}\right]. \tag{7.16}$$

As before, one would execute the reduction to a reachable system using a square-root algorithm, directly on the state transformations R_k:

$$\left[\begin{array}{cc} A_k R_k & B_k \end{array}\right] = \left[\begin{array}{c|c} R_{k+1;1} & 0 \end{array}\right] \left[\begin{array}{cc} Q_{1,1} & Q_{1,2} \\ Q_{2,1} & Q_{2,2} \end{array}\right] \tag{7.17}$$

for some Q still to be identified. To avoid ambiguities with pseudo-inverses, let us now work exclusively with Moore-Penrose inverses. E.g., as $R_{k:1}$ has full column rank (for each k), we have $R_{k;1}^{\dagger} = (R'_{k;1}R_{k,1})^{-1}R'_{k;1}$. To complement $R_{k;1}$, one then chooses $R_{k;2}$ columnwise orthogonal on $R_{k;1}$, i.e., $R'_{k,1}R_{k;2} = 0$, and then one has

$$\begin{bmatrix} R_{k;1} & R_{k;2} \end{bmatrix}^{\dagger} = \begin{bmatrix} R_{k;1} & R_{k;2} \end{bmatrix}^{-1} = \begin{bmatrix} R_{k;1}^{\dagger} \\ R_{k;2}^{\dagger} \end{bmatrix} \tag{7.18}$$

which provides for coherence between the pseudo-inverses. Actually, we do not have to compute the $R_{k;2}$ at all, since

$$\widehat{A}_{k;1,1} = R_{k+1;1}^{\dagger}A_kR_{k;1}; \quad \widehat{B}_k = R_{k+1;1}^{\dagger}B_{k;1}; \quad \widehat{C}_k = C_kR_{k;1} \tag{7.19}$$

are solely dependent on $R_{k,1}$ and $R_{k+1;1}$.

A dual operation will reduce the non-minimal state-space realization to one that is observable. This is immediate, because observability is reachability in the dual system and vice-versa, of course with time reversal. Again, the reduction can be computed using a square-root algorithm, this time progressing backward in time—we leave the details to the reader. When a minimal realization is desired, then one must proceed in two steps: first a reduction to a reachable system with a forward recursion, followed by a reduction to an observable system on the already reduced reachable system. It turns out that together this will result in a minimal system (the second reduction to observable does not destroy the reachability.). A detailed analysis of the overall result can be found in [10], p. 102.

7.5 LTI systems*

The main difference between an LTV and an LTI system is the fact that for the latter one needs fixed point solutions when solving recursive equations. Otherwise, algebraic manipulations on the state space are the same. E.g., when figuring out the reachability gramian, instead of recursing $M_{k+1} = B_kB'_k + A_kM_kA'_k$, one has to find M so that

$$M = BB' + AMA'. \tag{7.20}$$

This is a set of linear equations in the entries of M, which can be solved directly, but at the cost of numerical accuracy, as we know already. Because of the importance of this equation, let us investigate further. When A is strictly stable, i.e., all its eigenvalues are strictly inside the unit complex disk, then the unique positive definite solution of this equation is the reachability gramian

$$M = \sum_{k=0}^{\infty} A^k BB'(A')^k. \tag{7.21}$$

Thanks to the strict stability, $A^k \to 0$ exponentially and the sum converges (there can be solutions in the non strictly stable case, but these go beyond the present treatment. In that case the sum will not converge and the equations have to be solved directly.).

There are several methods to solve for M that are more efficient or stable than a direct inversion of the linear equations (7.20), but none of them is without some flaws, unfortunately. There are two main difficulties with the evaluation. First, the equation is a linear equation in δ^2 unknown, when $\delta \times \delta$ are the dimensions of A, so solving the equation in a traditional way requires $\mathcal{O}(\delta^6)$ operations, which is forbidding in applications where A is large. Secondly, when eigenvalues of A are close to 1 in magnitude, then the system becomes ill-conditioned and the series solution converges slowly as well. The various alternative methods deal with these issues in different ways.

Direct evaluation in square root form

One way to stabilize the system of Lyapunov-Stein equations (7.20) is by converting A to its Schur form (there are numerically stable and reasonably efficient algorithms to do so). Let $A = USU'$ with U a unitary matrix and S its lower triangular Schur form (which has the eigenvalues of A on its main diagonal), then eq. (7.20) transforms to

$$\widehat{M} = \widehat{B}\widehat{B}' + S\widehat{M}S' \tag{7.22}$$

with $\widehat{M} = U'MU$ and $\widehat{B} = U'B$. Suppose S, $\widehat{B}$ and hence $\widehat{M}$ are all scalars, then the equation reduces to $(1 - |S|^2)\widehat{M}^2 = |\widehat{B}|^2$ and $\widehat{M} = (1 - |S|^2)^{-1}|\widehat{B}|$. In the matrix case, a recursion can be set up on the "square root version" of this equation. Because of its independent interest, we give an algorithm in appendix of this chapter.

The procedure has been proven to be numerically stable, see the bibliographical notes for further references. The important point is that it is a square-root procedure, inheriting its numerical stability properties. The disadvantage of this direct method is the necessity to compute the Schur form of the state transition matrix A, which, in case of even moderately large systems can easily be prohibitive and would be hard to streamline on a parallel processor. The procedures to be discussed next do not have that problem, however at the cost of being iterative.

Iterative algorithms

From eq. (7.20) and its solution eq. (7.21) it is immediately obvious that an iterative computation of the solution M is possible. This can be done either with a linear iteration:

$$M_{k+1} = BB' + AM_kA' \tag{7.23}$$

with starting matrix $M_1 := BB'$, or a "doubling" iteration

$$M_{2k} = M_k + A^kM_k(A')^k \tag{7.24}$$

also with the same starting value, yielding $M = \lim_{k\to\infty} M_k$ (in the second case the k's will be powers of 2). The problem with these simple recursions is that they compute M rather than a square-root L of M with $M = LL'$. However, it is equally easy to see that these recursions can be converted to recursions on the square-root of the L-Q type:
for the linear iteration:

$$\begin{bmatrix} B & AL_k \end{bmatrix} = \begin{bmatrix} L_{k+1} & 0 \end{bmatrix} Q_k \tag{7.25}$$

and for the doubling iteration:

$$\begin{bmatrix} L_k & A^kL_k \end{bmatrix} = \begin{bmatrix} L_{2k} & 0 \end{bmatrix} Q_k \tag{7.26}$$

both with starting value $L_1 = B$ and appropriate L_k and Q_k (caution: not the same in each formula), and where k goes up with powers of 2 in the doubling case.

Although these are attractive formulas, they may be inconvenient. The linear iteration may converge very slowly (although it eventually converges exponentially, assuming A stable of course; with large matrices it may be problematic to reach that regime). The doubling iteration converges much faster but involves the computation of the matrix $(A^k)^2$ for k's that are powers of 2, which in some cases may also be a clumsy and intensive operation. Be that as it may, the doubling procedure in square-root form is a simple, effective and numerically stable operation (and might best be done by first converting A to its Schur form!). Improvements on the iterations are possible—but for that we refer to the bibliographic notes.

7.6 Infinitely indexed LTV systems*

One issue still to be settled about solving the Lyapunov-Stein equation recursively, is the question of a starting value (so far we assumed a cold, empty start.). Although this question cannot be settled in general (it is type-of-system dependent), we can state some important properties of the initial condition. Concentrating on the reachability gramian and the forward equation: $M_{k+1} = B_k B_k' + A_k M_k A_k'$, and assuming some starting value M_{k_0} is known, we see that its contribution in the solution for M_k with $k > k_0$ is given by $A^{<}_{k_0-1,k} M_{k_0} (A^{<}_{k_0-1,k})'$, which goes exponentially to zero for $k \to \infty$ when the system is u.e.s. (remember: $A^{<}_{k_0-1,k} := A_{k_0} \cdots A_{k-1}$.). So, for a u.e.s. system the initial condition does not matter if one starts early enough.

For systems whose behavior at $-\infty$ is known in the case of the reachability gramian, or $+\infty$ in the case of the observability gramian, much more can be said. In that case, the initial value can be computed by solving a fixed-point equation, and the recursion starts as soon as the system becomes time-varying—of course, also in this case the initial value will not matter much if the system remains stable (u.e.s.).

The observations just made apply for the respective square-root algorithms as well—proof of this is maybe a bit too technical: we skip it.

7.7 Discussion topics

- *The Lyapunov-Stein recursion is not invertible.* The forward Lyapunov-Stein recursion for the reachability gramian will be unconditionally numerically stable when the state transition operator A is u.e.s. (considering the LTV case). This means that any error made early on in the recursion will die down when progressing with it. The inverse proposition does not hold: the Lyapunov-Stein recursion cannot be inverted in that case. In fact: because of this property, the eventual values of the gramian become fully independent of the initial values after a few steps. Conversely: the initial values cannot be determined from later values.

- Balanced model reduction: based on the balanced form, and at each index point, an approximate (local) system with a smaller amount of states can be derived by neglecting small singular values in the joint diagonal reachability/observability gramian. Would it be possible to derive error bounds that describe how well the behavior of the reduced system approximates the original?

7.8 Notes

- The determination of the dimension of a collection of vectors of the same dimension and the calculation of a basis (often an orthonormal basis) is often a key step in a computation for digital signal processing. Important and often used algorithms have been developed for that purpose. We mention in order of precision: QR, rank-revealing QR and the singular value decomposition (SVD), the latter being the method of choice when the highest possible accuracy is desired because of its inherent numerical stability when executed well. However, in many practical cases QR may suffice (but it remains up to the design engineer to decide whether accuracy is sufficient, of course). For reference to these techniques, basic books on Numerical Analysis may suffice, see e.g., [15, 43, 34, 35].

- In the case of LTV system identification and state space reduction, the issue is little bit more involved, as it concerns not just one estimation of a single basis, but a recursive set of such: for each index point one typically needs a basis for the actual reachability or observability space at that point. The Lyapunov-Stein equation provides the link between the subsequent stages, and it can be solved with a recursive QR- or SVD algorithm. In the LTI case the situation is even more involved, because then a fixed-point solution has to be found. Many methods for this have been proposed, and we have treated a few salient ones in this chapter. For a survey of what the numerical community proposes in this respect, see [32]. An alternative, and often effective way is by the doubling procedure and its derivatives, proposed and much exploited by Kailath and his group [24].

- In the subsequent chapters, we shall see that the Lyapunov-Stein equation plays a central role in Dynamical System Theory. Actually, this distinction should be awarded to the square-root algorithm that solves the inner-outer factorization problem, which we shall treat at length in chapter 10, but there also, Lyapunov-Stein plays a role on the background. This is not surprising, in the view of the importance of reachability and observability in characterizing the dynamics of the system and the derivation of state space realizations. This importance will of course extend to control, optimization and synthesis problems, also to be treated in further chapters.
- *Using the sign matrix:* In the control literature, another method to solve the Lyapunov-Stein equation has become popular: the sign method. More information on the use of this method in our context can be found in [29].

7.9 Appendix: Square-root algorithm for the fixed-point Lyapunov-Stein equation based on the Schur form*

We give a brief summary on how to find the square-root fixed point solution of the LTI Lyapunov-Stein equation based on the Schur eigenvalue form of the transition operator A, both because of the importance of the issue and the interest of the algorithm itself (which may not be so well known). The equation to be solved is:

$$M = BB' + SMS' \tag{7.27}$$

and we assume that S is in lower Schur eigenvalue form—i.e., S lower triangular with of course its eigenvalues on the main diagonal (in case the original A does not have that form, it can be converted to it via a unitary state transformation: $AU = US$ with U unitary, and B must then be adapted as well. Finding U and S is done with what is known as a numerically stable algorithm.). We want to find a 'square-root' L of M, i.e., a lower triangular matrix such that $M = LL'$. The recursion is possible thanks to the following observation. Suppose a block subdivision of S and L as $S = \begin{bmatrix} S_{1,1} & 0 \\ S_{2,1} & S_{2,2} \end{bmatrix}$ and $L = \begin{bmatrix} L_{1,1} & 0 \\ L_{2,1} & L_{2,2} \end{bmatrix}$ with both $S_{1,1}$ and $L_{1,1}$ square of dimension $k \times k$, and conformally $B = \begin{bmatrix} B_1 \\ B_2 \end{bmatrix}$, we find equally

$$L_{1,1}L'_{1,1} = B_1B'_1 + S_{1,1}L_{1,1}L'_{1,1}S'_{1,1} \tag{7.28}$$

so that one can move from stage k in the recursion to stage $k+1$ by adding a new row to B_1, $S_{1,1}$ and $L_{1,1}$ and solving just for the last row of the latter. This can be done in square-root form, as we show now. We have generally

$$\begin{bmatrix} B & SL \end{bmatrix} \begin{bmatrix} B' \\ L'S' \end{bmatrix} = \begin{bmatrix} L & 0 \end{bmatrix} \begin{bmatrix} L' \\ 0 \end{bmatrix} \tag{7.29}$$

where L has been augmented with zeros to match the dimensions of $\begin{bmatrix} B_1 & SL \end{bmatrix}$ and hence there exists (at least one) unitary matrix (A may have complex eigenvalues!) such that

$$\begin{bmatrix} B & SL \end{bmatrix} = \begin{bmatrix} L & 0 \end{bmatrix} \begin{bmatrix} Q_{1,1} & Q_{1,2} \\ Q_{2,1} & Q_{2,2} \end{bmatrix} \tag{7.30}$$

in which the columns of $\begin{bmatrix} Q_{1,1} & Q_{1,2} \end{bmatrix}'$ form an orthonormal basis for the co-range (corresponding to the row space) of $\begin{bmatrix} B & SL \end{bmatrix}$ and the columns of $\begin{bmatrix} Q_{2,1} & Q_{2,2} \end{bmatrix}'$ for its kernel (which the array that has generated Q so far produces automatically).

To keep notation simple, let us now just add a row (and adjoining column where needed) to the last equation, indicated with lower case symbols—the situation being typical for the move from k to $k+1$, with the new S written as $\begin{bmatrix} S & 0 \\ s_1 & s_2 \end{bmatrix}$, the new L as $\begin{bmatrix} L & 0 \\ \ell_1 & \ell_2 \end{bmatrix}$, and the new Q as $\widehat{Q} = \begin{bmatrix} Q_{1,1} & Q_{1,2} & 0 \\ \widehat{Q}_{2,1} & \widehat{Q}_{2,2} & \widehat{Q}_{3,1} \\ \widehat{Q}_{3,1} & \widehat{Q}_{3,2} & \widehat{Q}_{3,3} \end{bmatrix}$ with fitting dimensions:

$$\begin{bmatrix} B & SL & 0 \\ b & s_1L + s_2\ell_1 & s_2\ell_2 \end{bmatrix} = \begin{bmatrix} L & 0 & 0 \\ \ell_1 & \ell_2 & 0 \end{bmatrix} \begin{bmatrix} Q_{1,1} & Q_{1,2} & 0 \\ \widehat{Q}_{2,1} & \widehat{Q}_{2,2} & \widehat{Q}_{3,1} \\ \widehat{Q}_{3,1} & \widehat{Q}_{3,2} & \widehat{Q}_{3,3} \end{bmatrix} \tag{7.31}$$

Notice that in this new expression the original $\begin{bmatrix} Q_{1,1} & Q_{1,2} \end{bmatrix}$ can be retained, because an orthonormal basis for $\begin{bmatrix} B & SL \end{bmatrix}$ is still needed, but now augmented with a scalar 0, while a new row is added to accommodate the new data and the kernel is accordingly modified as well. Notice also that the entries s_2 and ℓ_2 are just scalar, and ℓ_2 in particular has to be non-zero if the pair $\{A, B\}$ is reachable (for the more general non-minimal case, additional considerations must be made, which we skip here for brevity.). In this expression b, s_1 and s_2 are known as well as $Q_{1,1}$ and $Q_{1,2}$, while ℓ and the $\widehat{Q}$-entries have to be determined (as before, they will follow automatically from the LQ-array, that we now proceed to update.).

Postmultiplying with the hermitian conjugate of the $\widehat{Q}$-array (assuming we know it) we find

$$\begin{bmatrix} B & SL & 0 \\ b & s_1L + s_2\ell_1 & s_2\ell_2 \end{bmatrix} \begin{bmatrix} Q'_{1,1} & \widehat{Q}'_{2,1} & \widehat{Q}'_{3,1} \\ Q'_{1,2} & \widehat{Q}'_{2,2} & \widehat{Q}'_{3,2} \\ 0 & \widehat{Q}'_{2,3} & \widehat{Q}'_{3,3} \end{bmatrix} ?=? \begin{bmatrix} L & 0 & 0 \\ \ell_1 & \ell_2 & 0 \end{bmatrix} \tag{7.32}$$

From the first block column (which has k columns) we obtain

$$bQ'_{1,1} + s_1LQ'_{1,2} + s_2\ell_1Q'_{1,2} = \ell_1 \tag{7.33}$$

in which the only unknown is the (row) ℓ_1. Since s_2 is scalar (it is an eigenvalue of S), we find hence

$$\ell_1 = (bQ'_{1,1} + s_1LQ'_{1,2})(I - s_2Q'_{1,2})^{-1} \tag{7.34}$$

and good arguments can be made for the existence of the inverse, in the non-singular case (normally, both s_2 and $Q_{1,2}$ are strictly contractive—but it is enough that s_2 is in the open unit disc!).

From this point on, the normal LQ-factorization algorithm applies: one passes the newly added row $\begin{bmatrix} b & s_1L + s_2\ell_1 & s_2\ell_2 \end{bmatrix}$, through the orthogonal array build so far (still leaving ℓ_2 to be determined), and the result then has to be followed by a compression of the columns $k + 1$ to the end—standard procedure for an LQ-factorization. More precisely, when passing the partially known $\begin{bmatrix} b & s_1L + s_2\ell_1 & s_2\ell_2 \end{bmatrix}$ through the already existing array $\begin{bmatrix} Q'_{1,1} & Q'_{2,1} & \\ Q'_{1,2} & Q'_{2,2} & \\ & & 1 \end{bmatrix}$, out comes $\begin{bmatrix} \ell_1 & bQ'_{2,1} + (s_1L + s_2\ell_1)Q'_{2,2} & s_2\ell_2 \end{bmatrix}$, and the new array layer has to orthogonalize the part $\begin{bmatrix} x & s_2\ell_2 \end{bmatrix}$ with

$x = bQ'_{2,1} + (s_1 L + s_2 \ell_1)Q'_{2,2}$ known. The compression boils down to produce $\begin{bmatrix} x & s_2\ell_2 \end{bmatrix} = \ell_2 \begin{bmatrix} q_{1,1} & q_{1,2} \end{bmatrix}$ with $\begin{bmatrix} q_{1,1} & q_{1,2} \end{bmatrix}$ co-isometric. Using a positive choice for ℓ_2, this requires $\ell_2 = xx'/\sqrt{1-|s_2|^2}$, which determines ℓ_2 as well as the next layer in the array. The new contribution to the array forms a unitary matrix $\begin{bmatrix} q_{1,2} & q_{1,2} \\ q_{2,1} & q_{2,2} \end{bmatrix}$, and the updated transformation matrix becomes

$$\widehat{Q} = \begin{bmatrix} I & & \\ & q_{1,1} & q_{1,2} \\ & q_{2,1} & q_{2,2} \end{bmatrix} \begin{bmatrix} Q_{1,1} & Q_{1,2} & \\ Q_{2,1} & Q_{2,2} & \\ & & 1 \end{bmatrix} = \begin{bmatrix} Q_{1,1} & Q_{1,2} & 0 \\ q_{1,1}Q_{2,1} & q_{1,1}Q_{2,2} & q_{1,2} \\ q_{2,1}Q_{2,1} & q_{2,1}Q_{2,2} & q_{2,2} \end{bmatrix} \tag{7.35}$$

All recursive quantities have now been updated. Although the implicit determination of the missing data for a straight LQ factorization may seem a bit cumbersome, only L and $Q_{1,2}$ are $k \times k$ matrices, all the other quantities (matrices) have typically a small size (depending on the size of B of course). So the order of computation at each step is a bit more than $\mathcal{O}(k^2)$.

Chapter 8

Elementary operations, special matrices

One of the main purposes to consider semi-separable systems or systems described by state-space realizations is that such descriptions lead to efficient algorithms, i.e. algorithms that are linear rather than quadratic (or even worse) in the overall dimension of the transfer operator. In this chapter we show how elementary matrix operations, namely matrix-vector multiplication, addition of matrices and multiplication of matrices, are done using state space representations. More complex operations, such as matrix inversion, low complexity approximation of matrices and the computation of matrix eigenvalues have to wait to later chapters, but they also will have essentially the same property: numerical efficiency and preferably 'linear complexity' in the overall size of the transfer operator.

Menu

Hors d'oeuvre

Algebraic minimality

First course

Matrix-vector multiplication

Second course

Reduction to minimal

Addition of semi-separable systems

Third course
Multiplication

Cheese dish
Elementary inversion

Dessert
Inner functions

8.1 Algebraic minimality

Before starting on the efficiency path, one very important issue has to be dealt with first, and that is that the state space realizations given so far are not algebraically minimal. To illustrate the point: suppose you have a scalar LTI system with (irreducible) transfer function

$$T(z) := d + \frac{b_n z^n + \cdots + b_1 z}{a_n z^n + \cdots + a_1 z + 1} \tag{8.1}$$

then a minimal realization in companion form would be

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \left[\begin{array}{ccccc|c} 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ \ddots & \ddots & \ddots & \cdots & \vdots & \\ 0 & 0 & 0 & \cdots & 1 & 0 \\ -a_n & -a_{n-1} & -a_{n-1} & \cdots & -a_1 & 1 \\ \hline b_n & b_{n-1} & b_{n-1} & \cdots & b_1 & d \end{array}\right] \tag{8.2}$$

(it is a good exercise to show that this realization is ok!—hint: use LU-factorization to find $(I - zA)^{-1}$), and we see that the algebraically free parameters in the realization correspond exactly to the free parameters in the transfer function—in other words: *the realization is algebraically minimal*, by which is meant that the number of free parameters (coefficients) in the transfer operator corresponds exactly to the number of free parameters in the realization, which would not be the case when the matrix A were a full matrix. From the point of view of computational complexity this is optimal. However, it is probably far from optimal from an accuracy or sensitivity point of view (in particular, it is well known that pole location is very sensitively dependent on the companion form, especially when the poles are close to each other as is often the case in practice.). Whether better forms exist that are also algebraically minimal is an interesting and important issue that we shall discuss in a later chapter (Chapt. **??**): it is actually the gist of high performance system and circuit design as well. In this chapter we shall not consider this non-trivial point further, and just assume that we dispose of realizations that have the additional useful properties needed by the concrete problem at hand. If they are indeed algebraically minimal, additional computational benefits will ensue.

8.2 Matrix-vector multiplication

Let us start with an LTV causal realization $T = D + C(I - ZA)^{-1}ZB$, assuming that one wants to compute $y = Tu$, with u starting at some point k_0 in time (i.e., empty before k_0). Then one can simply put the recursion $x_{k+1} = A_k x_k + B_k u_k,\ \ y_k = C_k x_k + D_k u_k$ to work starting at $k = k_0$ with $x_{k_0} = -$ of appropriate dimension[1], and keeping on recurring. If the dimensions of A_k at any point k are $\delta_{k+1} \times \delta_k$, of u_k then the number of multiplications (and additions) will roughly be $\delta_{k+1}\delta_k + n_k(\delta_k + m_k - 1)$, i.e., square in the state dimension and linear in the input and output dimensions (in a more optimized case it could be linear in the state dimension.).

In case of a double sided matrix with a causal and an anticausal realization: $T = D + C_c(I - ZA_c)^{-1}ZB_c + C_a(I - Z'A_a)^{-1}Z'B_a$, the multiplication splits in a forward recursion (for the causal part) and a downward recursion (for the anti-causal part), totally independently from each other except for multiplication with the constant term, which of course has to be done only once.)

8.3 Adding two matrices

Let us start with the causal case. Let $T_1 \sim_c \begin{bmatrix} A_1 & B_1 \\ C_1 & D_1 \end{bmatrix}$ and $T_2 \sim_c \begin{bmatrix} A_2 & B_2 \\ C_2 & D_2 \end{bmatrix}$, then a potentially non-minimal realization for $T := T_1 + T_2$, assuming match-

[1]The overall x is a column stack of the individual x_k: $x = \mathrm{col}(x_k)_{k=-\infty}^{+\infty}$, hence an empty x_k is a 0×1 entry.

ing input and output dimensions, would be

$$T \sim_c \left[\begin{array}{cc|c} A_1 & 0 & B_1 \\ 0 & A_2 & B_2 \\ \hline C_1 & C_2 & D_1 + D_2 \end{array}\right] \tag{8.3}$$

i.e., the global state is the concatenation of the two states. However, this new state may not have minimal dimension, and might hence be reduced. As we saw in the previous chapter, the reduction is based on reducing the reachability and observability gramians to non-singular. Suppose that we already know the individual gramians of T_1 and T_2, then the new gramians can be computed with less effort, as follows.

Let's look at the reachability gramian (the observability case is just dual). Suppose $M_{1,k}$ and $M_{2,k}$ are the respective local reachability gramians of T_1 and T_2 and $M_{t,k}$ the local reachability gramian of $T_1 + T_2$, then we find

$$M_{tk} = \begin{bmatrix} M_{1,k} & M_{t,k;1,2} \\ M'_{t,k;1,2} & M_{2,k} \end{bmatrix} \tag{8.4}$$

in which the entry $M_{t,k;1,2}$ satisfies the Lyapunov-Stein equation $M_{t,k+1;1,2} = B_1 B_2' + A_1 M_{t,k;1,2} A_2'$. Hence, solving just one more Lyapunov-Stein equation produces the reachability gramian of the sum (this can of course be done with a square-root algorithm, we leave details for the discussion.). For example, in case $T_1 = T_2$ we would have for the new reachability gramian $\begin{bmatrix} M_1 & M_1 \\ M_1 & M_1 \end{bmatrix}$, which is of course singular with the same dimensions as the original.

The theory generalizes easily to double sided matrices as the causal and anti-causal parts can be treated independently from each other.

8.4 Matrix-matrix multiplication

The next case would be $T = T_2 T_1$, assuming that the output dimensions of T_1 match the input dimensions of T_2, and let us first look at the causal case, i.e., assuming both T_1 and T_2 causal. It is easy to see that a realization for T is again obtained by concatenating the states and is given by

$$\begin{bmatrix} A_t & B_t \\ C_t & D_t \end{bmatrix} := \left[\begin{array}{cc|c} I & 0 & 0 \\ 0 & A_2 & B_2 \\ \hline 0 & C_2 & D_2 \end{array}\right] \left[\begin{array}{cc|c} A_1 & 0 & B_1 \\ 0 & I & 0 \\ \hline C_1 & 0 & D_1 \end{array}\right] = \left[\begin{array}{cc|c} A_1 & 0 & B_1 \\ B_2 C_1 & A_2 & B_2 D_1 \\ \hline D_2 C_1 & C_2 & D_2 D_1 \end{array}\right] \tag{8.5}$$

and, again, this realization may not be minimal (that will happen when there are 'cancellations' between the 'numerators' and the 'denominators', but these notions are not really useful anymore in our LTV context.).

Again, the overall gramians will have to be reduced if one wants to obtain a minimal realization for the product. Here also, some additional efficiency may be obtained from the original gramians, but not as much as in the previous case (we leave details for the discussion.).

For general systems with both causal and anticausal parts the situation is more complex: we get four terms in the product: causal times causal, causal times anti-causal, anti-causal times causal and anti-causal times anti-causal. As we know already how to deal with causal times causal, and the anti-case is just similar, let's now look at e.g., anti-causal times causal: $T = T_2 T_1$ where $T_1 = C_1(I - ZA_1)^{-1}ZB_1$ and $T_2 = C_2(I - Z'A_2)^{-1}Z'B_2$ (the constants D_1 and D_2 do not contribute to the mixed form, they can best be taken with respect. the causal and anti-causal part of the originals.). The result is worth a lemma:

Lemma 1 Decomposition by parts lemma. *Let A_1 and A_2 be uniformly exponentially stable, C_1 and B_2 (conformal) bounded diagonal operators, then*

$$(I - Z'A_2)^{-1}Z'B_2C_1(I - ZA_1)^{-1}Z = (I - Z'A_2)^{-1}Z'A_2M + M + MA_1(I - ZA_1)^{-1}Z \tag{8.6}$$

where M is the unique diagonal operator $M = \mathrm{diag}[M_k]$, which satisfies the Lyapunov-Stein equation

$$M_{k+1} = B_{2,k}C_{1,k} + A_{2,k}M_kA_{1,k}. \tag{8.7}$$

Proof

By premultiplication with $(I - Z'A_2)$, post-multiplication with $(I - A_1Z)$, and using $Z(I - A_1Z)^{-1} = (I - ZA_1)^{-1}Z$, check that the first equation reduces to

$$ZMZ' = B_2C_1 + A_2MA_1 \tag{8.8}$$

which is exactly the Lyapunov-Stein recursion given. All these operations are legal because, even in the case of infinitely indexed systems, the inverses exist as bounded operators[2], and the Lyapunov-Stein equation has a unique solution thanks to the u.e.s. assumption. QED

[2]See chapter **??**.

It follows that the mixed term then decomposes additively in a strictly causal, constant and strictly anti-causal term as

$$C_2(I - Z'A_2)^{-1}Z'A_2MB_1 + C_2MB_1 + C_2MA_1(I - ZA_1)^{-1}ZB_1 \qquad (8.9)$$

Similarly, a product of a causal with an anti-causal term will decompose additively, but now a backward Lyapunov-Stein equation will have to be solved. Altogether one gets a lot of terms in the results, which may appear to be a bit messy. We shall see in the chapter on inversion, that in many practical situations the decomposition by parts comes in very handily.

8.5 Elementary inversion: arrow reversal

Returning to the causal case, let us consider a system $T = D + C(I - ZA)^{-1}ZB$ in which D is square and invertible. (In that case the input and output dimensions are equal.) An easy inverse is obtained just by "arrow reversal":

$$\left\{ \begin{array}{rcl} x_{k+1} & = & A_k x_k + B_k u_k \\ u_k & = & -D_k^{-1}C_k x_k + D_k^{-1} y_k \end{array} \right. \qquad (8.10)$$

see fig. (5.4). In the schema for the inverse, u_k is computed first, and then used to update the state, all with roughly the same computational complexity as for the original system (except the computation of D_k^{-1}).

This way of proceeding may seem extremely simple and obvious, but there is a serious problem: the resulting inverse system may not be stable, and will indeed not be stable in many cases, leading to erroneous results. To see this, eliminate u_k from the first equation to obtain the actual state space realization of the inverse system:

$$\begin{bmatrix} A_k - B_k D_k^{-1} C_k & B_k D_k^{-1} \\ -D_k^{-1} C_k & D_k^{-1} \end{bmatrix} \qquad (8.11)$$

and we see the Schur complement of D_k appearing as state transition matrix of the inverse system. There is no reason why this matrix should be u.e.s. In the LTI case (where this theory is equally valid), it turns out that the eigenvalues of $\Delta := A - BD^{-1}C$ correspond to the zeros of the system, and these may have arbitrary locations (to be precise: if a is an eigenvalue of Δ, then $1/a$ is a zero of T.). So this means that the elementary realization of the inverse is only computationally valid when Δ is in some sense stable, e.g., when Δ is u.e.s. In such a case, T is said to be *outer*—in the next paragraphs we shall meet what we shall call 'inner' operators, which will play an important role in the inversion theory of the subsequent chapters.

8.6 Inner matrices

As in almost all numerical analysis, isometric and unitary matrices play an important role, think about QR-factorization or the SVD. That is also the case with semi-separable or LTV systems, not to talk about circuit theory, where the notion of 'losslessness' plays that role. The concept of 'inner' has to do with the combination of causal and isometric, co-isometric and/or unitary. To avoid confusion later on, it is important to distinguish these cases. Here are the definitions:

Definition 5 *We call an operator left-inner, when it is causal and isometric. Right-inner when it is causal and co-isometric. Bi-inner (or just inner) when it is causal and unitary. Similar definitions hold for an anti-causal operator, in which case it is called left co-inner (or left conjugate inner), right co-inner or co-inner respectively.*

Our interest will of course mostly go to such operators which are also semi-separable, or, equivalently, which have finite state space realizations. The following theorem summarizes the important properties relating the operator to its possible realizations.

Theorem 2 *A semi-separable left-inner operator has a minimal realization that is isometric and u.e.s.. Dually, a semi-separable right-inner operator has a minimal realization that is co-isometric and u.e.s.. A semi-separable operator that is inner has a minimal realization that is unitary and u.e.s.. Conversely, a causal system that has an isometric and u.e.s. realization is left-inner. It will be right-inner if it is u.e.s. and has a co-isometric realization, and bi-inner if it is u.e.s. and has a unitary realization.*

Proof

The full proof of this theorem is somewhat delicate and has to be relegated to the chapter on infinitely indexed systems. Let us suffice here to give a number of relevant indications that make the result plausible. Actually, the property is easy to prove in the case of finite matrices, where the issue of a minimal realization being u.e.s. is automatically fulfilled. So let us start with that case, and consider the left-inner case.

So, let $T'T = I$ semi-separable, and let us consider a minimal realization for T in output normal form (that is, with all the observability operators $\mathbf{O}_k$ isometric or $\mathbf{O}_k'\mathbf{O}_k = I$ for all k). This choice already makes the pair $\begin{bmatrix} A_k \\ C_k \end{bmatrix}$ isometric, since $\mathbf{O}_k = \begin{bmatrix} C_k \\ \mathbf{O}_{k+1}A_k \end{bmatrix}$, and hence

$$\mathbf{O}_k'\mathbf{O}_k = I = C_k'C_k + A_k'\mathbf{O}_{k+1}'\mathbf{O}_{k+1}A_k = C_k'C_k + A_k'A_k. \tag{8.12}$$

Next, consider the South-West (or left-bottom) corner of the transfer operator T which contains D_k as its right-upper top element:

$$\begin{bmatrix} C_k\mathbf{R}_k & D_k \\ \mathbf{O}_{k+1}A_k\mathbf{R}_k & \mathbf{O}_{k+1}B_k \end{bmatrix} \tag{8.13}$$

The last column has to be isometric and orthogonal on all the previous, due the isometry of T. It then follows:
(1) $D_k'D_k + B_k'\mathbf{O}_{k+1}'\mathbf{O}_{k+1}B_k = D_k'D_k + B_k'B_k = I$ and
(2) $(D_k'C_k + B_k'\mathbf{O}_{k+1}'\mathbf{O}_{k+1}A_k)\mathbf{R}_k = 0$
and hence also $B_k'A_k + D_k'C_k = 0$, making the realization fully isometric, because we assumed the realization to be minimal, and hence $\mathbf{R}_k$ right invertible (it forms a row basis.). The converse property, namely that a minimal isometric realization of a finitely indexed operator makes the operator isometric is even easier, and is left as an exercise (hint: conservation of "energy" along the realization). All the other properties claimed are *mutatis mutandis* shown in a similar way.

What now with infinitely indexed systems? The difficulty is that now energy can disappear at infinity: so there may be local conservation of energy, but not global. This actually happens when the state transition operator is not u.e.s. We discuss this matter at length in the chapter on Hilbert space theory for infinitely indexed systems (chapter **??**). QED

A consequence of the previous is that a left-inner (respect. right-inner) operator can be embedded into an inner one, just by augmentation of the local isometry to unitary. More precisely: suppose $\begin{bmatrix} A & B \\ C & D \end{bmatrix}$ is an isometric realization of a left-inner operator (it has to be u.e.s.!), then there exist

B_2 and D_2 such that $\begin{bmatrix} A & B & B_2 \\ C & D & D_2 \end{bmatrix}$ is unitary, thereby creating a second transfer operator T_2 with realization $\begin{bmatrix} A & B_2 \\ C & D_2 \end{bmatrix}$ such that $T_t := \begin{bmatrix} T & T_2 \end{bmatrix}$ is inner. These issues will be discussed in more detail in chapter **??** using a "geometric" framework.

8.7 Topics for discussion

- Suppose you add two identical causal systems. How can the direct realization be reduced? Suppose you multiply them. Can the direct realization still be reduced? Under what conditions would a product of two systems be reducible?

- Z is an inner factor, maybe the most trivial one except for unitary diagonal matrices. It is interesting to study its matrix representation in a number of cases.

- Finite dimensional inner factors: a lower triangular finitely indexed matrix with scalar entries is necessarily diagonal (do you see that?). What are the consequences for inner factors?

- A question, which one may already raise at this point is whether a system that has a contractive input-output map (which we called 'behavior') has a contractive realization (the converse should be pretty easy to show). We shall consider this somewhat delicate issue in a further chapter (chapter **??**, where we shall show that this is indeed the case, and we shall derive a method to determine such a realization.).

8.8 Notes

- Although much of the material in this chapter is pretty standard, a couple of issues deserve highlighting. First, there is the LTV 'decomposition by parts lemma': this is the key technical property that makes much of the whole theory work. It allows, in this precise setting, for 'dichotomy': the splitting of a mixed causal-anticausal term in a causal and an anticausal part—an operation that is bread-and-butter for system theory.).

Next, on 'elementary inversion': one should resist the temptation to think that D_k non-singular is a necessary condition for system inversion. The shift operator Z has the simple inverse Z' and illustrates the fact that a causal (bounded) operator may very well have a bounded anticausal inverse. When a bounded, causal system has a bounded *causal* inverse, then it will belong to the class of 'outer' systems. In the case of finitely indexed systems, the existence of the inverse D_k (and hence also that D_k is square) is necessary and sufficient for the system to be outer. In the infinitely indexed case, the situation is much more complicated.

- It should also be apparent from the section on inner matrices, that such matrices always have a unitary anti-causal inverse, which could rightly be called anti-inner or inner for the shift Z'. In the more general case, system inversion requires figuring out what the causal and anti-causal parts of the inverse are—this will be a topic that we shall consider in detail.

Chapter 9

External and coprime factorizations

This and the next chapter are devoted to classical factorization theory in the non-classical setting of time-variant systems. Factorizations have played a major role in the development of system theory, and they remain important, because they not only represent a major type of matrix operation, but also because they are instrumental in solving many problems in control theory and numerical algebra, as shall be demonstrated in many of the subsequent chapters. We start out in this chapter with what is traditionally called 'coprime factorization', that is the representation of a causal system as the ratio of two matrix factors, one of which (the denominator) characterizes the dynamic behavior of the system. We consider two important cases: one in which the denominator is an inner operator (i.e., causal isometric or unitary), and one in which the denominator is polynomial in the shift Z. These two characterizations have different uses, which we discuss to some extent. The polynomial factorization involves an operation called 'deadbeat control', which is perhaps the simplest possible and hence most fundamental control action one might imagine, and is therefore important already just from that point of view. Since, strictly speaking, we are not only allowing factorizations that are coprime, we have termed this type of factorization 'external', in contrast to the factorization to be considered in the next chapter, known as inner-outer factorization, which will play an equally fundamental but complementary role in solving system theoretical problems like estimation and tracking theory.

Menu

Hors d'oeuvre

External factorizations with inner denominators.

First course

LTV representations as ratios of matrix polynomials.

Second course

The LTI case.

Dessert

Historical notes

9.1 External factorizations with inner denominators

Given a causal and semi-separable LTV system T, we already figured out that we could determine realizations by choosing matching bases for the reachability and the observability spaces. E.g., we could choose an orthonormal basis for each reachability space (i.e., the range of H_k' at each index k), determine the corresponding observability space and a realization according to the precepts of chapter 6. This produces A, B, C and D in input normal form, i.e. such that $AA' + BB' = I$. The orthonormal bases for the reachability spaces are given by $\mathbf{R} := B'Z'(I - A'Z')^{-1}$, and the causal part of $T\mathbf{R}$ is given by (with Π_f projection on the present and future):

$$\Pi_f \left[\left(D + C(I - ZA)^{-1}ZB\right) B'Z'(I - A'Z')^{-1}\right] = C(I - ZA)^{-1} \qquad (9.1)$$

the response one gets with zero as present and future inputs, the latter being the matching observability bases, given by $\mathbf{O}$ (notice that in this case

$$(I - ZA)^{-1}ZBB'Z'(I - A'Z')^{-1} = (I - ZA)^{-1} + A'Z'(I - A'Z')^{-1}.) \quad (9.2)$$

With $\left[\begin{array}{cc} A & B \end{array}\right]$ co-isometric, one can easily find matrices C_U and D_U such that

$$U \sim_c \left[\begin{array}{cc} A & B \\ C_U & D_U \end{array}\right] \qquad (9.3)$$

is unitary. In the matrix case, the corresponding causal operator $U = D_U + C_U(I - ZA)^{-1}ZB$ will be unitary (as shown in the previous chapter) and hence inner. Let us now consider $TU^{-1} = TU'$:

$$\begin{array}{rcl} TU' & = & (D + C(I - ZA)^{-1}ZB)\,(D_U' + B'Z'(I - A'Z')^{-1}C_U') \\ & = & (DB' + CA')Z'(I - A'Z')^{-1}C_u' + (DD_U' + CC_U') \\ & & +C(I - ZA)^{-1}Z(AC_U' + BD_U') \\ & = & (DB' + CA')Z'(I - A'Z')^{-1}C_U' + (DD_U' + CC_U') \end{array} \tag{9.4}$$

where the second equation is obtained through decomposition in parts and the third by using the orthogonality of the realization for U. Hence, $\Delta' := TU^{-1}$ is anti-causal (Δ is defined causal) and

$$T = \Delta' U = \Delta'(U')^{-1} \tag{9.5}$$

Hence: we have represented the causal T as the ratio of two anti-causal operators, the right one of which is unitary. This is what we call a *right external factorization*, often called *right coprime factorization* in the literature, although the latter term requires coprime-ness between the factors, which is not necessary for such a factorization to exist, but is often assumed. In fact: U is an inner factor that pushes T to anti-causality from the right, and it is actually 'minimal' in doing so, meaning in this context that it has the smallest possible state dimension at each index k to do so (to actually show this we need some more theory, which we do in chapter **??**.).

Similarly, and starting from an output normal form, we could obtain a *left external factorization*. This would produce $T = V\Delta_\ell'$ with V unitary and Δ_ℓ' anti-causal (Δ_ℓ causal). One starts out, dually, with a realization in output normal form and $\begin{bmatrix} A \\ C \end{bmatrix}$ isometric. Again, one may complete the isometric basis to unitary, thereby defining $V = D_V + C(I - ZA)^{-1}ZB_V$, and work out the product

$$\begin{array}{rcl} V'T & = & (D_V' + B_V'Z'(I - A'Z')^{-1}C')\,(D + C(I - ZA)^{-1}ZB) \\ & = & B_V'(I - ZA')^{-1}Z'(A'B + C'D) + (D_V'D + B_V'B) \\ & & +(D_V'C + B_V'A)(I - ZA)^{-1}ZB \\ & = & B_V'Z'(I - A'Z')^{-1}(A'B + C'D) + (D_V'D + B_V'B) \end{array} \tag{9.6}$$

where the causal term disappears again because of the unitarity of V and its realization. To find the two canonical external forms one puts the realization in input, respect. output, normal form, which in each case requires the solution of a Lyapunov-Stein equation, a forward, respect. a backward one.

Does all this have any significance? Let's investigate a bit more. Consider the case for U. Let $\Pi_{f,k}$ be 'projection on present and future' at index point k, and consider an input v such that $\Pi_{f,k}v = 0$—i.e., $v_\ell = 0$ for $\ell \geq k$. Next, take as input $u := U'v$, then we have as output $y := Tu = \Delta' v$ and hence $\Pi_{f,k}y = 0$, an output signal that lives in the strict past w.r. k, just like $u = U'v$ since by construction, also $\Pi_{f,k}U'v = 0$. Conversely, if $y = Tu$ is such that both $\Pi_{k,f}y = 0$ and $\Pi_{k,f}u = 0$, then there is a v with $\Pi_{f,k}v = 0$ such that $u = Uv$. In plain words: U' characterizes the co-kernel of the Hankel operator at each position k. Dually, V characterizes the kernel of the Hankel operator, through a similar reasoning, now on the dual system T', for which H' is the (anti-causal) Hankel operator, whose co-kernel is the kernel of H.

9.2 Fractional polynomial representations: the LTV case

In this section we develop an alternative to external factorization with a conjugate inner function in the denominator, namely a factorization theory for a causal semi-separable transfer operator or matrix as a ratio of two minimal polynomials in the shift Z (or dually Z'). Such polynomials represent lower triangular matrix with a staircase form, i.e., matrices with zero entries as soon as a certain (entry dependent) distance from the main diagonal is reached, and such that the support of non-zero entries form a staircase. Such matrices can be represented by polynomials in the shift operator, assuming that there is a uniform bound on the allowed non-zero distance (which will always be the case for finite matrices, but might be violated in the infinite indexed case). We shall of course aim at minimal representations. It is a remarkable fact that such representations do exist in the time-variant case, and that they can easily be derived as well, providing for an alternative external factorization theory, much in the same spirit as the celebrated polynomial representations for the LTI case.

The key to generating polynomial representations is a simple method called 'dead beat control'. Let us assume that we dispose of a minimal system realization $\{A, B, C, D\}$ (now with diagonal operators), and let us position ourselves at some index k. Consider now the problem to generate, from the index point k on, a set of inputs u_k aiming at bringing the state x_k at k to zero in as few steps as possible. If successful, this will produce a control law that 'beats the state to death' as fast as possible. Before deriving the general law, let us make a couple of observations.

First of all, any state x_k that belongs to the kernel of A_k does not need to be beaten to zero, A_k already does that in the present stage k. Next, suppose the state x_k belongs to the *pre-image for A_k of the range of B_k*[1], then there will be inputs u_k such that $x_{k+1} = A_k x_k + B_k u_k = 0$, namely $u_k := -B_k^+ A_k x_k$, where B_k^+ is any pseudo-inverse of B_k; hence $u_k = -F_{k,1}x_k$ with $F_{k,1} = B_k^+ A_k$. (This can be seen to be true, since by definition of pre-image there exists a $\widehat{u}_k$ such that $A_k x_k = -B_k \widehat{u}_k$, and hence $A_k x_k + B_k u_k = (I - B_k B_k^+)A_k x_k = -(I - B_k B_k^+)B_k \widehat{u}_k = 0$, since $B_k B_k^+ B_k = B_k$.) We can compute the necessary pre-image using the following, almost evident, lemma[2]

Lemma 2 *A vector x belongs to the pre-image of B by a matrix X iff for some vector y, $\begin{bmatrix} x \\ -y \end{bmatrix}$ belongs to the kernel of $\begin{bmatrix} X & B \end{bmatrix}$.*

Let now ξ_0 be a basis for the kernel of A_k, let (ξ_0, ξ_1) be a basis for the pre-image by A_k of B_k, and consider a full RQ-factorization of the pair

$$\begin{bmatrix} A_k & B_k \end{bmatrix} = \begin{bmatrix} 0 & R \end{bmatrix} \begin{bmatrix} Q_{1,1} & Q_{1,2} \\ Q_{2,1} & Q_{2,2} \end{bmatrix} \tag{9.7}$$

then all $x_k \in \bigvee Q'_{1,1}$ will be in $\bigvee(\xi_0, \xi_1)$. Moreover, $\bigvee\left(\xi_0, Q'_{1,1}\right) = \bigvee(\xi_0, \xi_1)$ and, for any $x_k \in \bigvee(\xi_0, \xi_1)$, the corresponding $u_k = -B_k^+ A_k x_k$ for any B_k^+.

This procedure can be made recursive. Let us concentrate first on the recursive generation of the basis for the pre-images by A_k. Suppose that in the next stage $k+1$, x_{k+1} can be beaten to death by some u_{k+1} in just one step, then x_{k+1} belongs to the pre-image by A_{k+1} of the range of B_{k+1}, and all such x_{k+1} form a subspace—let η be a basis for it (notice: $\bigvee \eta$ contains the kernel of A_{k+1}). Now consider the pre-image by A_k (call it $\mathcal{S}$) of the sum

[1]Suppose $a : \mathcal{X} \to \mathcal{Y}$ is a map from $\mathcal{X}$ to $\mathcal{Y}$ then a pre-image of any element $y \in \mathcal{Y}$ for a is an element $x \in \mathcal{X}$ such that $y = ax$. In the case of linear maps, the notion extends to spaces: if $\mathcal{S}_+$ is a subspace of $\mathcal{Y}$, then its pre-image for a is the largest subspace $\mathcal{S}$ such that $ax \in \mathcal{S}_+$ for all $x \in \mathcal{S}$.

[2]Given two subspaces $\bigvee \xi$ and $\bigvee \eta$, then the easiest way to compute their intersection is by finding the orthogonal complement of the sum of their orthogonal complements: $\bigvee(\xi) \cap \bigvee(\eta) = (\bigvee(\xi, \eta))^{\perp}$.

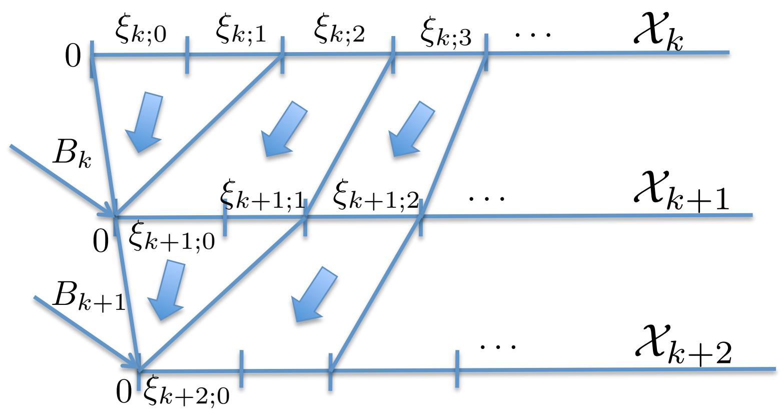


Figure 9.1: States that can be beaten to death in two stages.

of the spaces 'range of B_k' and $\bigvee \eta$ (i.e., $\mathcal{S} = \text{preim}_{A_k}[\bigvee(B_k, \eta)]$), then for any $x_k \in \mathcal{S}$ there shall be an input u_k that maps any vector in $\mathcal{S}$ to $\bigvee \eta$. In stage $k+1$, this vector can then be beaten to death by the procedure of the previous paragraph! See fig 9.1 for an illustration. The procedure to obtain a basis for $\mathcal{S}$ extends the procedure of the previous paragraph, as follows. We aim at obtaining an RQ factorization of the type

$$\begin{bmatrix} A_k & B_k & \eta \end{bmatrix} = \begin{bmatrix} 0 & R^{\text{new}} \end{bmatrix} Q^{\text{new}} \tag{9.8}$$

and this recursively, since we already have a partial result, eq. 9.7. Starting from eq. 9.7, rewriting $\begin{bmatrix} 0 & R \end{bmatrix}$ as $\begin{bmatrix} 0_1 & R \end{bmatrix}$ to keep track of the dimensions of the kernels, and producing an RQ factorization

$$\begin{bmatrix} R & \eta \end{bmatrix} = \begin{bmatrix} 0_2 & R^{\text{new}} \end{bmatrix} \begin{bmatrix} q_{1,1} & q_{1,2} \\ q_{2,1} & q_{2,2} \end{bmatrix} \tag{9.9}$$

we combine the two factorizations to obtain

$$\begin{bmatrix} A_k & B_k & \eta \end{bmatrix} = \begin{bmatrix} 0_1 & 0_2 & R^{\text{new}} \end{bmatrix} \begin{bmatrix} Q_{1,1} & Q_{1,2} & 0 \\ q_{1,1}Q_{2,1} & q_{1,1}Q_{2,2} & q_{1,2} \\ q_{2,1}Q_{2,1} & q_{2,1}Q_{2,2} & q_{2,2}. \end{bmatrix} \tag{9.10}$$

It follows that the new contribution to the state space $\mathcal{X}_k$ of states that can be beaten to death, now in two steps, is given by $\bigvee(Q'_{2,1}q'_{1,1})$, while we already had $\bigvee(\xi_0, \xi_1) = \bigvee(\xi_0, Q'_{1,1})$. We can now determine a new basis component ξ_2, by the rule $\bigvee(\xi_0, \xi_1, \xi_2) = \bigvee(\xi_0, Q'_{1,1}, (Q'_{2,1}q'_{1,1}))$ and reducing to a basis at each step. The result of this step follows from applying A_k to ξ_2 is (anticipating somewhat on the notation to come)

$$A_k x_k = B_k \hat{F} + \eta \widehat{A}_f \tag{9.11}$$

with

$$\begin{bmatrix} \widehat{F} \\ \widehat{A}_f \end{bmatrix} = -\begin{bmatrix} B_k & \eta \end{bmatrix}^+ A_k \tag{9.12}$$

for any pseudo-inverse $\begin{bmatrix} B_k & \eta \end{bmatrix}^+$, just as in the previous step, but now $\begin{bmatrix} B_k & \eta \end{bmatrix}$ replacing B_k, and the arguments remaining the same (the pseudo-inverse can be obtained through simple QR factorization or RLS). And the procedure can be repeated, adding another η_2 to $\begin{bmatrix} A_k & B_k & \eta \end{bmatrix}$ using the newly obtained Q^{new}.

To get the global result, let us now define at each time point a basis $\xi_k := \begin{bmatrix} \xi_{k,0} & \xi_{k,1} & \cdots & \xi_{k,c_k} \end{bmatrix}$[3] for the state space $\mathcal{X}_k$ at index k, such that 1., $\xi_{k,0}$ spans the kernel of A_k, 2., $[\xi_{k,0}, \xi_{k,1}]$ spans the subspace that A_k maps to the span of $\bigvee(\xi_{k+1,0}, B_k)$, etc... or, in short: $\xi_{k;0:i} := \begin{bmatrix} \xi_{k,0} & \xi_{k,1} & \cdots & \xi_{k,i} \end{bmatrix}$ spans the subspace of the state at time k that lies in the pre-image of $\bigvee[\xi_{k+1,0}, B_k, \cdots \xi_{k+1,1:i-1}]$ by A_k and hence can be brought to zero by adequate inputs in at most i steps.

At issue is now whether *any state*, at all indices k, can be beaten to death in a finite number of steps or, in other words, whether for each k there exists a c_k such that $\xi_k := \bigvee \begin{bmatrix} \xi_{k,0} & \xi_{k,1} & \cdots & \xi_{k,c_k} \end{bmatrix}$ spans the whole state space $\mathcal{X}_k$, and this at each index point k. The answer to this question follows from the input-controlled state evolution, and we state this as a separate proposition. To state the proposition comfortably, let's use the *continuing product* notation: $A^{>}_{k+c,k} := A_{k+c-1} \cdots A_k$ for an integer $c > 1$.

Proposition 2 *Any state x_k at any index k of the system described by $x_{k+1} = A_k x_k + B_k u_k$ can be brought to zero in c_k steps, if and only if for each k*

$$\bigvee \begin{bmatrix} A^{>}_{k+c_k-1} B_k & \cdots & B_{k+c_k-1} \end{bmatrix} \subset \bigvee A^{>}_{k+c_k,k}. \tag{9.13}$$

A sufficient condition for this is that the system is reachable at any index k.

[3]In this chapter, we adopt the following notation: a basis for a space $\mathcal{S}$ is a matrix ξ whose columns form the basis; hence $\mathcal{S} = \bigvee \xi$, where the symbol '$\bigvee$' stands for 'span'.

Proof

The proof is immediate from the global input-output relation, starting from a state x_k and using inputs $u_k, \cdots, u_{k+c-1}$ for some integer $c \geq 1$:

$$x_{k+c} = A^{>}_{k+c,k} x_k + \left[\begin{array}{ccc} A^{>}_{k+c-1} B_k & \cdots & B_{k+c-1} \end{array} \right] u_{k:k+c-1} \tag{9.14}$$

and a reasoning similar as before. The sufficiency claim follows from the fact that if the system is fully reachable then the partial reachability matrices $\left[\begin{array}{ccc} A^{>}_{k+c-1} B_k & \cdots & B_{k+c-1} \end{array} \right]$ have to reach full rank for some c. For c one chooses the minimal value c_k at each k (in the case of an infinite number of indices, one might require the 'horizon' c_k to be uniformly bounded). QED

Definition 6 *We call a system controllable iff condition 9.13 is satisfied for some c_k at all k.*

We see that the controllability condition is weaker than the reachability condition. (This is typical for discrete time systems. In continuous time systems driven by an ordinary differential equation, the two conditions coincide.)

Now consider the bases for the state at time points k and $k+1$. As A_k maps $\xi_{k,i}$ to the span of $[B_k, \xi_{k+1,0}, \xi_{k+1,1}, \cdots, \xi_{k+1,i-1}]$, we shall have, for some matrices $\widehat{A}_{f;k,i}$ and $\widehat{F}_{k,i}$ (for $i = 0$ we just take $\widehat{F}_{k,0} = 0$ when $\xi_{k,0}$ is not empty)

$$A_k \xi_{k;i} = \xi_{k+1;0:i-1} \widehat{A}_{f;k,i} + B_k \widehat{F}_{k,i} \tag{9.15}$$

with

$$\left[\begin{array}{c} \widehat{F}_{k;i} \\ \widehat{A}_{f;k,i} \end{array} \right] = - \left[\begin{array}{cc} B_k & \xi_{k+1;0:i-1} \end{array} \right]^{+} A_k. \tag{9.16}$$

Taking full bases ξ_k, ξ_{k+1} and stacking the vectors from $i = 0$ to c_k we get globally

$$A_k \xi_k = \xi_{k+1} \widehat{A}_{f;k} + B_k \left[\begin{array}{cccc} \widehat{F}_{k;0} & \widehat{F}_{k;1} & \cdots & \widehat{F}_{k;c_i} \end{array} \right] := \xi_{k+1} \widehat{A}_{f;k} + B_k \widehat{F}_k \tag{9.17}$$

Assuming full bases for all state spaces $\mathcal{X}_i = \bigvee \xi_i$, for which then ξ_k and ξ_{k+1} are invertible, we find, with $A_{f;k} := \xi_{k+1} \widehat{A}_{f;k} \xi_k^{-1}$ and $F_k := \widehat{F}_k \xi_k^{-1}$

$$A_k = \xi_{k+1} \widehat{A}_{f;k} \xi_k^{-1} + B_k \widehat{F}_k \xi_k^{-1} := A_{f,k} + B_k F_k \tag{9.18}$$

in which, in the ξ-basis,

$$\widehat{A}_{f;k} = \begin{bmatrix} 0 & \widehat{A}_{f,k;0,1} & \widehat{A}_{f,k;0,2} & \cdots & \widehat{A}_{f,k;0,c_k} \\ 0 & 0 & \widehat{A}_{f,k;1,2} & \cdots & \widehat{A}_{f,k;1,c_k} \\ 0 & 0 & 0 & \cdots & \widehat{A}_{f,k;2,c_k} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix}, \ \widehat{F}_k = \begin{bmatrix} 0 & \widehat{F}_{k;1} & \cdots & \widehat{F}_{k;c_k} \end{bmatrix} \tag{9.19}$$

with both $A_{f;k}$ and $\widehat{A}_{f;k}$ turn out to be nilpotent because they map any state x_k to zero in a finite number of steps.

The computation of the ξ_k as well as $A_{f,k}$ and F_k goes, in a first pass, by a determination of pre-images by a backward recursion: assume knowledge of ξ_{k+1}, and with a second, now local recursion, compute a basis, in sequence, first $\xi_{k;0}$ for the kernel of A_k, then an additional basis $\{\xi_{k;0}, \xi_{k;1}\}$ for the pre-image by A_k for $\bigvee\{\xi_{k+1,0}, B_k\}$, and then a further additional basis $\{\xi_{k;0}, \xi_{k;0}, \xi_{k;1}\}$ for the pre-image by A_k of $\{B_k, \xi_{k+1,0}, \xi_{k+1;1}\}$ etc... This can be done in one shot by RQ-factorization executed in the right order, as we already argued before (the full algorithm, although straightforward, gets a bit unwieldy). The second pass consists simply in the recursive determination of a pseudo-inverse, e.g., by QR factorization (the RLS algorithm). These two steps are unrelated to each other, as they involve different types of basis determinations (viewed from the position $k+1$, the first pass looks backward, while the second pass looks forward, so that the two passes involve unrelated data.).

Ratios of polynomials

The deadbeat construction annihilates any state x_k in at most c_k steps. This means, in particular, that the diagonal block matrices $Z\widehat{A}_f$ and, equivalently by state equivalence, ZA_f are nilpotent, because, for each k and each x_k, the continuous product $A_{f,k+c_k} \cdots A_{f,k+1} A_{f,k} x_k = 0$, and hence also $A_{f,k+c_k} \cdots A_{f,k+1} A_{f,k} = 0$. Now consider the operator

$$P \sim_c \begin{bmatrix} A - BF & B \\ -F & I \end{bmatrix}. \tag{9.20}$$

It is polynomial in Z! And its inverse

$$P^{-1} \sim_c \begin{bmatrix} A & B \\ F & I \end{bmatrix} \tag{9.21}$$

is a causal system, with causal (because polynomial) inverse! Let us now define $\Delta := TP$, then

$$\Delta = \left[D + C(I - ZA)^{-1}ZB\right]\left[I - F(I - ZA_f)^{-1}ZB\right] = D{+}C - DF(I{-}ZA_f)^{-1}ZB \tag{9.22}$$

(by direct expansion) and hence

$$\Delta \sim_c \left[\begin{array}{cc} A - BF & B \\ C - DF & D \end{array}\right] \tag{9.23}$$

is itself an operator which is polynomial in Z, and we have obtained an external representation of the original operator T as a ratio of two polynomials in Z: $T = \Delta P^{-1}$.

Bezout identities for the LTV case

Bezout identities play an important role in determining the properties of the factors in the just derived external factorizations. In particular, they are helpful in proving co-prime properties of the factors, by which is meant that the factors do not have common, non-trivial divisors. These properties are then further exploited in deriving various control laws, a topic that we leave for later. Let $T = \Delta_r P_r^{-1}$, with $P_r \sim_c \left[\begin{array}{cc} A_f & B \\ -F & I \end{array}\right]$, $A_f := A - BF$ nilpotent and $\Delta_r \sim_c \left[\begin{array}{cc} A_f & B \\ C - DF & D \end{array}\right]$ and let a dual (left) factorization be $T = P_\ell^{-1}\Delta_\ell$, $P_\ell \sim_c \left[\begin{array}{cc} A_g & -G \\ C & I \end{array}\right]$, with $A_g := A - GC$ nilpotent and $\Delta_\ell \sim_c \left[\begin{array}{cc} A_g & B - GD \\ C & D \end{array}\right]$ (such a factorization is similarly obtained by working on the rows instead of the columns, or, equivalently, on the transpose without time reversal). Now consider the joint polynomial matrix

$$\left[\begin{array}{c} -\Delta \\ P \end{array}\right] \sim_c \left[\begin{array}{c|c} A_f & B \\ \hline -(C - DF) & -D \\ -F & I \end{array}\right] \tag{9.24}$$

and let this matrix be completed so that its inverse is polynomial as well. This defines two new polynomial matrices M and N by

$$\left[\begin{array}{cc} M & -\Delta \\ N & P \end{array}\right] :\sim_c \left[\begin{array}{c|cc} A_f & -G & B \\ \hline -(C - DF) & I & -D \\ -F & 0 & I \end{array}\right] \tag{9.25}$$

The state transition matrix of the inverse of this operator is now

$$A_f - \begin{bmatrix} -G & B \end{bmatrix} \begin{bmatrix} I & D \\ 0 & I \end{bmatrix} \begin{bmatrix} -(C-DF) \\ -F \end{bmatrix} = A_g, \tag{9.26}$$

hence also nilpotent, and we find as realization for the inverse

$$\left[\begin{array}{c|cc} A_g & -G & B-GD \\ \hline C & I & D \\ F & 0 & I \end{array}\right] \sim_c \begin{bmatrix} P_\ell & \Delta_\ell \\ R & S \end{bmatrix}. \tag{9.27}$$

The LTV *Bezout relations* follow:

$$\begin{bmatrix} P_\ell & \Delta_\ell \\ R & S \end{bmatrix} \begin{bmatrix} M & -\Delta \\ N & P \end{bmatrix} = \begin{bmatrix} I & \\ & I \end{bmatrix} \tag{9.28}$$

or, in detailed form:

$$\begin{cases} P_\ell M + \Delta_\ell N &= I \\ -R\Delta + SP &= I \end{cases} \tag{9.29}$$

From these it follows that P_ℓ and Δ_ℓ are *left-coprime* in the sense that any common left polynomial factor in Z has to be *uni-modular*, i.e., has to have a polynomial inverse as well, and dually for P and Δ, which have to be *right-coprime*. To put it differently: there cannot be a meaningful cancellation in the factorization $P_\ell^{-1}\Delta_\ell$ nor in ΔP^{-1}, reflecting the minimality of the factorizations.

9.3 Polynomial representations for LTI systems*

LTI systems are not fundamentally different from the matrix case, some simplifications *and* some complications occur as usual. In a nutshell: the Hankel operators at each index point are equal, but they have infinite indices, so that orthonormal base vectors have infinite indices as well, but finite dimension if the system has a finite-dimensional state space. Nonetheless, we know that we can obtain an $\{A, B, C, D\}$ realization from a restricted version, after which the realization can be converted to input normal form (respect. output normal form) by solving $P = BB' + APA'$ for the reachability gramian P (respect. $Q = C'C + A'QA$ for the observability gramian)—preferably in square root form, see chapter 7 on how to do that. These fixed point Lyapunov-Stein equations are solvable under broad conditions, but in order to obtain converging bases for the reachability and controllability spaces, we have to ask A to be strictly stable. Here, A is just a constant matrix and the notion of strict stability (i.e., all eigenvalues of A are strictly inside the unit disc of the complex plane) and uniform exponential stability (u.e..s.) coincide.

Therefore, the question arrises: *what can be done when the stability condition is not satisfied?* In that case one cannot reasonably speak of a decent Hankel operator without making further assumptions. Previously, our strategy was to restrict the Hankel operator to finitely indexed submatrices, but that is what one could call an 'ad hoc' solution. It may be that the fixed point Lyapunov-Stein equation is still solvable, so that either an operator P or Q is well defined, although not obtainable through a series development, and, moreover, they may (or will) not be positive definite anymore. For a scalar example: let $B = 1$ and $A = 2$, then we would have $P = 1 + 4P$ and hence $P = -1/3$—there is no positive definite solution. From the theory of solving fixed point Lyapunov-Stein equations (see the Mathematical Introduction) we know that the Lyapunov-Stein system of equations will be non-singular iff the eigenvalues λ_k satisfy the condition $1 - \lambda_i \overline{\lambda_k} \neq 0$ for all relevant i and k. If they do not, there is likely no solution—as can already be seen by the simple example $A = 1$ and $B = 1$.

Hence, a different approach is called for to handle unstable systems, generally defined as systems for which the input-output (behavioral), or equivalently, Hankel map is unbounded. We already know how to derive realizations for such systems, but we now want to develop a 'system theory' for them that, like in the stable u.e.s. case, characterizes reachability and observability spaces. It turns out that this can be elegantly done with polynomial representations. In the remainder of this section we give a complete account of the LTI-theory, using the same method as for the LTV case. The approach we present is therefore different from the classical approach based on module theory, because the latter does not extend to LTV systems (they do not generate modules in a straightforward way). The 'dead beat control' method has the advantage to produce the desired forms directly, using numerical algebra, rather than indirectly, using algebraic properties of rings and modules. The use of polynomial representations is 'natural' in the unstable context, because it only handles one-sided series or operators (series or operators whose support does not extend to $-\infty$), so that they can multiply each other meaningfully.

Let us then consider *causal* discrete-time LTI systems, and assume that they have a well-defined response for every input with a finite time support. Let the input dimension (which is now constant over all time) be m, output dimension n, and let e_i be the i^{th} natural vector in $\mathcal{R}^m$, i.e., $e_{i,k} = \delta_{i,k}$ for $k = 1 \cdots m$. Then $T_i := Te_i$ will be the i^{th} impulse response, which, because of causality, will only be non-zero from index $k = 0$ on. Writing this as $T_i(z) := \sum_{k=0}^{\infty} z^k T_{i;k}$ and stacking inputs and outputs we obtain the $n \times m$ transfer function $T(z) := \left[\begin{array}{ccc} T_1(z) & \cdots & T_m(z) \end{array}\right]$. All this is well defined, whether or not the system is stable—in case of an unstable system, the magnitude of the $T_{i,k}$ will keep on increasing with k, often exponentially. With some abuse of notation we can also write $T(z) = \sum_{k=0}^{\infty} z^k T_k$, the T_k now being constant $n \times m$ matrices (in each context it should be clear whether $T_i(z)$ or T_i is meant.).

The easiest way to construct an external factorization for this type of LTI transfer functions, assuming they possess a finite dimensional state space, is to use 'dead-beat control', based on a preliminary realization: we already know how to obtain a minimal $\{A, B, C, D\}$ realization from a finite version of a sufficiently large partial Hankel matrix

$$H_k := \text{Han}(T_1, T_2, \cdots, T_{2k-1}) := \left[\begin{array}{cccc} T_1 & T_2 & \cdots & T_k \\ T_2 & T_3 & \iddots & T_{k+1} \\ \vdots & \iddots & \iddots & \vdots \\ T_k & T_{k+1} & \cdots & T_{2k-1} \end{array}\right] \tag{9.30}$$

so let us assume that we have this realization available. We may then *define* $(I - zA)^{-1} := I + zA + z^2A^2 + \cdots$ and we shall have $T(z) = D + C(I - zA)^{-1}zB$ as a one-sided formal series in z, in which A^k may grow exponentially, depending on the location of the eigenvalues[4] of A.

From a given and reachable pair $\{A, B\}$, the dead-beat analysis detailed in the next subsection produces a matrix F such that $A - BF$ is nilpotent (i.e., such that there is an integer k for which $(A - BF)^k = 0$) and hence $P \sim_c \left[\begin{array}{cc} A - BF & B \\ -F & I \end{array}\right]$ is polynomial. Hence $P(z) = I - F(I - z(A - BF))^{-1}zB$

[4]One-sided series, for example series in z^k with $k \geq K$ for some K can be multiplied with each other, even when their coefficients become unbounded when $k \to \infty$, because the multiplication of two such series only involve finite computations of the convolution type.

(as a matrix polynomial in z) and we may compute, as formal series:

$$\begin{array}{rcl}\Delta(z) & := & T(z)P(z) = [D + C(I - zA)^{-1}zB]\,[I - F(I - z(A - BF))^{-1}zB] \\ & = & D + (C - DF)(I - z(A - BF))^{-1}zB \end{array} \tag{9.31}$$

so that also $\Delta(z)$ is polynomial and, again formally in one-sided series calculus, $T(z) = \Delta(z)P^{-1}(z)$. One can check directly that $P(z)^{-1} \sim_c \begin{bmatrix} A & B \\ F & I \end{bmatrix} = I + zFB + z^2FAB + \cdots := Q(z)$ formally, since $P(z)Q(z) = Q(z)P(z) = I$, a product of a finite series in z with a formal series in z, for which the computation of individual terms is finite and hence well defined. The same is also true for the formal product $\Delta(z)Q(z) = \Delta(z)P(z)^{-1}$. Hence we have a consistent z-series algebraic theory (a so-called *module*) and $T(z)$ has a fractional representation as $\Delta(z)P(z)^{-1}$, which we would call a right factorization, to be denoted henceforth as $\Delta_r(z)P_r(z)^{-1}$. Likewise, one may define a left factorization $T(z) = P_\ell(z)^{-1}\Delta_\ell(z)$. This factorization can also be expressed as an "external" product, e.g., $T(z) = (z^{-\kappa}\Delta_r(z))(z^{-\kappa}P_r(z))^{-1}$, where κ is chosen as the largest of the orders (largest exponent in z) occurring in either $P_r(z)$ or $\Delta_r(z)$. Note that the realizations given for the P's and the Δ's do not have to be minimal. E.g., if $T(z)$ is already polynomial, the P's would disappear, and likewise if $T(z)$ happens to be the formal inverse of a polynomial.

Dead-beat control

An adaptation of the method developed for the LTV case in the previous section gives also the LTI solution. The procedure reduces to finding the ξ's, A_f and F recursively, where now $\xi_k = \xi_{k+1}$. The strategy goes as follows: 1., ξ_0 is simply a basis for the kernel of A: any vector in $\bigvee \xi_0$ is killed in one step by A itself, and the corresponding $F_0 = 0$, while $A_{f;0} = 0$. 2., ξ_1 is a basis for a space of state vectors that can be brought to zero in one step using an input u such that, for $x \in \bigvee \xi_1$, $Ax + Bu = 0$; we get the control law $u = -F_1x$ with $F_1 = B^+A$. 2., in the next step, the same procedure is followed to produce a next extension ξ_2 to the state space basis one has been constructing, namely such that for $x \in \bigvee \xi_2 : Ax + F_2u + \xi_{0:1}A_{f;2} = 0$. The general step then extends $\xi_{0:i-1}$ to ξ_i and produces F_i and $A_{f;i}$ as before. We leave the details to the reader–see also ref. [12]!

When A is invertible, an alternative description is as follows: using A^{-1}, we have $\bigvee \xi_0 = 0$, $\bigvee\{\xi_0, \xi_1\} = \bigvee\{\xi_0, A^{-1}\begin{bmatrix} \xi_0 & B \end{bmatrix}\}$, $\bigvee \begin{bmatrix} \xi_0 & \xi_1 & \xi_2 \end{bmatrix} =$

$\bigvee\{\xi_0, \xi_1, A^{-1} \left[\begin{array}{cc} \xi_1 & B \end{array} \right]\}$ etc... To compute this, one has to recursively compute the increasing basis (actually orthonormality is not needed, but perhaps advisable for numerical reasons). Warning: the procedure may not work with A^+, because $AA^+(Bu + \xi_{0:i-1}\alpha)$ is not guaranteed to be a member of $\bigvee(B, \xi_{0:i-1})$. However, when $\{A, B\}$ is reachable we can indeed extend the procedure to A^+. For this we need a couple of properties:

Lemma 3 *Suppose $\{A, B\}$ is reachable, then the kernel $\mathcal{K}[A\ B] = 0$.*

Proof

For the reachability operator we have $\bigvee \mathcal{R} = \bigvee \left[\begin{array}{cc} A\mathcal{R} & B \end{array} \right]$ spanning the whole state space. Hence also $[A\ B]$ spans the whole state space, making the kernel zero. QED

Let us now consider the equation $Ax + Bu + \xi\alpha = 0$ needed to bring vectors in the state space to $\bigvee \xi$ using a control by B. Using the lemma on $[A_1\ \xi]$ with $A_1 = [A\ B]$ we find that any solution will be of the form

$$\left[\begin{array}{cc} A & B \end{array} \right] \left[\begin{array}{c} x \\ u \end{array} \right] + \xi\alpha = 0 \tag{9.32}$$

with

$$\left[\begin{array}{c} x \\ u \end{array} \right] = -\left[\begin{array}{cc} A & B \end{array} \right]^+ \xi\alpha. \tag{9.33}$$

Using reachability and the Moore-Penrose pseudo-inverse for convenience, we get

$$\left[\begin{array}{c} x \\ u \end{array} \right] = -\left[\begin{array}{c} A' \\ B' \end{array} \right] (AA' + BB')^{-1}\xi\alpha. \tag{9.34}$$

It follows that the basis ξ can now be further extended with additional base vectors generating $\bigvee A'(AA' + BB')^{-1}\xi$. The advantage of this procedure is that only the recursive computation of these subsequent base extensions is needed. Further elaboration of this procedure is left to the reader as a research project!

Example

Let $A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ and $B = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$. We see immediately that $\xi_0 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$. The second step requires the pre-image under A of $\begin{bmatrix} \xi_0 & B \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}$, which is immediately seen to yield $\xi_1 = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \\ 0 \end{bmatrix}$ by direct evaluation (find x, u and α such that $Ax + Bu + \xi_0\alpha = 0$). Hence $A\xi_1 = \xi_0\alpha_{0,1} + B(F\xi_1)$, giving $\begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}(-\frac{1}{\sqrt{2}}) + \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}(\frac{1}{\sqrt{2}})$, and hence $\alpha_{0,1} = -\frac{1}{\sqrt{2}}$ and $F\xi_1 = \frac{1}{\sqrt{2}}$. The final step produces easily $\xi_2 = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \\ 0 \end{bmatrix}$, $\alpha_{0,2} = 0$, $\alpha_{0,3} = 1$ and $F\xi_2 = 0$. Hence $F = F\xi\xi' = \begin{bmatrix} 0 & \frac{1}{\sqrt{2}} & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & -\frac{1}{2} & 0 \end{bmatrix}$, and one can check that the resulting $A - BF = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 \\ -\frac{1}{2} & -\frac{1}{2} & 0 \\ -\frac{1}{2} & \frac{1}{2} & 0 \end{bmatrix}$ is nilpotent of order 2. Applying the $\xi' \cdots \xi$ state transformation, we find in this case $\widehat{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$, $\widehat{B} = \begin{bmatrix} 1 \\ 0 \\ \sqrt{2} \end{bmatrix}$, $\widehat{F} = \begin{bmatrix} 0 & -\sqrt{2} & 0 \end{bmatrix}$ and $\alpha = \widehat{A} - \widehat{B}\widehat{F} = \begin{bmatrix} 0 & -\frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$, which is visibly nilpotent. The example shows that it is not easy to assess the form of A_f and BF a priori, both can be full matrices in general.

Bezout identities

In many classical treatments of LTI system theory and control, Bezout identities play an important role. Typically they are derived using a Euclidean algorithm that determines greatest common divisors, here we derive them just from the previous treatment (and the system theory we have developed so far). Using dead beat control, we have obtained the factorization $T = \Delta P^{-1}$, with $P \sim_c \begin{bmatrix} A_f & B \\ -F & I \end{bmatrix}$, $A_f := A - BF$ nilpotent and $\Delta \sim_c \begin{bmatrix} A_f & B \\ C - DF & D \end{bmatrix}$. Of course, a dual (left) factorization is equally possible, in which $T = P_\ell^{-1}\Delta_\ell$, $P_\ell \sim_c \begin{bmatrix} A_g & -G \\ C & I \end{bmatrix}$, $A_g := A - GC$ is nilpotent and $\Delta_\ell \sim_c \begin{bmatrix} A_g & B - GD \\ C & D \end{bmatrix}$ (such a factorization is obtained in the same way as before, but now working on the rows instead of the columns, or, equivalently, on the dual: $T' = \Delta_\ell' P_\ell^{-\prime}$ as functions of z', the time reversal being impliced in the prime). How are these two factorizations related to each other? One very direct but somewhat artificial approach goes as follows. Consider the joint polynomial matrix

$$\begin{bmatrix} -\Delta(z) \\ P(z) \end{bmatrix} \sim_c \left[\begin{array}{c|c} A_f & B \\ \hline -(C - DF) & -D \\ -F & I \end{array}\right] \tag{9.35}$$

(the minus sign on Δ is for later convenience, sorry for the anticipation!), and let us try to complete this matrix so that its inverse is polynomial as well. So let us define two new polynomial matrices M and N by putting

$$\begin{bmatrix} M(z) & -\Delta(z) \\ N(z) & P(z) \end{bmatrix} :\sim_c \left[\begin{array}{c|cc} A_f & -G & B \\ \hline -(C - DF) & I & -D \\ -F & 0 & I \end{array}\right] \tag{9.36}$$

The state transition matrix of the inverse is now

$$A_f - \begin{bmatrix} -G & B \end{bmatrix} \begin{bmatrix} I & D \\ 0 & I \end{bmatrix} \begin{bmatrix} -(C - DF) \\ -F \end{bmatrix} = A_g, \tag{9.37}$$

hence also nilpotent, and we find as realization for the inverse

$$\left[\begin{array}{c|cc} A_g & -G & B - GD \\ \hline C & I & D \\ F & 0 & I \end{array}\right] \sim_c \begin{bmatrix} P_\ell(z) & \Delta_\ell(z) \\ R(z) & S(z) \end{bmatrix}. \tag{9.38}$$

The famous *Bezout relations* follow:

$$\begin{bmatrix} P_\ell(z) & \Delta_\ell(z) \\ R(z) & S(z) \end{bmatrix} \begin{bmatrix} M(z) & -\Delta(z) \\ N(z) & P(z) \end{bmatrix} = \begin{bmatrix} I & \\ & I \end{bmatrix} \tag{9.39}$$

or, in detailed form:

$$\begin{cases} P_\ell(z)M(z) + \Delta_\ell(z)N(z) & = & I \\ -R(z)\Delta(z) + S(z)P(z) & = & I \end{cases} \tag{9.40}$$

From these it follows that $P_\ell(z)$ and $\Delta_\ell(z)$ are *left-coprime* in the sense that any common left polynomial factor has to be *uni-modular*, i.e., has to have a polynomial inverse as well, and dually for $P(z)$ and $\Delta(z)$, which have to be *right-coprime*. To put it differently: there cannot be a meaningful cancellation in the factorization $P_\ell(z)^{-1}\Delta_\ell(z)$ nor in $\Delta(z)P(z)^{-1}$, reflecting the minimality of the factorizations.

9.4 Discussion items

- Elementary examples: external factorizations are somewhat peculiar in our setup, in particular with semi-infinite or infinite indices, which are worth considering in an exploratory way. Look at related examples to see the peculiarities. For example, try the left and right external factorizations of the following causal matrices:

$$\begin{matrix} \begin{bmatrix} \boxed{1} & & & \\ 1/2 & 1 & & \\ & 1/2 & 1 & \\ & & 1/2 & 1 \end{bmatrix}, & \begin{bmatrix} \boxed{1} & & & \\ 1/2 & 1 & & \\ & 1/2 & 1 & \\ & & \ddots & \ddots \end{bmatrix}, \\ \begin{bmatrix} \boxed{1} & & & \\ 2 & 1 & & \\ & 2 & 1 & \\ & & 1 & 1 \end{bmatrix}, & \begin{bmatrix} \boxed{1} & & & \\ 2 & 1 & & \\ & 2 & 1 & \\ & & \ddots & \ddots \end{bmatrix} \end{matrix} \tag{9.41}$$

  where the second and last matrix are half infinite.

- Nerode equivalences with polynomials: consider the external factorization with polynomial matrices. How can such factorizations be interpreted in terms of 'Hankel type' maps, i.e., maps for strict past to future or strict future to past? What would be appropriate spaces to apply such maps on?

- An interesting issue is how to control the state optimally. The dead beat control is a minimum time control, but it seems to have major drawbacks as a control strategy. Which would those be? Much 'trajectory control' (e.g., the Appolo mission, or having an airplane keep to its planned trajectory) uses a 'differential model' i.e., a model where the state is actually the deviation from the nominal trajectory in the global state space. 'Keeping to the trajectory' is then trying to keep the deviation small. Often this has to be done with a limited energy budget. How can this be done? Let us discuss this point a bit further, anticipating somewhat on the chapter on optimal control.

  To begin: we should be sanguine about our model, in particular how it deals with 'energy', since the various components of both the input and the state may have different physical dimensions (for example: the state may be of the form $x = \begin{bmatrix} r \\ v \end{bmatrix}$ with r a position and v a velocity). It pays to normalize variables so that we, as engineers, know where we stand energy-wise (in the example, we may want to characterize the kinetic energy of the system as $\|v\|^2$, where v is then a normalized velocity $v := \sqrt{\frac{m}{2}}v_r$ with v_r the actual velocity). The same thing may happen with the inputs. We might assume that each input requires some energy, normalized as $\|u\|^2$ (using the Euclidean norm). With the state the situation may be more delicate: there we may have to use a 'semi-norm' to characterize the energy properties of the desired trajectory. For example: one may want to restrict $\|v\|^2$ only. Or else (maybe more to the point), the minimal input energy needed to bring x to zero (typically x would be a deviation from a nominal trajectory rather than the absolute position). In that case the state variable x and the operators $\{A, B\}$ would be chosen in such a way that $\|x\|^2$ represents that minimal input energy. Such issues may force a more delicate analysis than what we discuss in the next paragraph.

  So, let us assume that the goal of the control is to reduce $\|x\|^2$ using an input on which there is an energy limit at each step: $\|u_k\|^2 \leq L$ for each k—henceforth we just put $u := u_k$ for the control at a specific step k. Let us also assume 1., that B has full column range (otherwise one can reduce the input space) and 2., that we have already established that the one-step unconstrained minimal norm optimal control $u = -B^\dagger Ax$, with $B^\dagger$ the Moore-Penrose inverse of B, is too large with $\|u\|^2 > L$. What is then the one step minimal norm optimal control with $\|u\|^2 = L$? It would be given by solving a constrained optimization problem on the control u with Lagrangian

$$\mathcal{L} = (x'A' + u'B')(Ax + Bu) + \lambda(u'u - L) \tag{9.42}$$

and λ as Lagrange multiplier on the constraint (see the mathematical notes on optimization with Lagrangians).

Requiring $\nabla_u \mathcal{L} = 2B'Ax + 2B'Bu + 2\lambda u = 0$ at the optimal point, we find

$$u_{\text{opt}} = -(\lambda + B'B)^{-1}B'Ax \tag{9.43}$$

and the control law is now given by $-Fx$ with $F = (\lambda + B'B)^{-1}B'A$ instead of $B^\dagger A = (B'B)^{-1}B'A$. λ parametrizes the norm reduction, and it obviously reduces the input norm which has to be

$$\|u\|^2 = x'A'B(\lambda + B'B)^{-2}B'Ax = L. \tag{9.44}$$

At this point, one can start playing all sorts of control games. We see that the last expression is dependent on x—e.g., when $Ax = 0$ no control is needed, which we know already from the deadbeat control, and if $\|B^\dagger Ax\|^2 \le L$ then $u = -B^\dagger Ax$ would do as well.

However, often one wishes a control that works for all states within a certain range, say $\|x\| \le M$ (using Euclidean norms throughout). This one obtains by requiring $\|(\lambda + B'B)^{-1}B'A\| \le \sqrt{\frac{L}{M}}$, taking the smallest possible λ that satisfies this equation (the bigger λ, the smaller the norm. If B is just a vector and $\beta = B'B$ we could take $\lambda = \sqrt{\frac{M}{L}}\|B'A\| - \beta$ when positive.). How good is the solution then? One extra criterion that is often used, is to require stability of the system under the feedback law, so whatever F is chosen, one would require $A - BF$ to have all its eigenvalues inside the unit disc. This requirement is automatically satisfied for the deadbeat control (where all eigenvalues are zero), but needs extra attention in the more general case (it may even be that there is no solution, of course). Let's explore this issue further for the LTI case.

- **Companion form:** in classical or traditional control methods for single input–single output systems, the companion form for A and B in 'controller canonical form' corresponding to a transfer function

$$T(z) = d + \frac{c_{\delta-1}z^{-\delta+1} + \cdots + c_1 z^{-1} + c_0}{z^{-\delta} + a_{-\delta+1}z^{-\delta+1} + \cdots a_1 z^{-1} + a_0} \tag{9.45}$$

is given by

$$T(z) \sim_c \left[\begin{array}{ccccc|c} 0 & 1 & 0 & \cdots & 0 & 0 \\ & 0 & 1 & \ddots & 0 & 0 \\ & & \ddots & \ddots & \vdots & \vdots \\ & & & 0 & 1 & 0 \\ -a_0 & -a_1 & \cdots & \cdots & -a_{\delta-1} & 1 \\ \hline c_0 & c_1 & \cdots & \cdots & c_{\delta-1} & d \end{array}\right]. \tag{9.46}$$

It is pretty straightforward (and a good exercise) to do a deadbeat-control analysis on this form! The form is very popular with control engineers, because it allows easy *pole placement.* Using $F = \mathrm{col}[F_0, F_1, \cdots, F_{\delta-1}]$ we find

$$A_f = A - BF = \left[\begin{array}{cccc|c} 0 & 1 & 0 & \cdots & 0 \\ & 0 & 1 & \ddots & 0 \\ & & \ddots & \ddots & \vdots \\ & & & 0 & 1 \\ \hline -a_0 - F_0 & -a_1 - F_1 & \cdots & \cdots & -a_{\delta-1} - F_{\delta-1} \end{array}\right]. \tag{9.47}$$

For example: putting all $F_i = -a_i$ produces dead-beat control, and an arbitrary denominator for the controlled system is achieved by setting $F_i = -a_i + p_i$ for a desired characteristic polynomial $\chi_{A_f}(\lambda) = \lambda^\delta + p_{\delta-1}\lambda^{\delta-1} + \cdots + p_0$ of A_f. The approach can be generalized to LTI multiport systems thanks to the Heymann-Hautus lemma—see the literature on this matter [18]. Needless to say, working on the characteristic polynomial has its numerical problems (ill-conditioning of the roots), and is only suitable for low-dimensional problems. We postpone further discussions on 'optimal control' to chapter 12.

9.5 Notes

What we call ‘external factorization’ is usually called ‘coprime factorization’ in the literature. The reason to introduce a new term is that an external factorization as we conceive it does not have to be coprime in the usual algebraic sense—‘coprime’ means: the factors have no common divisor (in the matrix case one must distinguish between right and left divisors.). Coprime factorizations, and in extension, external factorizations have played an important role in the development of dynamical system theory, in the wake of the seminal book of Kalman, Falb and Arbib [26]. It was recognized early on that in the LTI case, rational matrix functions and in particular, polynomial matrices in a single variable z provide the algebraic framework needed to characterize important objects related to system theory, such as kernels, state characterization, state equivalence, canonical forms for the state transition operator etc... This realization provided a valuable link with the pre-state space approach, which was entirely based on rational matrix functions, and in particular the characterization of their dynamic properties through Smith-Macmillan forms and other algebraic devices based on algebraic structures called ‘rings’, ‘principal ideal domains’ and modules (as a multiport generalization of polynomial or rational algebras). All these lead to many results in electrical engineering, in particular in control theory and network theory.

Unfortunately, even though module theory produced many nice and important results, it does not provide the correct framework for time variant or non-linear systems (for which, as we shall see in further chapters, time variant theory plays an important role.). This is mainly due to the fact that the general shift operator Z does not commute with ‘instantaneous’ operators, which in our case are the diagonal operators A, B, C and D. Hence: no module, but, as already mentioned, an Arveson ‘nest algebra’. Remarkable now is that most algebraic properties needed still work in that setting, at the ’cost’ of using more elementary methods than are common in ring and module theory. The biggest casualty of the reduced algebraic structure is eigenvalue theory, but most properties and techniques needed for system analysis remain: coprime factorization, inner-outer factorization (the topic of next chapter), reachability, controllability and then more elaborate topics such as embedding, interpolation and model reduction. The reason why most properties needed still work is to be found in their ‘geometric foundation’, that can be properly explored by just using simple matrix calculus. This is the approach we have been taking and, we hope, already pretty convincingly at this point.

All this works fine, but a new question appears: whether some of the key results of the polynomial theory can be reproduced, without resort to divisibility methods (the Euclidean algorithm) that lie at the basis of the classical canonical form theory. It turns out that the notion 'dead beat control' as proposed by P. Van Dooren brings exactly what is needed [12]. We shall see, in the chapter on polynomial models, that it easily generalizes to LTV systems. So we shall dispose of two quite different types of external factorization: the 'inner' type, that uses inner denominators, and the polynomial type, that uses matrix polynomials in the shift. The two types lead to similar results under certain regularity conditions, but their domains of applicability can be very different, as will appear in the following chapters.

Chapter 10

Inner-outer factorization

Inner-outer factorization is probably the most fundamental and most important operation in system theory. It certainly solves major problems in almost all areas of interest: system inversion, estimation theory, control. On the face of it, it seems not much more than the application of something like QR-factorization or SVD on a system description, aiming at determining important system subspaces, which then play a central role in solving the issues just mentioned. Its power is due to the fact that it leads to a linear recursion and stable numerical operations. In this chapter we simply concentrate on the algorithm itself, leaving its use in major problems and applications to later chapters. We start with working out a sample case, which will turn out to be of direct use in the applications treated in the subsequent chapters, then move on to the underlying theory and end up with 'geometric' considerations that will improve our general insights.

Menu

Hors d'oeuvre

QR-factorization

First course

A prototype Outer-Inner factorization

Second course

Semi-separable Outer-Inner factorization

Dessert

Infinitely indexed systems

10.1 Introduction: echelon forms

Let us start out with a simple algebraic exercise: rotating a unitary (column) vector $u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$ of dimension $1 + n$, with $\|u\|^2 = |u_1|^2 + \sum_{i=1}^{n} |u_{2,i}|^2 = 1$ and with non-negative real first element u_1 to the positive first axis, which we call e_1 generically (ignoring its dimension)—e_1 is then a vector of dimension $n+1$. The following orthogonal (or unitary in the complex case) matrix pulls the trick:

$$Q_u := \begin{bmatrix} u_1 & -u_2' \\ u_2 & I - u_2 \frac{1}{1+u_1} u_2' \end{bmatrix} \tag{10.1}$$

and we shall have $u = Q_u e_1$ and $Q_u' u = e_1$, which is easily verified directly. The matrix Q_u has $\det Q_u = 1$—it is a generalized rotation matrix. One can actually show that it can be produced by a sequence of elementary rotations (often called 'Givens rotations'), but a direct application of such a matrix to an arbitrary vector—say $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ with x_1 scalar—produces the following 'efficient' computation:

$$Q_u x = \begin{bmatrix} u_1 x_1 - u_2' x_2 \\ x_2 + u_2 (x_1 - \frac{u_2' x_2}{1+u_1}) \end{bmatrix} \tag{10.2}$$

in which the inner product $u_2' x_2$ should be executed only once.

A more general non-zero vector $a = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix}$ of dimension $1 + n$ can of course also be rotated to the direction of the first unit vector e_1. It turns out to be useful to do that a bit carefully in the general complex case. Let $\|a\|$ be the euclidean norm of the vector and $a_1 = |a_1| e^{j\phi}$ (with $j = \sqrt{-1}$), then $u = a \frac{e^{-j\phi}}{\|a\|}$ will be like u before, and the effect of Q_u on a will be $Q_u a = e_1 \|a\| e^{j\phi}$ (it is useful to retain the norm of a and the phase of the first element for further use.). Let, for a general non-zero vector a, Q_a be defined consistently as $Q_a := Q_{a \frac{e^{-j\phi}}{\|a\|}}$.

Suppose next that one disposes of a collection of vectors $A = \begin{bmatrix} a_1 & \cdots & a_m \end{bmatrix}$ of dimension $1+n$ (say columns of a matrix), and suppose that we are entitled to apply rotations to them (i.e., to the left). Suppose a_k is the non-zero vector with the smallest k and that its first element is $a_{k,1} = |a_{k,1}|e^{j\phi_1}$. Applying Q_{a_k} to the stack now produces the following typical form:

$$\begin{aligned} & Q'_{a_k}\left[\begin{array}{cccccc} a_1 & \cdots & a_{k-1} & a_k & a_{k+1} & \cdots & a_m \end{array}\right] \\ & \quad = \left[\begin{array}{cccccc} 0 & \cdots & 0 & e_1\|a_k\|e^{j\phi_1} & Q'_{a_k}a_{k+1} & \cdots & Q'_{a_k}a_m \end{array}\right] \\ & \quad = \left[\begin{array}{ccccccc} 0 & \cdots & 0 & \|a_k\|e^{j\phi_1} & * & \cdots & * \\ 0 & \cdots & 0 & 0 & b_{k+1} & \cdots & b_m \end{array}\right] \end{aligned} \tag{10.3}$$

where the "$*$" indicate entries that have been modified (and will remain unchanged later one), and the $\left[\begin{array}{ccc} b_{k+1} & \cdots & b_m \end{array}\right]$ is a new collection of vectors, now of dimension n, and on which the procedure can be repeated without producing new fill-ins in the zero elements obtained so far, now with one dimension less (some of the zeros shown above may disappear, e.g., when a_1 is already non-zero.). Continuing this way, now on the b's and realizing that products of rotation matrices remain orthogonal or unitary, after a number of steps one obtains a so called *echelon* form:

$$A = \left[\begin{array}{cc} Q_1 & Q_2 \end{array}\right]\left[\begin{array}{c} R_1 \\ 0 \end{array}\right] \tag{10.4}$$

in which

$$R_1 = \left[\begin{array}{cccccccccccc} 0 & \cdots & 0 & R_{1,k_1} & \cdots & * & * & \cdots & * & * & \cdots \\ 0 & \cdots & 0 & 0 & \cdots & 0 & R_{2,k_2} & \cdots & * & * & \cdots \\ & & & & & \vdots & & & & & \\ 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & R_{\delta,k_\delta} & \cdots \end{array}\right], \tag{10.5}$$

δ is the rank of A, Q is orthogonal or unitary. One sees easily that the columns of Q_1 form a basis for the range of A, while the columns of R_1' form a basis for the co-range (i.e., the range of A'), and the columns of Q_2 for the co-kernel.

A similar, even more powerful result could have been obtained by SVD (Singular Value Decomposition) of A:

$$A = \left[\begin{array}{cc} U_1 & U_2 \end{array}\right]\left[\begin{array}{cc} \Sigma & 0 \\ 0 & 0 \end{array}\right]\left[\begin{array}{c} V_1' \\ V_2' \end{array}\right] = \left[\begin{array}{cc} U_1 & U_2 \end{array}\right]\left[\begin{array}{c} \Sigma V_1' \\ 0 \end{array}\right] \tag{10.6}$$

at the cost of more computations. Here $\Sigma = \left[\begin{array}{ccc} \sigma_1 & & \\ & \ddots & \\ & & \sigma_\delta \end{array}\right]$ are the singular values of A in order: $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_\delta > 0$, the columns of U_1 form a basis for the range of A while the columns of V_1 and $V_1\Sigma$ for the co-range.

The example in the next section (and many in subsequent chapters) actually uses a variant of the QR-algorithm just presented, namely an algorithm that starts at the bottom right corner and produces an RQ factorization, with R again an echelon matrix and Q an orthogonal or unitary matrix. The procedure now starts out with a collection of rows rather than columns, it is dual to the preceding:

$$A = \begin{bmatrix} 0 & R_2 \end{bmatrix} \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} \tag{10.7}$$

R_2 is obtained by compressing towards the last column starting with the bottom row (skipping it when zero), it will look like

$$\begin{bmatrix} \vdots & & \vdots & \vdots \\ R_{\delta,k_\delta} & & * & * \\ 0 & & * & * \\ \vdots & & \vdots & \vdots \\ 0 & & R_{2,k_2} & * \\ 0 & & 0 & * \\ \vdots & \cdots & \vdots & \vdots \\ 0 & & 0 & R_{1,k_1} \\ 0 & & 0 & 0 \\ \vdots & & \vdots & \vdots \\ 0 & & 0 & 0 \end{bmatrix} \tag{10.8}$$

these columns now forming a base for the range of A. Also in this case, a more accurate result can be obtained through SVD, when needed.

10.2 A prototype example of Outer-Inner factorization

Let us work simply on a 4×4 block-lower triangular (i.e., causal) matrix whose realization is given. Hence, let

$$T = \begin{bmatrix} \boxed{D_0} & & & \\ C_1B_0 & D_1 & & \\ C_2A_1B_0 & C_2B_1 & D_2 & \\ C_3A_2A_1B_0 & C_3A_2B_1 & C_3B_2 & D_3 \end{bmatrix} \tag{10.9}$$

be our prototype system, with the use of the realization given explicitly. Our goal will be to find a factorization $T = \begin{bmatrix} T_o & 0 \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = T_0V_1$ such that V is causal unitary, T_o is left invertible and V_1 co-isometric, with right inverse $V_1{}'$, **and**, for doing so, to work solely on the realizations. We shall discover that realizations for T_o and V are just as simple as the realization for T, and can easily be derived from it via a linear forward recursion.

The procedure works column by column starting with the first (actually it works on the main block diagonals downwards, but that will soon be apparent.). All blocks of the first column except the first share B_0, let us see what an RQ-factorization of $\begin{bmatrix} B_0 \\ D_0 \end{bmatrix}$ achieves. Let

$$\begin{bmatrix} B_0 \\ D_0 \end{bmatrix} := \begin{bmatrix} 0 & Y_1 & B_{o0} \\ 0 & 0 & D_{o0} \end{bmatrix} \begin{bmatrix} Q_{1,1} & Q_{1,2} \\ Q_{2,1} & Q_{2,2} \\ Q_{3,1} & Q_{3,2} \end{bmatrix} \tag{10.10}$$

in which Y_1 and D_{o1} are left invertible, Q is unitary and all quantities of the right hand side are computed from the given left hand side quantities by the RQ algorithm (this is commonly called 'array processing'. In the next section we shall see what the blocks in Q actually mean.).

Applying Q' to the first block column (i.e., to the right) of T now produces (you can just as well exchange the order of B_0 and D_0)

$$\begin{bmatrix} 0 & 0 & D_{o0} \\ 0 & C_1Y_1 & C_1B_{o0} \\ 0 & C_2A_1Y_1 & C_2A_1B_{o0} \\ 0 & C_3A_2A_1Y_1 & C_3A_2A_1B_{o0} \end{bmatrix} \tag{10.11}$$

The columns of the last (third) sub-column form a basis, because D_{o0} is left invertible. Let us permute the second and third columns (the latter will become part of the final left factor) leaving the zero columns to the left. The remainder, combined with the rest of the matrix that has been left untouched is now

$$\left[\begin{array}{cc|cc} C_1Y_1 & D_1 & 0 & 0 \\ \hline C_2A_1Y_1 & C_2B_1 & D_2 & \\ C_3A_2A_1Y_1 & C_3A_2B_1 & C_3B_2 & D_3 \end{array}\right] \tag{10.12}$$

(actually, $\mathrm{diag}[Q_1', I, \cdots, I]$ is the orthogonal transformation applied to the full matrix.) One leaves the first block row intact and moves to the second block row, noticing that the dimensions of the diagonal element (and hence of the second block column) have changed. The next step is now to reduce the new first diagonal element $\begin{bmatrix} C_1Y_1 & D_1 \end{bmatrix}$, taking into account that the effect has to be propagated down the combined column (this is the "generic" step!). Compute therefore a new RQ factorization, with a new Q structured as before

$$\begin{bmatrix} A_1Y_1 & B_1 \\ C_1Y_1 & D_1 \end{bmatrix} = \begin{bmatrix} 0 & Y_2 & B_{o1} \\ 0 & 0 & D_{o1} \end{bmatrix} Q \tag{10.13}$$

Application of this Q' to the right of the new first column (it has to be verified that this will not change any of the other elements that have already been set aside: there are no "fill ins") produces the new second sub-column

$$\begin{bmatrix} 0 & 0 & D_{o1} \\ 0 & C_2Y_2 & C_2B_{o1} \\ 0 & C_3A_2Y_1 & C_3A_2B_{o1} \end{bmatrix} \tag{10.14}$$

and, again, the columns of the third sub-column form a basis. Again, we permute relevant columns and move the zero column to the far left. Let's check the overall result, after application of the two subsequent Q':

$$\left[\begin{array}{cc|cc|ccc} 0 & 0 & D_{o0} & 0 & 0 & 0 & 0 \\ 0 & 0 & C_1B_{o0} & D_{o1} & 0 & 0 & 0 \\ 0 & 0 & C_2A_1B_{o0} & C_2B_{o1} & C_2Y_2 & D_2 & 0 \\ 0 & 0 & C_3A_2A_1B_{o0} & C_3A_2B_{o1} & C_3A_2Y_2 & C_3B_2 & D_3 \end{array}\right] \tag{10.15}$$

The next operation takes place on block sub-columns 3 and 4, with again a new Q, similarly as the previous:

$$\begin{bmatrix} A_2Y_2 & B_2 \\ C_2Y_2 & D_2 \end{bmatrix} = \begin{bmatrix} 0 & Y_3 & B_{o2} \\ 0 & 0 & D_{o2} \end{bmatrix} \tag{10.16}$$

and produces

$$\left[\begin{array}{ccc|ccc|cc} 0 & 0 & 0 & D_{o0} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & C_1B_{o0} & D_{o1} & 0 & 0 & 0 \\ 0 & 0 & 0 & C_2A_1B_{o0} & C_2B_{o1} & D_{o2} & 0 & 0 \\ 0 & 0 & 0 & C_3A_2A_1B_{o0} & C_3A_2B_{o1} & C_3B_{o2} & C_3Y_2 & D_3 \end{array}\right] \tag{10.17}$$

The final step is simpler, with a last Q for which $\left[\begin{array}{cc} C_3Y_2 & D_3 \end{array}\right] = \left[\begin{array}{cc} 0 & D_{o3} \end{array}\right]$ with D_{o3} left invertible, giving finally

$$\left[\begin{array}{cccc|cccc} 0 & 0 & 0 & 0 & D_{o0} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & C_1B_{o0} & D_{o1} & 0 & 0 \\ 0 & 0 & 0 & 0 & C_2A_1B_{o0} & C_2B_{o1} & D_{o2} & 0 \\ 0 & 0 & 0 & 0 & C_3A_2A_1B_{o0} & C_3A_2B_{o1} & C_3B_{o2} & D_{o3} \end{array}\right] \tag{10.18}$$

The operations so far have produced a global RQ-factorization $T = \left[\begin{array}{cc} 0 & T_o \end{array}\right] Q$ in which T_o is left invertible and Q is orthogonal or unitary. In addition, T_o has partly inherited the state space structure of T, we have obtained $T_o = D_o + C(I - ZA)^{-1}ZB_o$. As already explained in chapter 8, a left inverse for T_o will have the same state structure as T_o itself, and because all the D_{ok} are by themselves left invertible, a realization for a left inverse is simply $T_o^+ = D_o^+ - CD_o^+(I - Z\Delta)^{-1}ZD_o^+B_o$, in which $\Delta = A - B_oD_o^+C$. Notice that the realizations of T_o and T_o^+ are not necessarily minimal, it may even be that T_o is purely block diagonal.

What about the overall Q? Backtracking, we see that each Q just operates on the first block-column of the subsequent matrices, which corresponds to subsequent block columns of the original. However, part of the result is propagated further on, but in a limited way: just to the next block column. This is a strong indication that also the Q matrix has a limited state space structure. It is very well possible to track the structure of Q down in detail from the previous, but there is an attractive short cut, using our diagonal algebra, presented in the next section.

10.3 The general matrix case

Let us try to factor the original causal $T = D + C(I - ZA)^{-1}ZB$ into $T = \left[\begin{array}{cc} 0 & T_o \end{array}\right] V$ in which T_o is left invertible and V is causal unitary, with unitary realization $V = D_V + C_V(I - ZA_V)^{-1}ZB_V$ (we have put the zero's out front for further consistency.). Let also $T_o = D_o + C_o(I - ZA_o)^{-1}ZB_o$ a proposed realization for T_o, and let us see how we can determine all these realization components so that the factorization equation is satisfied. As V is going to be unitary, we can invert it and write $TV' = \begin{bmatrix} 0 & T_o \end{bmatrix}$, or, using realizations

$$\begin{aligned}&[D + C(I - ZA)^{-1}ZB][D_V' + B_V'Z'(I - A_V'Z')^{-1}C_V'] \\ &\quad = \begin{bmatrix} 0 & D_o \end{bmatrix} + \begin{bmatrix} 0 & C_o \end{bmatrix}(I - ZA_o)^{-1}ZB_o\end{aligned} \tag{10.19}$$

Let us first look at the "quadratic term" $(I - ZA)^{-1}ZBB_V'Z'(I - A_V'Z')^{-1}$. It is easy to see (and we did this 'decomposition in parts' already before) that it is equal to $(I - ZA)^{-1}ZAY + Y + YA_V'Z'(I - A_V'Z')^{-1}$ with

$$Y^{<-1>} = BB_V' + AYA_V', \tag{10.20}$$

by pre- and post multiplication with respect. $(I - ZA)$ and $(I - A_V'Z')$. Hence, the first member becomes

$$\begin{aligned}&(DD_V' + CYC_V') + C(I - ZA)^{-1}Z(BD_V' + AYC_V') \\ &\quad +(DB_V' + CYA_V')Z'(I - A_V'Z')^{-1}C_V'\end{aligned} \tag{10.21}$$

and this should now be equal to the second member of eq. (10.19). First, we can get the anti-causal part of the product zero by requiring $DB_V' + CYA_V' = 0$. Next, we can choose $A_o = A$ and $C_o = C$. Finally, further unknown quantities such as D_o and C_o can be identified from $\begin{bmatrix} 0 & D_o \end{bmatrix} = DD_V' + CYC_V'$ and $\begin{bmatrix} 0 & B_o \end{bmatrix} = DB_V' + CYA_V'$. Putting all the equations together we find

$$\begin{bmatrix} AY & B \\ CY & D \end{bmatrix} \left[\begin{array}{c|c} A_V' & C_V' \\ B_V' & D_V' \end{array}\right] = \left[\begin{array}{c|cc} Y^{<-1>} & 0 & B_o \\ 0 & 0 & D_o \end{array}\right]. \tag{10.22}$$

This already resembles very much our previous example, except for some ordering of the columns (as one would expect since we juggled around with the order in the example.). To see this, one should realize that if one specializes the equation to a specific index k, one actually gets

$$\begin{bmatrix} A_kY_k & B_k \\ C_kY_k & D_k \end{bmatrix} \left[\begin{array}{c|c} A_{Vk}' & C_{Vk}' \\ B_{Vk}' & D_{Vk}' \end{array}\right] = \left[\begin{array}{c|cc} Y_{k+1} & 0 & B_{ok} \\ 0 & 0 & D_{ok} \end{array}\right], \tag{10.23}$$

which shows that once Y_k (and of course the realization for T at index k) are known, an RQ factorization will produce the local realizations for V and T_o, as well as the new value Y_{k+1}. One extra observation (which we shall soon confirm), is that in the example we can choose $Y^{<-1>}$ such that it is also left invertible. Although such a choice is strictly speaking not necessary, it is useful because it clearly puts the co-kernel in evidence, as we shall see. So, let us require $Y^{<-1>}$ to be left invertible (i.e., has smallest column dimension), and split C'_V and D'_V accordingly, actually rewriting them as $\begin{bmatrix} C'_W & C'_V \\ D'_W & D'_V \end{bmatrix}$ for some new, to be defined W. This produces:

$$\begin{bmatrix} AY & B \\ CY & D \end{bmatrix} \left[\begin{array}{c|cc} A'_V & C'_W & C'_V \\ B'_V & D'_W & D'_V \end{array}\right] = \left[\begin{array}{c|cc} Y^{<-1>} & 0 & B_o \\ 0 & 0 & D_o \end{array}\right], \tag{10.24}$$

and then moving the zero block column in front, we finally get

$$\begin{bmatrix} AY & B \\ CY & D \end{bmatrix} \left[\begin{array}{c|cc} C'_W & A'_V & C'_V \\ D'_W & B'_V & D'_V \end{array}\right] = \left[\begin{array}{c|cc} 0 & Y^{<-1>} & B_o \\ 0 & 0 & D_o \end{array}\right], \tag{10.25}$$

in which both $Y^{<-1>}$ (hence also Y) and D_o are left invertible. With $T_o = D_o + C(I - AZ)^{-1}ZB_o$, $V = D_V + C_V(I - ZA_V)^{-1}ZB_V$ and $W = D_W + C_W(I - ZA_V)^{-1}ZB_V$, the final result is $T = \begin{bmatrix} 0 & T_o \end{bmatrix} \begin{bmatrix} W \\ V \end{bmatrix} = T_oV$, with T_o left causally invertible, V co-isometric and the co-kernel of T consisting of all output vectors in the range of W'.

10.4 Infinitely indexed systems

What happens with the inner-outer or outer-inner factorization of a causal operator $T = D + C(I - ZA)^{-1}ZB$ when the indices run from $-\infty$ to $+\infty$? First of all, as the operators are now all infinite dimensional, we need to put some boundedness assumptions on T and the diagonal operators A, B, C and D. Let us just assume that T is indeed bounded as a map from an input $\ell_2^{\mathbf{m}}$ space to an output $\ell_2^{\mathbf{n}}$ space (see chapter **??** for further explanations on the notation: 'ℓ_2' just means 'quadratically summable' and '$\mathbf{m}$' (respect. '$\mathbf{n}$') is the sequence of input (respect. output) indices), together with the assumption that T is causal and semi-separable. Next, we can, e.g., choose a realization for T in input normal form, by choosing an orthonormal basis for the co-range of each (by definition finite dimensional) H_k. This leads to an A and a B that are contractive (and hence bounded), but it is very well conceivable that A so obtained is not u.e.s.. Let us therefore put the *extra requirement* that the resulting A is indeed u.e.s. so that the expression for the realization makes sense (the inverse $(I - ZA)^{-1}$ exists as a bounded operator.). If then also the C operator (and of course also D—but this would be automatic) is bounded, we would have a realization in which all four realization operators are bounded and A is, moreover, u.e.s.. Conversely, suppose a realization exists with bounded A, B, C and D and, in addition, A u.e.s., then each state transformation with *bounded* operator $\hat{x} = Tx$ would also yield such a realization, in particular: both the input and output normal forms would be such as well. Therefore the following definition:

Definition 7 *A causal system T is called regular, if it is semi-separable and has a realization $T = D + C(I - ZA)^{-1}ZB$ with bounded A, B, C, D and A u.e.s.*

Once we deal with regular systems, the outer-inner factorization of the previous section just goes through, and has the added benefit that the inner and the outer factor obtained will be regular as well. However, this is not true of the left-inverse of the outer factor as we show by example in chapter **??**. The discussion of this delicate question would lead too far here, let us suffice with a somewhat imprecise definition:

Definition 8 *We say that a causal operator T is left-outer, if there exists a causal operator T^+ (potentially unbounded) such that $T^+T = I$. T is said to be right-outer, if there exists a 'causal' T^+ (potentially unbounded) such that $TT^+ = I$. T is said to be outer if it is both left- and right-outer, or, equivalently, iff it has a 'causal' inverse T^{-1} (potentially unbounded).*

The notion of 'causality' will have to be extended to a limited class of potentially unbounded operators, and this is done in the chapter mentioned, where we give precise definitions. The issue of boundedness of the inverse of the outer factor plays already for some simple LTI systems, take e.g. $T(z) = z(z-1)$, then z is the inner factor and $z-1$ the outer factor; $(z-1)^{-1}$ is unbounded, but may be considered causal as the limit of $(z - (1 + \epsilon))^{-1}$ when $\epsilon > 0$ goes to zero. In continuous-time LTI systems, this situation occurs when there are zeros on the imaginary axis in the original operator, a case that is very common in electrical circuit theory. Such systems are only invertible in a weak sense.

Summarizing: in the previous section we have established the fact that an arbitrary causal and regular operator T always admits a factorization

$T = T_{ol}V_r$, in which T_{ol} is left-outer and V_r is right-inner. Such a factorization is called an outer-inner factorization. Dually, T admits an inner-outer factorization $T = V_\ell T_{or}$ with V_ℓ left-inner and T_{or} right-outer. When T is already left-outer, then the outer-inner factorization is trivial, with $V_r = I$.

10.5 Items for discussion

- As in the chapter of external factorizations, it is interesting to work some simple examples, and study inner-outer as well as outer-inner factorization of a few 'simple' cases, particularly the matrices

$$\begin{bmatrix} \boxed{1} & & & \\ 1/2 & 1 & & \\ & 1/2 & 1 & \\ & & 1/2 & 1 \end{bmatrix}, \begin{bmatrix} \boxed{1} & & & \\ 1/2 & 1 & & \\ & 1/2 & 1 & \\ & & \ddots & \ddots \end{bmatrix}, \begin{bmatrix} \boxed{1} & & & \\ 2 & 1 & & \\ & 2 & 1 & \\ & & \ddots & \ddots \end{bmatrix}. \tag{10.26}$$

  Several interesting phenomena appear, that will motivate quite a few further developments! Compare also what happens in relation to the external factorizations discussed before.

- With respect to the introductory section: numerical analysts have developed a method called 'Householder transformation' to bring a given vector in the direction of the first axis. The method presented here is based on a (generalized) rotation. It has several advantages over the Householder transformation, which uses a reflection instead of a rotation. It is interesting to compare the two approaches.

- Riccati equation: when one squares the square-root equation 10.22 (i.e., multiply it to the right with its conjugate) one obtains, with $M := YY'$

$$\begin{bmatrix} AMA' + BB' & AMC' + BD' \\ CMA' + DB' & CMC' + DD' \end{bmatrix} = \begin{bmatrix} M^{<-1>} + B_oB_o' & B_oD_o' \\ D_oB_o' & D_oD_o' \end{bmatrix} \tag{10.27}$$

  Since D_o has to be minimal, it will have a left inverse $D_o^\dagger$, and we will have $B_o = (AMC' + BD')(D_o^\dagger)'$ as well as $(D_oD_o')^\dagger = (CMC' + DD')^\dagger$. It follows that M will be a (semi-)positive definite solution of the 'Riccati' recursion

$$M^{<-1>} = AMA' + BB' - (AMC' + BD')(CMC' + DD')^{\dagger}(CMA' + DB') \tag{10.28}$$

In the matrix case, the recursion would start with 'empty', and it should be recognized that it does not have to produce a strict positive definite solution, since the dimension of M depends on the dimension of Y, which may disappear. One may argue that it is not wise to solve this recursion directly, as it produces the needed Y only indirectly and in quadratic form, thereby losing numerical accuracy (do you know why?) and requires the computation of a pseudo-inverse as well.

10.6 Notes

- Although inner-outer or outer-inner factorization are maybe the most central operations in dynamical system theory, they have not been recognized as such in many treatments, because their far reaching effects have often not been seen clearly, especially in the engineering community. However, already in the early times of Hardy space theory its importance was recognized by the mathematicians working on complex function analysis, leading first to the Beurling theorem and then later, when matrix functions were considered, to the extension of the Beurling theorem known as the Beurling-Lax theorem. In a sweeping generalization of the basic ideas contained in Hardy space theory, Arveson [4] set up the a new algebraic category called "Nest Algebras", for which the basic concepts behind inner-outer factorization, namely the properties of a special type of nested invariant subspaces, hold. In more recent times, it has been realized that these concepts even extend usefully to non-linear systems, especially the work of Willems [**?**], Ball and Helton [5] and van der Schacht [38] and their students have shown the way into that still not fully explored and very promising direction.

- A different approach (leading to the same effects) has come from estimation theory, and in particular the work of Kailath and his early students. When studying the Kalman filter and its somewhat pedestrian way of computing state estimations, they realized that a more direct way would be based on the propagation of the square root of a covariance rather than the covariances themselves. This then lead to the famous "square root algorithm" for the Kalman filter [21], which, as we shall see in the chapter on Kalman filtering, is nothing but a direct implementation of inner-outer factorization on the assumed model.

- In the following chapters we shall encounter many applications of inner-outer factorization theory: to estimation theory (the Kalman filter and the LU-factorization), to system inversion theory and to control. In all these cases, what the factorization mainly achieves is what one could call a *dichotomy* on the inverse of the system, if it exists, and otherwise a substitute for the inverse. Dichotomy produces a segregation of the causal and the anticausal components: the causal part goes into the outer factor and the anticausal part in the inner factor. It is remarkable that this can be done with a linear recursion, or, to put it differently: no need to compute eigenvalues (in the LTI case). As a side effect, the theory generalizes to timevariant and even nonlinear. This does not come without some cost: there may be ambiguity when the outer factor turns out not to be invertible (in the LTI case: when there are zeros on the boundary). Nonetheless, the theory remains valid even in that case, but produces a result that has to be carefully interpreted and does not necessarily give complete answers, which then are very hard to get.

Chapter 11

The Kalman filter

Menu

Hors d'oeuvre

Linear state estimation

Main course

The Kalman filter as outer-inner factorization

Dessert

Discussion issues

For a long time it has been common knowledge that the celebrated Kalman filter [25] can be considered, for discrete-time systems, a case of Cholesky factorization on a special matrix (i.e., positive definite LU, LL' or Cholesky factorization) numerically executed on the state-space description, and that it can be obtained via a square root algorithm. This insight goes back to the pioneering work of Morf and Kailath [28], who have derived many additional properties relating estimation theory to efficient and numerically attractive algorithms. Following this path, the connection with outer-inner factorization then becomes almost obvious.

We shall follow the reverse path: from outer-inner factorization of the specific Kalman case, to the interpretation as estimation and innovation filter. However, this admittedly non-historical approach is likely the most direct. It provides an easy proof for the Kalman filter formulas and leads easily to nonlinear generalizations as well. For extensive reference to the classical Kalman filter literature, see [22]. We change our notation on outer-inner factorization a bit to conform to what is commonly used in the literature.

Also traditionally, the filter situation considered is time variant (the Kalman filter was probably the first great success of time variant system theory!), assumed to start at index 0 and recursively being updated from $k \geq 0$ to $k+1$ as long as necessary, without putting a limit on how far it goes, but always keeping the situation finite. In a separate chapter, we consider the corresponding LTI situation and the connection with spectral factorization and Wiener filtering.

11.1 Linear state estimation basics

The classical Kalman filter situation starts out with a given stochastic linear and time-varying system model. We restrict ourselves here to the discrete time case, described, by a minimal, linear, discrete time realization:

$$\left\{ \begin{array}{rcl} x_{k+1} & = & A_k x_k + B_k u_k \\ y_k & = & C_k x_k + \nu_k \end{array} \right. \tag{11.1}$$

As before, x_k is the state of the system at index point k, but the inputs are now assumed to be stochastic vectors (i.e., unknown except for their statistical properties) and described respectively by u_k, the vector of input noises and ν_k, the vector of measurement or output noises. In the original formulation, these noise processes are assumed of zero mean and *uncorrelated*, with given covariances. Moreover, one assumes that the process starts at $k=0$ with as initial input not only u_0, but also x_0, the initial state, which is also assumed to be stochastic, zero mean and having a given covariance P_0 uncorrelated with all other inputs. The goal of the Kalman filter for such a given process is to find an estimate $\hat{x}_{k+1}$ recursively for each state x_{k+1} based on the measurement of the outputs from $k=0$ to the actual k, which minimizes the statistical quadratic error.

Let us make these assumptions more precise. We write the mean of a stochastic vector or matrix with the expectation operator $\mathbf{E}$ and assume $\mathbf{E}(x_0) = 0$, $\mathbf{E}(u_k) = 0$, $\mathbf{E}(\nu_k) = 0$ and all further means resulting from linear operations on those will of course also be zero: $\mathbf{E}(y_k) = 0$ and $\mathbf{E}(x_k) = 0$ for all k as they are all linearly dependent on the original stochastic variables. Covariances of the processes u_k and ν_k are supposed known (in chapter **??** we shall deal with a situation where this is not the case), we put as given, with $\delta_{k,l} := 1$ when $k = \ell$ and otherwise zero: $\mathbf{E}(u_k u'_\ell) := Q_k \delta_{k,\ell}$, $\mathbf{E}(\nu_k \nu'_\ell) = R_k \delta_{k,\ell}$ and $\mathbf{E}(u_k \nu'_\ell) = 0$ for all k and ℓ. Also x_0 is assumed uncorrelated with all the other noise sources and $P_0 := \mathbf{E}(x_0 x_0')$ is assumed known (in a practical situation this data would follow from a statistical analysis.).

$\hat{x}_k$ now has to be determined so that $P_k := \mathbf{E}[(x_k - \hat{x}_k)(x_k - \hat{x}_k)']$ is minimized at each index k, *given the measured outputs* $y_{1:k-1} := \mathrm{col}[y_0, \cdots, y_{k-1}]$. The vector $e_{x,k} := x_k - \hat{x}_k$ is classically defined as the *innovations* at index k, we define similarly the *output innovation* as $e_{y,k} := y_k - \hat{y}_k$, where $\hat{y}_k$ is the least squares estimate of y_k given $y_{0:k-1}$. We shall also use normalized innovations and indicate them with a bar, as e.g., in $\bar{e}_k = P_k^{-1/2} e_k$. The Kalman filter will be a recursive process that reads in the y_k's in sequence, and determines from them the least squares estimates $\hat{x}_{k+1}$ at stage k as efficiently as possible.

All variables in the model being stochastic and zero mean, a special Euclidean-like inner-product algebra can be defined for them. The inner product between two zero mean stochastic variables, say u and v, is defined as $(u, v) = \mathbf{E}(uv')$. Hence the quadratic norm of a variable $\|u\|^2 = \mathbf{E}(uu')$ is its covariance, and two variables are *orthogonal* when uncorrelated. Extending this to (column) vectors, there is an issue on how to define "orthogonality". One way is to say that two stochastic vectors $u = \mathrm{col}\begin{bmatrix} u_1 & \cdots & u_n \end{bmatrix}$ and $v = \mathrm{col}\begin{bmatrix} v_1 & \cdots & v_n \end{bmatrix}$ are orthogonal if $\mathbf{E}(\mathrm{trace}(uv') = 0)$. With this definition, only components with the same k, namely u_k and v_k (all k) are orthogonal on each other. We want to require more, namely that all components of u are orthogonal on *all* components of v, or, in formula $\mathbf{E}(uv') = [\mathbf{E}u_k v_\ell']_{:,:} = 0$ as a matrix. This is necessary for estimation problems in which all the components are individually available, the case we have here. (One should realize that, in practice, each stochastic variable u is actually represented as a 'data vector', and in the present formalism, a data vector is a *row* vector. A stochastic vector is then represented as a 'data matrix', it is a column of rows, each row being a single data vector.) [1]

[1]Working with such 'matrix' inner products (one could write $(A, B) := AB'$—a 'matrix inner product' between the two matrices A and B assumed to have the same number of columns) is not the same as working with scalar inner products defined on vectors because there is no corresponding scalar inner product. This may seem strange, but if one keeps in mind that there is an underlying inner product space on individual components, then things naturally fall into place: suppose that A is an $m \times N$ matrix, and B an $n \times N$ matrix, then the 'orthonormality' condition $AB' = 0$ boils down to mn scalar orthogonality conditions.

11.2 The normalized model for the Kalman filter

Let us first write the given equations in a normalized form, with $\bar{u}_k$ and $\bar{\nu}_k$ normalized (i.e., uncorrelated zero mean processes with unit covariance), and make the given covariances explicit in square-root form:

$$\left\{ \begin{array}{rcl} x_{k+1} & = & A_k x_k + \left[\begin{array}{cc} B_k Q_k^{1/2} & 0 \end{array} \right] \left[\begin{array}{c} \bar{u}_k \\ \bar{\nu}_k \end{array} \right] \\ y_k & = & C_k x_k + \left[\begin{array}{cc} 0 & R_k^{1/2} \end{array} \right] \left[\begin{array}{c} \bar{u}_k \\ \bar{\nu}_k \end{array} \right] \end{array} \right. \tag{11.2}$$

The estimation principle used is pretty simple. Assuming all inputed processes x_0, u_k and ν_k zero mean, the estimate $\hat{x}_{k+1}$ with the smallest least square error, measured as a covariance, is such that the estimation error $e_{x,k+1}$ is 'orthogonal' (component wise) on the known data, in this case $y_{0:k}$, (so-called "Wiener principle"). 'Orthogonal' in this context is by definition *uncorrelated* : two zero mean stochastic vectors w_1 and w_2 of the same dimension are said to be orthogonal when $\mathbf{E}(w_1 w_2') = 0$ (in case w_1 and/or w_2 are vectors, this is a zero outer product, and all entries are zero, meaning that the individual entries of w_1 are orthogonal to the individual entries of w_2).

However, a more general approach, equivalent in the linear case, is to just determine the *a posteriori estimate* $x_{k+1}|_{y_{0:k}}$, which exists also in nonlinear cases and for arbitrary distributions (so far we did not make any assumptions on the distributions), and has the same orthogonality property. This entitles us to write $\hat{x}_{k+1} := x_k|_{y_{0:k}}$, which will always be such that $\mathrm{E}[(x_{k+1} - \hat{x}_{k+1})y_{0:k}'] = 0$ (notice that this is not, in general, a covariance. It is just a matrix of stochastic inner products—it will be a covariance only when the processes are zero-mean). We shall see that the outer-inner factorization will produce the necessary component wise orthogonality in all cases.

11.3 Outer-inner factorization

The detailed outer-inner recursion for the Kalman filter then runs as follows:

Step 0

$$\begin{bmatrix} - & - & - \\ A_0 P_0^{1/2} & B_0 Q_0^{1/2} & 0 \\ C_0 P_0^{1/2} & 0 & R_0^{1/2} \end{bmatrix} = \begin{bmatrix} - & - & - \\ 0 & M_1 & B_{o,0} \\ 0 & 0 & D_{o,0} \end{bmatrix} V_0 \tag{11.3}$$

(R-Q factorization) in which V_0 is unitary. For consistency purposes, R_0,

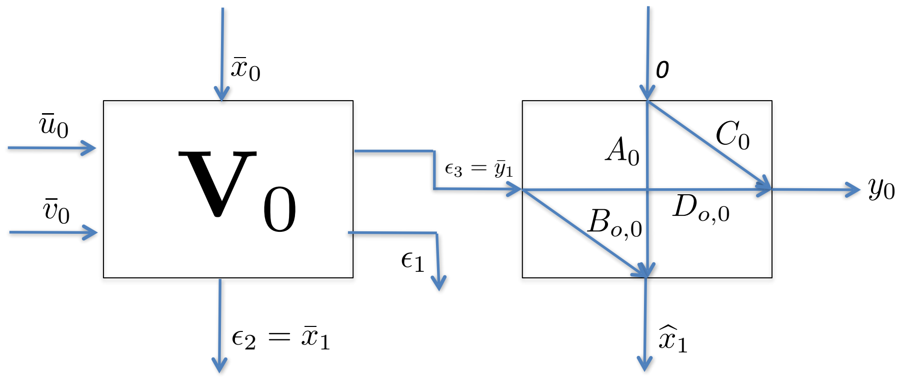


Figure 11.1: The 0'th step in the outer-inner factorization of the Kalman filter model.

P_0, Q_0 must be (square) non-singular, and $\begin{bmatrix} A_0 & B_0 \end{bmatrix}$ must be non-singular, which is achieved by requiring the realization to be minimal (see the simple proof in the notes at the end of the chapter). The consequence is that both $D_{o,0}$ and M_1 are square, non-singular. Let now

$$\begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \end{bmatrix} := V_0 \begin{bmatrix} \bar{x}_0 \\ \bar{u}_0 \\ \bar{\nu}_0 \end{bmatrix} \tag{11.4}$$

then the unitarity of V_0 together with the assumption that $\begin{bmatrix} \bar{x}_0 \\ \bar{u}_0 \\ \bar{\nu}_0 \end{bmatrix}$ are normalized uncorrelated, makes all the epsilon's normalized and uncorrelated as well. Filling in the epsilon's in equation 11.3 one has:

$$\begin{cases} x_1 &= M_1 \epsilon_2 + B_{o,0} \epsilon_3 \\ y_0 &= C_0 x_0 + \nu_0 = D_{o,0} \epsilon_3 \end{cases} \tag{11.5}$$

The output least squares estimate based on previous outputs (there are none and the only thing known about x_0 is that it is zero-mean) is trivially $\hat{y}_0 = 0$ and has covariance $D_{o,0}D'_{o,0} = R_0 + C_0P_0C'_0$, so that $\epsilon_3 = \bar{e}_{y,0}$ is the normalized output innovation. Next, since y_0 is the only stochastic quantity now known (the Kalman filter measures the output recursively starting at index 0) and $\epsilon_2 \perp \epsilon_3$, we have that $\hat{x}_1 = B_{o,0}\epsilon_3 = B_{o,0}\bar{e}_{y,0}$ and hence the estimation error on x_1 is $x_1 - \hat{x}_1 = M_1\epsilon_2$, with covariance $P_1 = M_1M'_1$ and normalized innovation $\bar{e}_{x,1} = \epsilon_2$. This identifies $M_1 = P_1^{1/2}$, while $B_{o,0}$ is traditionally called the *normalized Kalman gain* at this stage.

The first stage leaves us with a cascade of a unitary section and an outer section, and a propagation of the relevant quantities as shown in fig. 11.1, in which one should notice that $\mathbf{E}\widehat{x}_1\bar{x}'_1 = 0$, or, in other words: the best least squares estimate of x_1 is orthogonal on the (normalized) innovation, since $\mathbf{E}\widehat{x}_1\bar{x}'_1 = B_{o,0}(\mathbf{E}\epsilon_3\epsilon'_2) = 0$.

Step k

The outer-inner factorization as derived from the outer-inner theory (chapter 10) looks as follows in the general step k:

$$\begin{bmatrix} A_kM_k & B_kQ_k^{1/2} & 0 \\ C_kM_k & 0 & R_k^{1/2} \end{bmatrix} = \begin{bmatrix} 0 & M_{k+1} & B_{o,k} \\ 0 & 0 & D_{o,k} \end{bmatrix} V_k \tag{11.6}$$

in which V_k is unitary and M_k stands for what we have called Y_k in chapter 10: a change of notation because in this context, Y_k will soon receive a different meaning.

We assume as recursive hypothesis that the state input of the inner part (V_k) is the innovation $\bar{e}_k$ computed in the previous stage, and the state input of the outer filter is the k'th estimate $\widehat{x}_k$ with the property $\mathbf{E}(\widehat{x}_k\bar{e}'_k) = 0$, to show that the new inner and outer parts update these quantities for $k+1$. We assume, in addition, that the covariance of the input innovation is $M_kM'_k = \mathbf{E}(x_k - \widehat{x}_k)^2$, and have to update this property as well for the next stage (fig. 11.2.

The proof follows the same pattern as in step 0, and is only slightly more complicated. Using the k'th stage of the inner-outer factorization, let

$$\begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \end{bmatrix} := V_k \begin{bmatrix} \bar{e}_k \\ \bar{u}_k \\ \bar{\nu}_k \end{bmatrix} \tag{11.7}$$

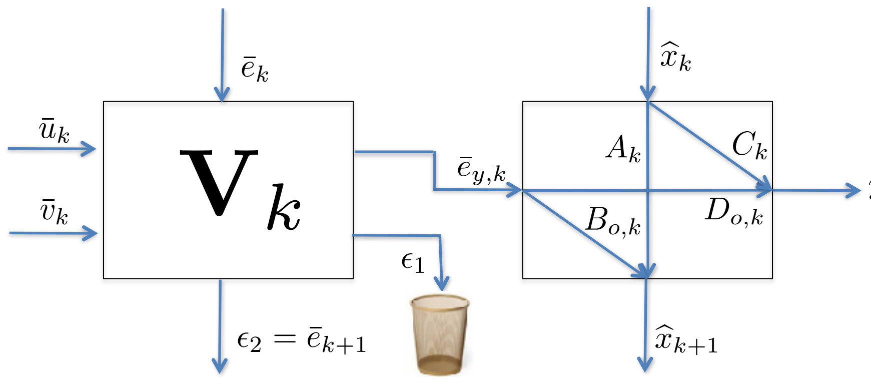


Figure 11.2: The k'th step in the outer-inner factorization of the Kalman filter model.

we have again that the ϵ's are orthonormal (with the inner products we use, this means: zero-mean, uncorrelated and of covariance one, properties that get preserved in the product with the unitary matrix V_k), and

$$\left\{ \begin{array}{rcl} x_{k+1} & = & M_{k+1}\epsilon_2 + B_{o,k}\epsilon_3 \\ y_k & = & C_k\widehat{x}_k + D_{o,k}\epsilon_3 \end{array} \right. , \tag{11.8}$$

in which $D_{o,k}$ and M_{k+1} are square non-singular, thanks to (1) the non-singularity of R_k and (2) the presumed minimality of the state space model (i.e., the observability of $\{A, C\}^2$). Remark now first that $D_{o,k}\epsilon_3$ is just zero mean noise added to the known quantity $C_k\widehat{x}_k$, so that $\widehat{y}_k = C_k\widehat{x}_k$, where $\widehat{y}_k$ is the estimate of y_k given $y_{0:k-1}$. After measuring y_k, also $\epsilon_3 = D_{o,k}^{-1}(y_1 - \widehat{y}_1) = \bar{e}_{y,k}$ is known, and it follows from the first equation, that $\widehat{x}_{k+1} = B_{o,k}\bar{e}_{y,k}$ and that $x_{k+1} - \widehat{x}_{k+1} = M_{k+1}\epsilon_2$, in which ϵ_2 is zero-means, second order white noise. The output of the inner part V_k is hence $\bar{e}_{k+1}$ and the covariance of the estimation error $x_{k+1} - \widehat{x}_{k+1} = M_{k+1}\epsilon_2$ is $\mathbf{E}(x_{k+1} - \widehat{x}_{k+1})^2 = M_{k+1}M'_{k+1}$.

The square root of the covariance of the normalized output innovation is $D_{o,k}$, commonly written as $R_{e_{y,k}}^{1/2}$, while $B_{o,k}$ is commonly known as the *normalized Kalman gain* $\bar{K}_k$. The factorization and the resulting identifications in the more traditional notation, also with $M_k = P_k^{1/2}$ with P_k the covariance of the innovation in x_k are shown in fig. 11.3. The resulting Kalman estimation filter is then the inverse of the outer factor, shown in fig. 11.4.

[2]For a proof of this important but technical point, see the notes at the end of the chapter

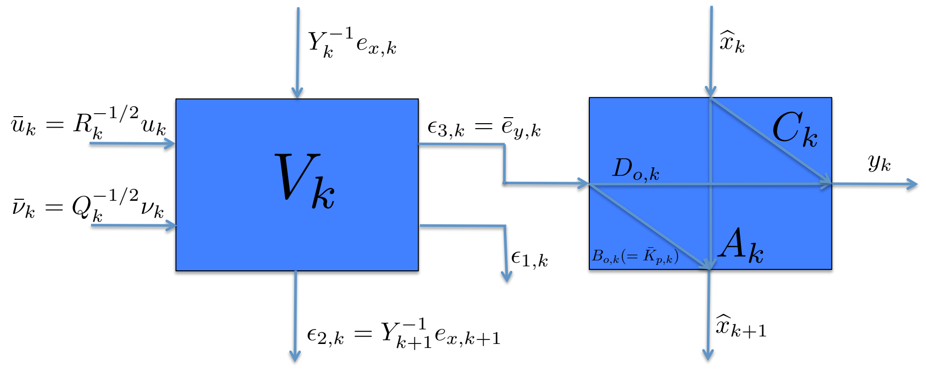

$Y_k^{-1}e_{x,k}$
$\bar{u}_k = R_k^{-1/2}u_k$
$\bar{\nu}_k = Q_k^{-1/2}\nu_k$
V_k
$\epsilon_{3,k} = \bar{e}_{y,k}$
$\epsilon_{1,k}$
$\epsilon_{2,k} = Y_{k+1}^{-1}e_{x,k+1}$
$\widehat{x}_k$
C_k
$D_{o,k}$
y_k
$B_{o,k}(= \bar{K}_{p,k})$
A_k
$\widehat{x}_{k+1}$

Figure 11.3: Outer-inner factorization for the case of the Kalman estimation filter.

Smoothing

A further quantity that is worth noting is the so-called instantaneous smoothed estimate defined as $f_k = \mathbf{E}[x_k|y_{0:k}]$ (while $\hat{x}_k = \mathbf{E}[x_k|y_{0:k-1}]$), and given by $f_k = \hat{x}_k + P_kC_k'R_{y,k}^{-1}(y_k - C_k\hat{x}_k)$, so that $\hat{x}_{k+1} = A_kf_k$, because $\bar{K}_{p,k} = A_kP_kC_k'R_{y,k}^{-1/2}$, which follows directly from the outer-inner factorization. The important thing is that $\hat{x}_{k+1}$ only depends on $\hat{x}_k$ and y_k (and not on x_k). It is a numerically stable dependence because of the invertibility of the outer factor, V_k taking the role of handling the innovations. This principle can be generalized to nonlinear systems, as is outer-inner factorization, see the chapter on particle filtering for a first approach. S

11.4 Discussion issues

- We have required minimality of the Kalman model filter. It is interesting to see how this assumption is used. Let us first check the time-variant case.

  In stage 1, minimality amounts, perhaps surprisingly, to $\bigvee(B_0, A_0)$ spanning the full dimension of x_1 (or, $w'\,[B_0 \;\; A_0] = 0 \Rightarrow w' = 0$). This is because the almost trivial -1 stage is lacking in the treatment. In that stage the state, which for a strictly orthodox time-variant treatment is needed, the (zero mean stochastic) state x_0 with (non-singular) covariance $P_0 = M_0M_0'$ is generated simply by inputing it. The model

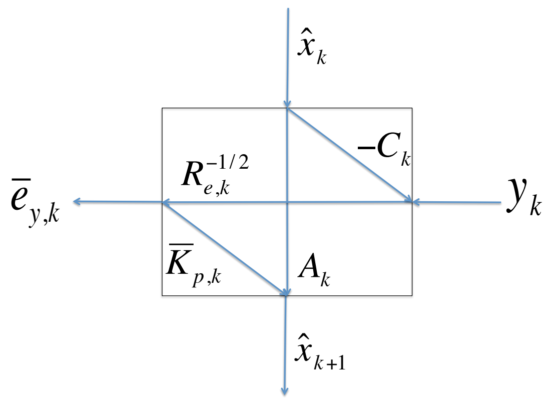


Figure 11.4: The Kalman estimation filter is the inverse of the outer factor.

and outer-inner factorization for stage -1 is then simply

$$\begin{bmatrix} | & P_0^{1/2} \\ \cdot & - \end{bmatrix} = \begin{bmatrix} P_0^{1/2} & | \\ - & \cdot \end{bmatrix} \begin{bmatrix} | & I \\ \cdot & - \end{bmatrix} \tag{11.9}$$

That makes $B_{-1} = P_0^{1/2}$, the other entries in the system model empty and $M_0 = P_0^{1/2}$. The reachability matrix $\mathcal{R}_1 = \begin{bmatrix} B_0 & A_0 B_{-1} \end{bmatrix}$ is hence to be supposed non-singular for the system to be reachable (at index point 0).

In the general $\underline{\text{stage k}}$, the inductive hypothesis makes M_k non-singular, and one has to prove M_{k+1} non-singular as well. For that it is necessary and sufficient that $[B_k \;\; A_k M_k]$ be non-singular. From the inner-outer theory (or by direct calculation), we have the minimal factorization $M_k = \mathcal{R}_{k-1}\mathcal{R}'_{V,k-1}$ for all k, which implies that both $\mathcal{R}_{k-1}$ and $\mathcal{R}_{V,k-1}$ are non-singular. That $\mathcal{R}_k$ will be non-singular is assumed by the condition of minimality, so the proof amounts to showing that $\mathcal{R}_{V,k}$ also has full range. However, by construction in the square-root algorithm, M_{k+1} has full row-range, which means, for any conformal vector w, that $M_{k+1}w = 0 \Rightarrow w = 0$. But, $M_{k+1}w = 0 \Rightarrow \mathcal{R}_k\mathcal{R}'_{V,k}w = 0$. Hence, suppose $\mathcal{R}_{V,k}$ singular, then there would be $w \neq 0$ so that $\mathcal{R}'_{V,k}w = 0$ and hence $M_{k+1}w = 0$ with $w \neq 0$, contradicting the minimality-by-construction of M_{k+1}.

(A more direct proof than in the previous paragraph can also be obtained as follows. The 'stage k' can be brought back tot the 'stage 0' case, by compressing all the stages from 0 to k into one global stage. With the original minimal filter realization at index i given by $\begin{bmatrix} A_i & B_i \\ C_i & D_i \end{bmatrix}$, then the filter section combining the steps $0 \cdots k$ has the (cascaded) realization $\begin{bmatrix} A_{[k]} & B_{[k]} \\ C_{[k]} & D_{[k]} \end{bmatrix}$, in which

$$\left\{ \begin{array}{lcl} A_{[k]} & := & A_k \cdots A_0 \\ B_{[k]} & := & \left[\begin{array}{ccc} (A_k \cdots A_1 B_0) & \cdots & B_k \end{array} \right] (= \left[\begin{array}{cc} A_k B_{[k-1]} & B_k \end{array} \right]) \\ C_{[k]} & := & \text{col} \left[\begin{array}{ccc} C_0 & \cdots & (C_k A_{k-1} \cdots A_0) \end{array} \right] = (\text{col} \left[\begin{array}{cc} C_{[k-1]} & C_k A_{[k-1]} \end{array} \right]) \\ D_{[k]} & := & T_{0:k,0:k}. \end{array} \right. \tag{11.10}$$

When outer-inner factored, this filter produces, as before in the initial step, both the estimate $\hat{x}_{k+1}$ and the normalized innovation $P_{k+1}^{-1/2} e_{x,k+1}$, for exactly the same reasons as before (we do not repeat the argument, because this cascaded filter can just be considered to be the initial step in its own right). This brings the proof back to the proof given for 'stage 0', now with a complete reachability matrix $\begin{bmatrix} B_{[k]} & A_{[k]} \end{bmatrix}$ reaching up to x_{k+1}.)

In the *time invariant case*, either one may reduce the problem to the time-variant case, where one starts out measuring at index point 0 and assumes an initial state x_0 whose covariance is known. This actually reduces the problem to the previous case, and one may then study how the estimation evolves with increasing indices, the only difference being that now the subsequent A_k, C_k, Q_k and R_k are all the same. In case the model is internally stable, in the sense that $\lim_{k\to\infty} A^k = 0$, then it is easy to see that M_k eventually reaches a non-singular fix-point (still assuming minimality of the model representation, of course), and the prediction filter becomes gradually independent of the initial state. Alternatively, one may study time-invariant outer-inner factorizations of the original transfer function directly, which would also involve some stability conditions on the original model. This latter endeavor goes beyond our treatment here.

- *Riccati equation.* We have derived the Kalman filter using outer-inner factorization. Traditionally, an opposite road is followed: one derives the Kalman estimation by 'brute force', just solving the equations that follow from the Wiener principle mentioned at the start of this chapter and so deriving a quadratic equation for the covariance P_{k+1} of the innovation recursively from P_k for each k. Based on these derivations, Kailath and his coworkers [21] derived what they called the *square root algorithm* for updating the square-root $P_k^{1/2}$ of the covariance rather than the covariance itself. This square-root algorithm is nothing but our outer-inner factorization, and, as we have done, the direct derivation of the Kalman filter from outer-inner appears to be simpler and more insightful than the original. The resulting quadratic equation is called a *Riccati equation*, in our case a recursive equation (the original term Riccati equation was in honor of the mathematician who studied differential equations with a quadratic term). It is easy to derive this Riccati equation directly from the outer-inner or, equivalently, square-root equation. From equation 11.6, which we can write shorthand $\mathbf{T}_k = \mathbf{T}_{o,k} V_k$, we find, after post-multiplication with the transpose and using $V_k V_k' = I$, $\mathbf{T}_k \mathbf{T}_k' = \mathbf{T}_{o,k} \mathbf{T}_{o,k}'$, which written out produces

$$\left\{ \begin{array}{ccc} P_{k+1} + B_{o,k} B_{o,k}' & = & A_k P_k A_k' + B_k Q_k B_k' \\ B_{o,k} D_{o,k}' & = & A_k P_k C_k' \\ D_{o,k} D_{o,k}' & = & C_k P_k C_k' + R_k \end{array} \right. \tag{11.11}$$

From these equations, $B_{o,k}$ and $D_{o,k}$ can be eliminated and introduced in the equation for P_{k+1}. This produces in sequence $D_{o,k} = (C_k P_k C_k' + R_k)^{1/2}$ (which is invertible thanks to the non-singularity of R_k), then $B_{o,k} = A_k P_k C_k' (C_k P_k C_k' + R_k)^{-1/2}$ and, finally, the recursive matrix Riccati equation

$$P_{k+1} = A_k P_k A_k' + B_k Q_k B_k' - A_k P_k C_k' (R_k + C_k P_k C_k')^{-1} C_k P_k A_k' \tag{11.12}$$

Quite a bit of effort in the literature is devoted to study this equation and derive properties that can often easily be obtained just from the outer-inner factorization (like the existence of a guaranteed positive definite solution).

- The Kalman filter in the form presented so far is purely predictive: it does not use any future outputs. Often, and in particular in image processing, 'future' information is present. Assume, e.g., that you would dispose not only of y_k, but also of y_{k+1} at stage k. This is called in the literature as 'smoothing'. A nice topic for discussion is how the Kalman filter can be extended to that case.

- Another good topic for discussion is what to do with Gaussian processes that are not zero mean, or, more generally, non-linear processes.

11.5 Notes

The Kalman filter was conceived and derived by a few people in parallel in the period 1958-1961, to name: Stratonovitch, Kalman, Bucy and somewhat earlier by Thiele and Swerling. It has played a key role in the development of the Appolo navigation computer, as was devised by Schmidt of the Nasa research navigation research group at Ames Laboratories in Mountain View, after a visit of Kalman there. The great advantage of the new approach was its recursive character: it allowed, given the available data at a certain point in time, to make the best possible incremental choices for the next step. This meant in the first place that one would have to estimate as accurately as possible the state of the rocket, given noisy position and velocity measurements and, next, derive from the estimates the necessary controls to move that state forward in the desired way (the intended trajectory, or a new updated desirable one). 'Reachability' and 'observability' obviously had to play a central role there.

From that point on, Kalman started to develop the 'state space theory' for dynamical systems in a systematic way, focussing on these most essential concepts (as we are also doing in this book, in the wake of the approach proposed by Kalman). Although the first derivations of the Kalman filter were for time variant or even non-linear systems, it soon seemed that the time invariant case would lead to a richer algebraic content, and, moreover, most of the community was geared towards LTI systems and input-output rather than state space descriptions. The connection between the matrix algebra for state space descriptions (the A,B,C,D formalism) and the traditional transfer function approach was soon firmly established and a host of new algebraic results followed that strengthened both sides: it provided the state space people with the firm algebraic foundation of module theory (polynomial and series calculus) and the transform calculus with new ways of characterizing the 'degree' of a system instead of the cumbersome Smith-McMillan form. It all seemed like an ideal symbiosis, be it that it could not be generalized neither to time variant nor to non-linear systems.

With the advent of numerical calculus, the situation changed dramatically, and the emphasis returned from transfer function calculus to matrix algebra. The method of choice in numerical analysis is the use of orthogonal (or unitary) transformations, and it is no wonder that pretty soon after the discovery of the Kalman filter and the rather ad hoc (Bayesian) computations connected to it, came the idea of using the notion of 'innovation' instead, which, inductively, lead to a new type of algorithm to compute the Kalman estimation filter based on orthogonal computations: the 'square root algorithm', first proposed by Kailath. It was later found out that this algorithm is actually a special case of inner-outer factorization. Turning the tables around, one can use it as a basis to develop the necessary Bayesian innovation theory needed for the Kalman filter. This has been the approach that we have followed in this book.

Chapter 12

Least squares optimal control

An estimation filter looks forward: given past performance indicators (represented as outputs) and a model of the system including stochastic disturbing terms, it tries to estimate future states. In the previous chapter we discovered that an outer-inner factorization achieves this feat recursively for the linear least square estimation error (llse). Often, one would not only like to predict where a system is going to, but to control its behavior 'on the fly' so that it goes where one wants it to go, and this, again, in a least squares optimal fashion. That is 'least squares optimal control'. In this chapter we explore this problem in the context of our LTV system theory, using the classical modeling of the optimal control problem as it was introduced by a.o. Bellman, using a simple stochastic model that covers the least squares tracking problem for incremental models. Just as in the case of optimal least square estimation treated in the previous chapter, we shall find that a simple inner-outer factorization (dual to the previous) solves the problem directly, and leads to the classical results originally obtained by Bellman.

Menu

Hors d'oeuvre

The optimal control situation considered

Main course

The solution through Moore-Penrose inversion via inner-outer factorization

Dessert

Dynamic programming and discussions

12.1 The assumptions

Let us assume we are given a system, partially described by a state evolution equation

$$x_{k+1} = A_k x_k + B_k u_k + w_k, \tag{12.1}$$

in which w_k is a zero mean noise term, assumed to be an independent, zero mean process (e.g., Gaussian with $\mathbf{E} w_k w'_\ell = 0$ for $k \neq \ell$, or, otherwise such that w_k and w_ℓ are independent stochastic variables for all $k \neq \ell$)) **and** a known initial state x_0, with the goal to choose a sequence of inputs $u_0, \cdots, u_n$ (an 'effort') so that a sequence of subsequent states $x_1, \cdots, x_{n+1}$ results with the property that some desirable 'cost function' or 'objective function' involving these inputs and states is optimized. For each input vector u_k and each state vector x_k we may choose a contribution to the cost as $u'_k N'_k N_k u_k$ and $x'_k M'_k M_k x_k$, where the matrices N_k and M_k, which take into account the specific circumstances of the problem (e.g., different units or scales, possibilities etc...), may be chosen arbitrarily to some extent (see further: we shall require the N_k to be square non-singular, for simplicity, meaning that every input has an associated cost). The total cost function is then

$$\mathcal{L} := \mathbf{E}\left(\sum_{k=1}^{n+1} x'_k M'_k M_k x_k + \sum_{k=0}^{n} u'_k N'_k N_k u_k\right) \tag{12.2}$$

(we use the symbol '$\mathcal{L}$' to illustrate its relation to the classical 'Lagrangian'.). As the x_k are functions of the u_k through the system equations, the problem is to find the sequence of inputs $\widehat{u}_k : k = 0 \cdots n$ that minimize $\mathcal{L}$, often written as

$$\widehat{u}_{0:n} = \operatorname{argmin}_{u_k : k=0\cdots n} \mathcal{L}. \tag{12.3}$$

12.2 System representation

The optimal control situation admits an attractive recursive system representation shown in fig. 12.1. It takes as inputs (1) the given starting state

x_0 and (2) the input sequence $u_k, k = 0 \cdots n$ chosen, and produces as outputs the local contributions to the performance in "square root form" namely $[L_k]_{k=0}^{n+1} = \begin{bmatrix} M_k x_k \\ N_k u_k \end{bmatrix}_{0:n+1}$ (with M_0 empty), the actual full performance being $\mathcal{L} = \mathbf{E}(L'L)$. The local system description for optimization purposes is then

$$\left\{ \begin{array}{ccc} x_{k+1} & = & A_k x_k + B_k u_k + w_k \\ L_k & = & \begin{bmatrix} M_k \\ 0 \end{bmatrix} x_k + \begin{bmatrix} 0 \\ N_k \end{bmatrix} u_k \end{array} \right. \tag{12.4}$$

The overall input-output equations after $n+1$ steps for this representative system model are read from the diagram in fig. 12.1 as

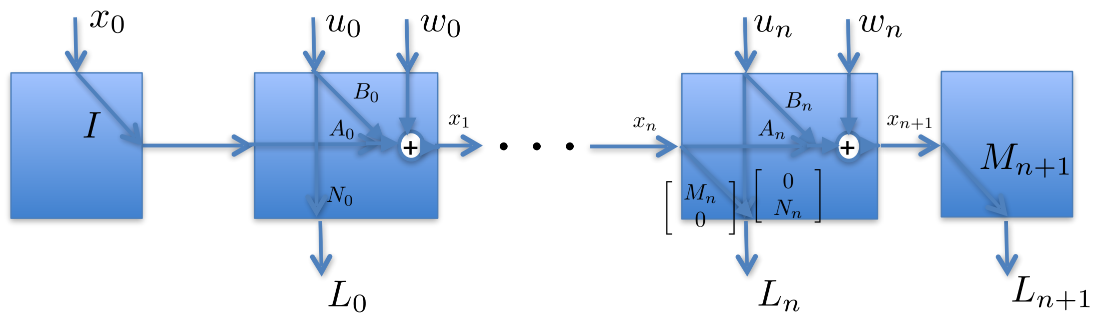


Figure 12.1: System model of the optimal control situation.

$$X_0 x_0 + Su + T_w w = L \tag{12.5}$$

where

$$X_0 := \begin{bmatrix} 0 \\ M_1 A_0 \\ \vdots \\ M_{n+1} A_n \cdots A_0 \end{bmatrix}, \; S := \left[\begin{array}{c|c|c|c|c} N_0 & & & & \\ \hline M_1 B_0 & 0 & & & \\ 0 & N_1 & & & \\ \hline M_2 A_1 B_0 & M_2 B_1 & 0 & & \\ 0 & 0 & N_2 & & \\ \hline \vdots & \ddots & \ddots & \ddots & \\ \hline M_n A_{n-1} \cdots A_1 B_0 & & & & 0 \\ 0 & \cdots & \cdots & \cdots & N_n \\ \hline M_{n+1} A_n \cdots A_1 B_0 & \cdots & \cdots & \cdots & M_{n+1} B_n \end{array} \right] \tag{12.6}$$

and

$$T_w := \left[\begin{array}{c|c|c|c|c} 0 & & & & \\ \hline M_1 & 0 & & & \\ \hline M_2A_1 & M_2 & 0 & & \\ \hline \vdots & \ddots & \ddots & \ddots & \\ \hline M_nA_{n-1}\cdots A_1 & \cdots & \cdots & \cdots & 0 \\ \hline M_{n+1}A_n\cdots A_1 & \cdots & \cdots & \cdots & M_{n+1} \end{array}\right]. \tag{12.7}$$

X_0x_0 is given and known. S is the (causal) 'system matrix' for this case, which also happens to have a left inverse (is left outer) when all the N_k are square invertible. The overall goal is to find an input sequence u that produces the minimal cost $\mathbf{E}(L'L)$. This problem is solved directly by the Moore-Penrose pseudo-inverse $S^\dagger$ of S, because S is left-invertible, and

$$\mathbf{E}(L'L) = \mathbf{E}[(X_0' + u'S')(X_0 + Su)] + \mathbf{E}(w'T_w'T_ww) \tag{12.8}$$

in which the second term is a independent of u (as well as X_0x_0) and the minimum is obtained by minimizing the first term, giving[1]

$$u_{\min} = -S^\dagger X_0 \tag{12.9}$$

and the whole exercise reduces to finding an efficient, preferably recursive $S^\dagger$. This we do in the next section by inner-outer factorization, giving $S = US_o$ with U isometric ($U'U = I$) and, in this case, S_o square invertible, because the original S has already been chosen left invertible, thanks to the choice for square and invertible N_k's. Therefore $S^\dagger = S_o^{-1}U'$. The minimal error is then given by $L = (I - SS^\dagger)X_0 = (I - UU')X_0$, which is orthogonal on the projection $UU'X_0$ of X_o on the space spanned by the columns of U, which in turn is the range of S (see fig. 12.2). The minimally obtainable cost is then given by

$$\mathbf{E}(L_{\min}'L_{\min}) = X_0'(I - UU')X_0 + \mathbf{E}(w'T_w'T_ww) \tag{12.10}$$

which is of the form $x_0'\mathbf{C}x_0$+constant term for some cost matrix $\mathbf{C}$—quadratic in x_0 as is to be expected.

[1]**Proof:** let $h := u - u_{\min}$, then $\mathbf{E}(L'L) = \mathbf{E}[(I - SS^\dagger)'X_0'X_0(I - SS^\dagger)] + \mathbf{E}(h'S'Sh)$ because $S'(I - SS^\dagger) = 0$, which will be minimum if and only if $h = 0$. Same reasoning as in the deterministic Moore-Penrose case, now in a probabilistic context.

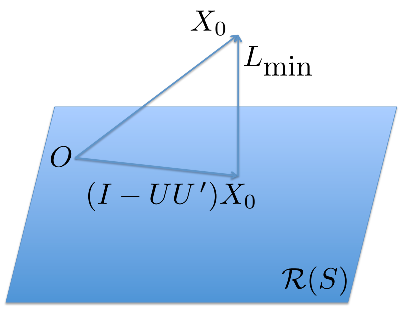


Figure 12.2: The minimally obtainable cost $L_{\min}$ is orthogonal on the range of S.

12.3 Inner-outer factorization

Preliminary remark: from the analysis in the previous section, we see that the noise term $T_w w$ plays no role in the optimization process, which only involves the 'system matrix' S and not the noise processing matrix T_w. This section details the recursive inner-outer factorization of S, which also would correspond to the non-stochastic case. An interpretation for the connecting matrices Y_k in the stochastic context appears in the next section. As a consequence, the states $\bar{x}_k$ appearing in this section are not the same as those appearing in the previous section, the latter being contaminated with noise. One shows: $\bar{x}_k = \mathbf{E}x_k$.

Let $S = US_o$ be an inner-outer factorization of S, then the general recursion for inner-outer factorization, repeated here for convenience, is given by

$$\left[\begin{array}{c|c} YA & YB \\ \hline C & D \end{array}\right] = \left[\begin{array}{ccc} B_W & A_U & B_U \\ \hline D_W & C_U & D_U \end{array}\right] \left[\begin{array}{c|c} 0 & 0 \\ Y^{<-1>} & 0 \\ C_o & D_o \end{array}\right] \qquad (12.11)$$

where $S = US_o$, $S = D + C(I - ZA)^{-1}ZB$, $S_o = D_o + C_o(I - ZA)^{-1}ZB$, $W = D_W + C_U(I - ZA_U)^{-1}ZB_W$ defines the co-kernel of S and $U = D_U + C_U(I - ZA_U)^{-1}ZB_U$ provides a (causal) orthonormal basis for the range of S. It is a backward recursion starting from the last relevant realization. D_o and $Y^{<-1>}$ form row bases (are right invertible), and are obtained by a QL factorization that works starting in the South-East (lower-right) corner, working upwards.

The recursion starts at $n+1$ in the present case. We simply have Y_{n+1} empty, and
<u>Step $n+1$</u>

$$\left[\begin{array}{c|c} - & \cdot \\ \hline M_{n+1} & | \end{array}\right] = \left[\begin{array}{cc} - & \cdot \\ \hline I & | \end{array}\right] \left[\begin{array}{c|c} M_{n+1} & | \\ - & \cdot \end{array}\right] \tag{12.12}$$

(using row and column separators to indicate forced partitions in the respective matrices). Hence we have $Y_n := M_{n+1}$, and $S_{o,n+1} = \left[\begin{array}{c|c} | & | \\ \hline \cdot & \cdot \end{array}\right]$.
<u>Step n</u>

$$\left[\begin{array}{c|c} Y_n A_n & Y_n B_n \\ \hline M_n & 0 \\ 0 & N_n \end{array}\right] = \left[\begin{array}{ccc} B_{W,n} & A_{U,n} & B_{U,n} \\ \hline D_{W,n} & C_{U,n} & D_{U,n} \end{array}\right] \left[\begin{array}{c|c} 0 & 0 \\ Y_{n-1} & 0 \\ C_{o,n} & D_{o,n} \end{array}\right] \tag{12.13}$$

in which $D_{o,n}$ is necessarily square non-singular thanks to the non-singularity assumption on N_n. We see that if M_n is square non-singular, then also Y_{n-1} will be, but this assumption is not necessary for the inner-outer factorization to produce a fully outer S_o with present realization $\begin{bmatrix} A_n & B_n \\ C_{o,n} & D_{o,n} \end{bmatrix}$.
<u>$\cdots$</u> (like step n)
<u>Step 0</u> (almost final)

$$\left[\begin{array}{c|c} | & Y_0 B_0 \\ \hline | & N_0 \end{array}\right] = \left[\begin{array}{cc} B_{W,0} & B_{U,0} \\ \hline D_{W,0} & D_{U,0} \end{array}\right] \left[\begin{array}{c|c} | & 0 \\ | & D_{o,0} \end{array}\right] \tag{12.14}$$

with realization for $S_{o,0}$ given by $\left[\begin{array}{c|c} | & B_0 \\ \hline | & D_{o,0} \end{array}\right]$.

Alternatively, one can keep the last step conformal with the previous ones (what we shall do in the sequel), which produces (with different $B_{W,0}$ and $D_{W,0}$, previously there was no $A_{U,0}$ and $C_{U,0}$. However, $B_{U,0}$ and $D_{U,0}$ are the same.)

$$\left[\begin{array}{c|c} Y_0 A_0 & Y_0 B_0 \\ \hline 0 & N_0 \end{array}\right] = \left[\begin{array}{ccc} B_{W,0} & A_{U,0} & B_{U,0} \\ \hline D_{W,0} & C_{U,0} & D_{U,0} \end{array}\right] \left[\begin{array}{c|c} 0 & 0 \\ Y_{-1} & 0 \\ C_{o,0} & D_{o,0} \end{array}\right] \tag{12.15}$$

<u>Step -1</u>
The final step is against almost trivial, but important because it settles

boundary conditions. The backward recursion gives, with $B_{-1} = I$:

$$\begin{bmatrix} | & Y_{-1} \\ \cdot & - \end{bmatrix} = \begin{bmatrix} | & I \\ \cdot & - \end{bmatrix} \begin{bmatrix} \cdot & - \\ | & Y_{-1} \end{bmatrix} \tag{12.16}$$

so that $B_{U,-1} = I$ and $D_{o,-1} = Y_{-1}$ with all other entries empty.

The connecting sequence Y_k satisfies the recursion $Y_{k-1} = A'_{U,k} Y_k A_k + C'_{U,k} M_k$, with initial value $Y_n = M_{n+1}$. Fig. 12.3 displays a graphical representation of the result (we do not show the kernel part represented by $W = D_W + C_U(I - ZA)^{-1} Z B_W$.). The figure also shows the signal propaga-

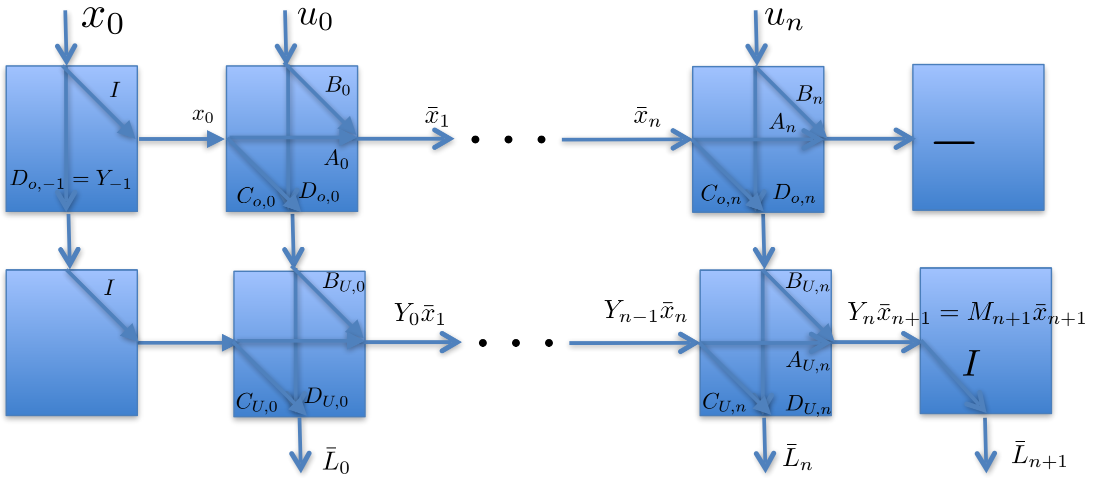


Figure 12.3: Inner-outer factorization of the system matrix S.

tion in the factored model. This assignment is worth a separate proposition.

Proposition 3 *Stage k in the inner-outer factorization of the full model (shown in fig. 12.1) propagates* $\begin{bmatrix} \bar{x}_k \\ Y_{k-1}\bar{x}_k \end{bmatrix}$ *to* $\begin{bmatrix} \bar{x}_{k+1} \\ Y_k\bar{x}_{k+1} \end{bmatrix}$*, where the $\{\bar{x}_k\}$ is any sequence of states in S.*

Proof

Recursively. First remark that the propagation of the state in the outer factors is the same as in the original model. Furthermore, at any stage k we have, applying $\begin{bmatrix} \bar{x}_k \\ u_k \end{bmatrix}$ to the inner-outer factorization

$$\begin{bmatrix} A_{U,k} & B_{U,k} \\ C_{U,k} & D_{U,k} \end{bmatrix} \begin{bmatrix} Y_{k-1}\bar{x}_k \\ C_{o,k}\bar{x}_k + D_{o,k}u_k \end{bmatrix} = \left[\begin{array}{c} Y_k\bar{x}_{k+1} \\ \hline M_k\bar{x}_k \\ N_k u_k \end{array}\right] = \left[\begin{array}{c} Y_k\bar{x}_{k+1} \\ \hline \bar{L}_k \end{array}\right]. \quad (12.17)$$

The input at stage 0 follows from the initial step -1, inducing the recursion. QED

Fig. 12.3 shows the division of labor between the outer and the inner filter clearly. The input of the inner filter consists of two orthogonal components: the first entry $Y_{-1}\bar{x}_0$ and the sequence $\{y_{o,k} := C_{o,k}\bar{x}_k + D_{o,k}u_k\}_{k=0:n}$ (these are all orthogonal on each other as different components of a single vector) while the output of the inner filter is $\bar{L}_{o:k+1}$, whose (quadratic) norm squared equals

$$\|\bar{L}\|^2 = \|\mathrm{col}\,(Y_{-1}x_0, y_{o,0:n})\,\|^2 = \|Y_{-1}x_0\|^2 + \sum_{k=0}^{n} \|y_{o,k}\|^2. \quad (12.18)$$

Because the outer filter is invertible, it can set the inputs such that all $y_{o,k} = 0$ by choosing $u_k = -D_{0,k}^{-1}C_{0,k}x_k$, thereby minimizing the cost to $\|Y_{-1}X_0\|^2 +$ [noise variance independent from inputs]. The outer filter produces the 'feedback gain' $F_k = D_{0,k}^{-1}C_{0,k}$ as its inverse with zero input, and the inner filter takes care of the orthogonal decomposition and the production of the resulting optimal cost.

12.4 Dynamic programming

Although we can produce the solution directly with the formalism developed so far and some brute force (see the notes at the end), a more enlightening znd adaptive approach follows the inner-outer recursion. Closely related to the inner-outer recursion, is a recursion on the optimality of the solution, introduced by Bellman [7] and known as *dynamic programming.*

Let us first observe that in the way we have stated the optimization problem, everything depends solely on the initial state x_0 once the system model is given, since the obejctive is to find the optimal input $\widehat{u}_{0:n}$ that will minimize the cost $\mathcal{L}(x_0, u_{0:n})$ of the evolution from index 0 to n. Both x_0 and the sequence of inputs $u_{0:n}$ determine the sequence of states $x_{1:n+1}$ (except for intervening noise), and hence the overall cost. Taking the minimum over the inputs eliminates them as free variables and leaves the initial state vector x_0 as only free variable.

This being established, the next step is to consider how an optimal trajectory depends on an intermediate state x_k, with $0 \leq k \leq n$. The cost of that part of the trajectory that starts with x_k is only dependent on the states $x_{k:n+1}$ and the inputs $u_{k:n}$, due to the formulation of the problem (we do not 'look back' and do not let future costs depend on past states except through x_k). This means that the optimal trajectory from 0 to $n+1$ also has to be optimal from k to $n+1$, given the state x_k as lying on that optimal trajectory. But this latter optimal trajectory will again be solely dependent on x_k.

As a consequence, one can recursively compute optimal trajectories starting from the end point $n+1$. To formalize this idea, let us define the optimal cost of a trajectory starting at state x_k as $\widehat{\mathcal{L}}(x_k, [k:n])$, then we can write

$$\widehat{\mathcal{L}}(x_0, [0:n]) = \min_{u_{[o,k]}, x_{k+1}} (\mathcal{L}(x_0, u_{[0:k]}) + \widehat{\mathcal{L}}(x_{k+1}, [k+1:n])). \tag{12.19}$$

The dynamic programming recursion is then obtained by applying the principle moving from index point $k+1$ to index point k:

$$\widehat{\mathcal{L}}(x_k, [k:n]) = \min_{u_k, x_{k+1}} (\mathcal{L}(x_k, u_k) + \widehat{\mathcal{L}}(x_{k+1}, [k+1:n])), \tag{12.20}$$

the gain being that now the optimization can be done working on just two parameter vectors u_k and x_{k+1}, instead of the whole sequence $u_{k:n}$.

In the case of quadratic optimization, this latter expression simplifies dramatically! Let us fist look at stage n, and use the inner-outer factorization described in the previous section.

Stage n

The cost of state x_{n+1} is simply (entering stage $n+1$, x_{n+1} is assumed known)

$$\mathcal{L}(x_{n+1})(= \widehat{\mathcal{L}}(x_{n+1})) = x'_{n+1} M'_{n+1} M_{n+1} x_{n+1} + \mathbf{E}(w'_n M'_{n+1} M_{n+1} w_n). \tag{12.21}$$

It is quadratic in x_{n+1}. The situation (model filter at stage n) is shown in fig. 12.4. We have

$$\begin{bmatrix} L_{n+1} \\ L_n \end{bmatrix} = \begin{bmatrix} M_{n+1}A_n \\ \hline M_n \\ 0 \end{bmatrix} x_n + \begin{bmatrix} M_{n+1}B_n \\ \hline 0 \\ N_n \end{bmatrix} u_n + \begin{bmatrix} M_{n+1} \\ \hline 0 \\ 0 \end{bmatrix} w_n \tag{12.22}$$

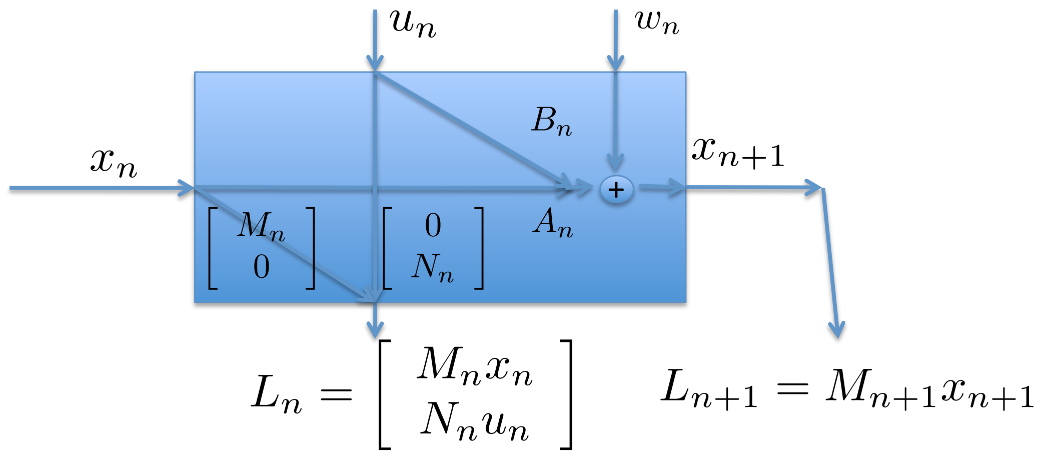

u_n
w_n
B_n
x_n
x_{n+1}
$+$
A_n
$\begin{bmatrix} M_n \\ 0 \end{bmatrix}$
$\begin{bmatrix} 0 \\ N_n \end{bmatrix}$
$L_n = \begin{bmatrix} M_n x_n \\ N_n u_n \end{bmatrix}$
$L_{n+1} = M_{n+1} x_{n+1}$

Figure 12.4: The model filter at stage n.

where now x_n is fixed and w_n is statistically independent of everything else. The overall cost will be quadratically minimized for the optimal control $\widehat{u}_n$ given by

$$\widehat{u}_n = -\begin{bmatrix} M_{n+1}B_n \\ \hline 0 \\ N_n \end{bmatrix}^{\dagger} \begin{bmatrix} M_{n+1}A_n \\ \hline M_n \\ 0 \end{bmatrix} x_n \tag{12.23}$$

The inner-outer factorization with $Y_n = M_{n+1}$ provides for this:

$$\left[\begin{array}{c|c} Y_n A_n & Y_n B_n \\ \hline M_n & 0 \\ 0 & N_n \end{array}\right] = U_n \left[\begin{array}{c|c} 0 & 0 \\ Y_{n-1} & 0 \\ C_{o,n} & D_{o,n} \end{array}\right] \tag{12.24}$$

in which $D_{o,n}$ is square invertible by a similar assumption on N_n. It follows that (the pseudo-inverse is unique in this case)

$$\begin{bmatrix} M_{n+1}B_n \\ \hline 0 \\ N_n \end{bmatrix}^{\dagger} = \begin{bmatrix} 0 & 0 & D_{o,n}^{-1} \end{bmatrix} U_n' \text{ and } \begin{bmatrix} M_{n+1}A_n \\ \hline M_n \\ 0 \end{bmatrix} = U_n \begin{bmatrix} 0 \\ Y_{n-1} \\ C_{o,n} \end{bmatrix} \tag{12.25}$$

and hence the optimal control $\widehat{u}_n = -F_n x_n$ with $F_n = D_{o,n}^{-1} C_{o,n}$.

What about the optimal cost with x_n given and assumed fixed at this point? We have $x_{n+1} = A_n x_n + B_n u_n + w_n$, hence $\bar{x}_{n+1} = A_n x_n + B_n u_n$

applying $\begin{bmatrix} x_n \\ u_n \end{bmatrix}$ both sides of eq. 12.24, using the unitarity of U_n and putting $C_{o,n}x_n + D_{o,n}u_n = 0$ (the optimal choice for each fixed x_n), we obtain, using this optimal control

$$x_n' Y_{n-1}' Y_{n-1} x_n = L_n' L_n + \bar{x}_{n+1}' M_{n+1}' M_{n+1} \bar{x}_{n+1}, \tag{12.26}$$

It follows that the optimal cost, given x_n, for stages n and $n+1$ is

$$\widehat{\mathcal{L}}(x_n, [n:n]) := x_n' Y_{n-1}' Y_{n-1} x_n + \mathbf{E}(w_n' M_{n+1}' M_{n+1} w_n), \tag{12.27}$$

hence, quadratic in x_n and with an input independent noise variance.

Concluding the discussion of stage n, we see that the inner-outer factorization gives *exceedingly simple* expressions for the optimal control $\widehat{u}_n = -F_n x_n$ with $F_n = D_{o,n}^{-1} C_{o,n}$ as a direct feedback on the state x_n, and the propagation of the optimal cost $\widehat{\mathcal{L}}(x_n, [n:n]) = x_n' Y_{n-1}' Y_{n-1} x_n + c(n)$, where $c(n)$ is a state independent constant term.

Stage $0 < k < n$:

With the notation introduced at the end of the treatment of stage n, the situation of stage k becomes identical, with k replacing n. In summary:

Initial data: optimal downstream cost $x_{k+1}' Y_k' Y_k x_{k+1} + c(k+1)$ [the recursion hypothesis].

Inner-outer factorization:

$$\left[\begin{array}{c|c} Y_k A_k & Y_k B_k \\ \hline M_k & 0 \\ 0 & N_k \end{array}\right] = U_n \left[\begin{array}{c|c} 0 & 0 \\ Y_{k-1} & 0 \\ C_{o,k} & D_{o,k} \end{array}\right] \tag{12.28}$$

Result:

feedback gain on stage k: $\widehat{u}_k = -F_k x_k$ with $F_k = D_{o,k}^{-1} C_{0,k}$

new optimal cost: $x_k' Y_{k-1}' Y_{k-1} x_k + c(k)$ with $c(k) = \mathbf{E}(w_k' Y_k' Y_k w_k) + c(k+1)$.

Stage 0:

Stage 0 is somewhat special because x_0 does not participate in the overall cost, as given initial state. This can of course simply be handled by the same formalism as before, putting $M_0 = [\cdot]$, but a quick analysis produces the same result, of course.

12.5 Computational aspects

Traditionally, the dynamic programming update is done through the recursion, obtained by squaring the inner-outer expression (putting for simplicity $\mathbf{L}_k := Y_k'Y_k$)

$$\begin{bmatrix} \mathbf{L}_k + C_{o,k}'C_{o,k} & C_{o,k}'D_{o,k} \\ D_{o,k}'C_{o,k} & D_{o,k}'D_{o,k} \end{bmatrix} = \begin{bmatrix} M_k'M_k + A_k'\mathbf{L}_{k+1}A_k & A_k'\mathbf{L}_{k+1}B_k \\ B_k'\mathbf{L}_{k+1}A_k & N_k'N_k + B_k'\mathbf{L}_{k+1}B_k \end{bmatrix}. \tag{12.29}$$

To solve the resulting system of equations, one first factorizes $N_k'N_k + B_k'\mathbf{C}_kB_k = D_{o,k}'D_{o,k}$ (which is square non-singular by assumption), and then eliminates $C_{o,k}$ to obtain the *discrete time Riccati recursion*

$$\mathbf{L}_k = M_k'M_k + A_k'\mathbf{L}_{k+1}A_k - A_k'\mathbf{C}_{k+1}B_k(N_k'N_k + B_k'\mathbf{L}_{k+1}B_k)^{-1}B_k'\mathbf{L}_{k+1}A_k \tag{12.30}$$

which produces a positive definite result $\mathbf{L}_k$ (as can be proven easily). Computationally, one finds the feedback matrix
$F_k = (N_k'N_k + B_k'\mathbf{L}_{k+1}B_k)^{-1}B_k'\mathbf{L}_{k+1}A_k$, first and then performs the Riccati update, thereby leaving $C_{o,k}$ and $D_{o,k}$ implicit.

However, this way of doing things is not advisable for two reasons: (1) the square root recursion produces $\mathbf{L}_k = Y_{k-1}'Y_{k-1}$ in square root form using a single QL factorization, thereby avoiding needless products, an inversion and an awkward subtraction and (2) loss of numerical accuracy, the conditioning of $\mathbf{L}_k$ being the square of the conditioning of Y_{k-1} (the conditioning says how sensitive the result is to inaccuracies in the data—an important issue as much of the data used in system theory is often approximate.)

12.6 Discussion items

1. Just as with the Kalman filter, what could be called the 'Bellman' filter is the result of a simple inner-outer factorization (in the Kalman case: an outer-inner factorization). So: formally, one can be considered the dual of the other. Or, more precisely: the Bellman filter is a Kalman filter on the dual system. This should be more than a pure technicality: both filters solve the same generic optimization problem but in a different context!

2. We have approached the optimization problem purely from a 'Moore-Penrose' approach (except in the section on dynamic programming,

where we briefly worked on a Lagrangian). A different approach could be called 'Hamiltonian', proposed by Bellman and extensively discussed in the classical book of Luenberger [27], and which has found wide acceptance in the control community, e.g., in the not less classical treatment of Anderson and Moore [2]. This approach is considerably more general than our Moore-Penrose shortcut, which can be considered a special case. Here is briefly how the Hamiltonian approach works, specialized to deterministic case (i.e., with all noise terms put to zero).

One considers the Lagrangian of eq. 12.2 in its unadorned form, and adds the system equations as constraints. This produces the augmented Lagrangian, with an added Lagrangian multiplier vector λ:

$$\mathcal{L}_a := \sum_{k=1}^{n+1} x_k' M_k' M_k x_k + \sum_{k=0}^{n} u_k' N_k' N_k u_k - \sum_{k=0}^{n} \lambda_k' \left(x_{k+1} - A_k x_k - B_k u_k\right) \tag{12.31}$$

Requesting the gradients to be zero (a necessary condition for a minimum along the constraint trajectory is that the gradient of the cost function be parallel to the gradient of the constraint) produces:

$$\begin{array}{lcl} \nabla_{x_{n+1}}\mathcal{L}_a & : & 2M_{n+1}' M_{n+1} x_{n+1} - \lambda_{n+1} = 0 \\ \nabla_{x_k|k=0\cdots n}\mathcal{L}_a & : & 2M_k' M_k x_k - \lambda_{k-1} + A_k' \lambda_k = 0 \\ \nabla_{u_k|k=0\cdots n}\mathcal{L}_a & : & 2N_k' N_k u_k + B_k' \lambda_k = 0 \\ \nabla_{\lambda_{k:k=0\cdots n}}\mathcal{L}_a & : & x_{k+1} - A_k x_k - B_k u_k = 0 \end{array} \tag{12.32}$$

i.e., a forward recursion for x_k with initial condition x_0 and a (coupled) backward recursion for λ_k with initial condition λ_{n+1} still to be determined (to meet the terminal condition given by the first equation in 12.32: $\lambda_{n+1} = 2M_{n+1}' M_{n+1} x_{n+1}$):

$$\lambda_{k-1} = A_k' \lambda_k + 2M_k' M_k x_k \tag{12.33}$$

This then leads to the 'Hamiltonian system of equations', often cited in the literature:

$$\begin{bmatrix} x_{k+1} \\ \lambda_{k-1} \end{bmatrix} = \begin{bmatrix} A_k & -B_k (2N_k' N_k)^{-1} B_k' \\ 2M_k' M_k & A_k' \end{bmatrix} \begin{bmatrix} x_k \\ \lambda_k \end{bmatrix} \tag{12.34}$$

This system cannot easily be solved as it is: it has partial boundary conditions at both sides. One way of dealing with it is dynamic programming: one starts from the far end and solves recursively for the 'optimal return cost functions', which one actually knows at the far end and which can then be determined recursively by a backward recursion. The optimal cost function at some intermediate point k is a function of the state x_k reached at that point and which is unknown until the initial state is reached. In the quadratic case the optimal cost function has a simple form: $x_k'\mathbf{L}_k x_k$, so that the recursion reduces to a recursion on $\mathbf{L}_k$, namely the Riccati recursion 12.30, which, in turn, reduces to the square-root recursion 12.13 and as a result $\mathbf{L}_{k+1} = Y_k{}'Y_k$. This makes the circle round, and one can surmise from our previous treatment (or prove directly) that

$$\lambda_{k+1} = 2Y_k{}'Y_k x_{k+1} = 2\mathbf{L}_{k+1}x_{k+1} \tag{12.35}$$

which basically propagates the cost function backwardly.

3. Our treatment has been a purely LTV treatment with a finite horizon. Other flavors, like LTI and an infinite horizon can pretty easily be accommodated using the same basic principles, be it with further assumptions. It is interesting to discuss various alternatives.

4. What about 'optimal time control' given constraints on the inputs and maybe on the states as well? Considering the noise-free case again, the simplest instance would likely be getting the state to zero with a prescribed energy budget. This can be done using the least squares theory of this chapter, with some adaptations. Let the energy necessary to bring an initial state x_0 to x_{n+1} be measured as $\|u_{0:n}\|_2^2$ (a different quadratic norm can easily be accommodated), then the minimal energy needed to bring a state x_0 in the pre-image of B_0 zero in one step is given by $\|B_0^\dagger A_0 x_0\|^2$. By the same reasoning, in two steps it would be $\| \begin{bmatrix} A_1B_0 & B_1 \end{bmatrix}^\dagger A_1A_0x_0\|^2$ for states that can be brought to zero in at most two steps, etc... with $\|u_{0:n}\|^2 = \| \begin{bmatrix} A_n\cdots A_1B_0 \mid \cdots \mid B_n \end{bmatrix}^\dagger A_n\cdots A_o x_0\|^2$ for states that can be brought to zero in at most n steps. A computation of $\mathcal{R}_{n+1}^\dagger := \begin{bmatrix} A_n\cdots A_1B_0 \mid \cdots \mid B_n \end{bmatrix}^\dagger$ is hence necessary. $\mathcal{R}_{n+1}$ is the reachability operator of a system with an $n+1$ horizon starting at 0. An interesting issue is how to deal with this case efficiently. $\{A, B\}$ can be brought to input normal form by a *backward* recursion, starting at position n and recursing down to position 0. This is reasonably efficient, but has to be redone when one

moves to position $n + 1$ etc... Is it possible to use results obtained at stage n to update to stage $n+1$? Once this is done, then one can make diagrams of minimal times needed to bring various states to zero given an overall energy budget for the inputs, an issue that is also very much worth exploring further.

It seems that we have reached the possibilities of quadratic optimization at this point. If one wants to go further, and e.g., investigate what minimal time control would be given a bound on the maximal *power* available, then the whole optimization process becomes non-quadratic and simple formulas for the return cost functions are lost. Nonetheless, the dynamic optimization principle remains valid and will still lead to adequate feedback laws, e.g., the famous 'bang-bang' control laws. For a nice exposition on these questions, see [13].

5. There are many examples of dynamic programming in different contexts. One very common one is the 'Viterbi algorithm' in decoding theory, namely to decode convolutional codes used in various coding schemas for telecommunications. Interestingly, the Viterbi algorithm uses a state space description to model the statistics (transition probabilities) of state propagation, rather than the propagation itself and searches for the most optimal path in a probabilistic sense.

12.7 Notes

1. The treatment of the optimal control problem as defined here was originally proposed by Bellman [7], who was inspired by the Lagrangian-Hamiltonian approach in modern mechanics. This seminal work gave a new boast to control theory, which so far was mostly based on input-output considerations, with root-locus techniques playing a major role. The big difference in the new approach is the use of state-space models and state feedback. A huge literature and many novel applications followed. For excellent if not classical introductions to the subject we already mentioned the books of Luenberger and Anderson-Moore. Our approach in this chapter is much more modest and aims only at showing (1) the connection with the basic numerical algebra methods we propose for dynamical system theory, at least in the more restricted quadratic setting and (2) that inner-outer factorization plays the central role.

Although it was clear from the start what the connection is between inner-outer factorization and quadratic optimization, this has not been the approach chosen in most of the literature, which has been heavily biased towards deriving and solving the resulting Riccati equation. Both from a theoretical and a numerical point of view this approach is questionable, because the move to the Riccati equation weakens the original form (with a disastrous loss of numerical accuracy) and thereby omits essential properties. This will become the more apparent when we treat more involved problems such as generalized Moore-Penrose inversion or spectral factorization in the next chapters.

2. Optimal control in the form presented in this chapter is not the only control issue, not even the only optimal control issue. In later chapters we shall consider other interesting forms of control such as H-infinity or robust control, where the cost to be optimized is not expressed in terms of state and input quantities, but in terms of operators mapping various inputs to various outputs. However, traditionally control was viewed mainly as a stabilization method, whereby output-input control is applied to stabilize the input-output transfer function of the system under consideration.

3. The discrete time, linear treatment given here is exemplary for the basic mechanisms occurring in an optimal control situation. However, the original treatments (like the one of Bellman, o.c.) were put in a continuous time, non-linear setting. This yielded a very powerful and general theory. One may wonder whether the inner-outer approach we followed can be extended to such settings, and that appears indeed to be the case, following the lead proposed by Ball and Helton, see [6]. We shall come back to this issue in a later chapter. Even further extensions are very much possible and do not seem to have been exploited in the literature, e.g., to discrete-event or so-called hybrid systems: these are systems that combine discrete-time components with continuous evolution in between. There are good arguments for the statement that, from a numerical point of view, all these more general situations can be reduced (with quite a bit of effort) to the inner-outer situation we have described in this chapter.

Chapter 13

Appendix 1: Applied Linear Algebra Summary

13.1 Preliminaries

In this section we review and give a summary of the algebraic concepts and notation used in this book. For more information, look up a basic textbook in linear algebra like [36] and one on numerical linear algebra like [15].

Logic and sets

All mathematics is based on an underlying understanding of logic and set theory, see e.g., [17]. The most elementary form of mathematical logic is proposition logic, which only handles a simplified algebra of "truth values" and is closely related to Boolean algebra. The extension of proposition logic to predicate logic provides the common framework in which most basic mathematics is formulated. We borrow from it its basic notation, and its basic modes of reasoning. Predicate logic extends proposition logic by the introduction of *quantors*, informally constructed as follows:

Quantor, Variable and Scope	:	$Property(Variable)$
$\forall$(x $\in$ human)	:	will die(x)
$\forall$(x $\in$ male)$\exists$(y $\in$ female)	:	married(x, y)
$\sum i = 1 : n$	:	u_i

The latter in short notation:

$$\textstyle\sum_{i=1:n} u_i$$

The first of these propositions is thought universally valid. The second is not true. The third is not a proposition. It would be one if one would write $\sum_{i=1:n} u_i = 10$, which, given values for the u_i could be true or not.

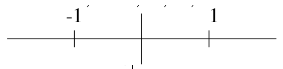
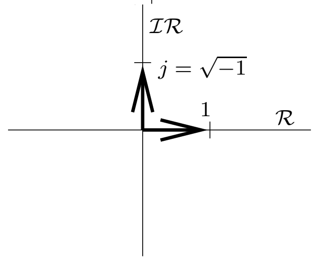
Quantors are not commutative, in particular: $\forall\exists\neq\exists\forall$—one must interpret $\forall x \exists y : \cdots$ as $\forall x\,(\exists y : \cdots)$[1]. Brackets must be used for nesting when the meaning is unclear. Most mathematical objects are described by a construct of predicate logic:

Sets: $\{x[\in \text{Scope}] : \text{logic specification of the set of x's}\}$.

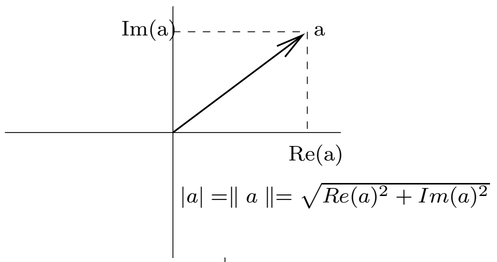
Example: $\{x \in \text{Integers} : \exists y \in \text{Integers}(x = 2y)\}$ specifies the set of even integers.

Numbers

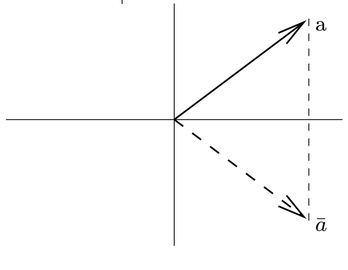
We will commonly use the following sets of numbers:

Integers	**Z**	$\cdots, -1, 0, 1, \cdots$
Reals	**R**	
Complex numbers	**C**	

for $a \in \mathbf{C}$:



conjugate:



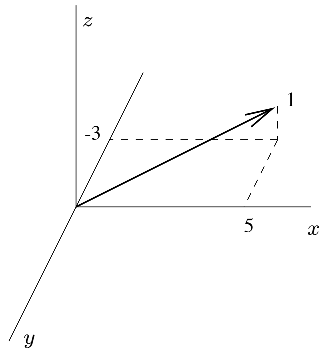
[1]Consider, e.g., $\forall x \exists y$ married$(x, y) = \exists y \forall x$ married(x, y)?

Vector Spaces

A vector space X $\boxed{\text{over}}$ $\mathbf{R}$ or $\boxed{\text{over}}$ $\mathbf{C}$ as "base spaces" is a set of elements called *vectors* on which an operation "addition" is defined with its normal properties (i.e., the inverse exists, a neutral element called zero (0) and the addition is commutative and associative—it forms a so called *abelian group*), and on which, in addition, *multiplication* with a scalar element of the base space is defined as well, with a slew of additional properties (such as $(x+y)a = xa + ya$, $xab = bax = (xa)b$ and $x0 = 0$, in which x, y are vectors and a, b numbers).

Concrete examples are common:

$\mathbf{R}^3$ $\quad \begin{bmatrix} 5 \\ -3 \\ 1 \end{bmatrix}$



$\mathbf{C}^3$ $\quad \begin{bmatrix} 5+j \\ -3-6j \\ 2+2j \end{bmatrix} \qquad \approx \mathbf{R}^6$

The addition for these is defined as:

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} x_1 + y_1 \\ x_2 + y_2 \\ \vdots \\ x_n + y_n \end{bmatrix}$$

and the scalar multiplication:
$a \in \mathbf{R}$ or $a \in \mathbf{C}$:

$$a \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} a = \begin{bmatrix} ax \\ ay \\ az \end{bmatrix}$$

(where only the middle term is a matrix multiplication—see further.)

Example

The most interesting case for our purposes is where a vector is actually a discrete time sequences $\{x(k) : k = 1 \cdots N\}$. The space that surrounds us and in which electromagnetic waves propagate is mostly linear (except when relativistic effects play a role). Signals reaching an antenna are added to each other.

Composition Rules:

Here is a summary of the most important consistency rules for vector spaces:

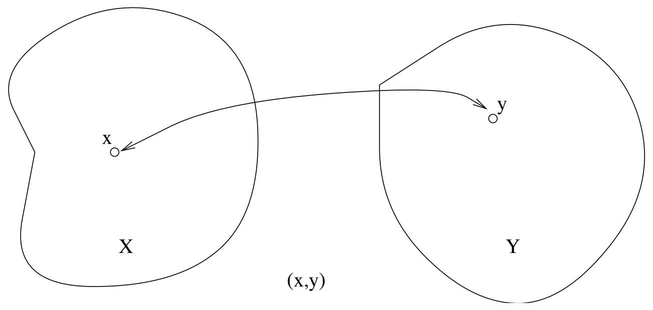
$$\begin{array}{lcll} x+y & = & y+x & \text{commutativity od addition} \\ (x+y)+z & = & x+(y+z) & \text{associativity of addition} \\ & 0 & & \text{neutral element} \\ x+(-x) & = & 0 & \text{inverse of addition} \\ (x+y)a & = & xa+ya & \text{distributivity of } * \text{ w.r. } + \end{array}$$

$$\left.\begin{array}{lcl} x*0 & = & 0 \\ x*1 & = & x \\ (x*a)*b & = & x*ab \end{array}\right\} \quad \text{consistencies}$$

Relations between sets

Let X and Y be sets, a relation R is a subset of the set of all ordered pairs:

$$R \subset X \times Y$$



It establishes a correspondence between some elements of the two sets. A relation has:

a domain: $\{x \in X : (\exists y \in Y : (x,y) \in R)\}$
a range: $\{y \in Y : (\exists x \in X : (x,y) \in R)\}$

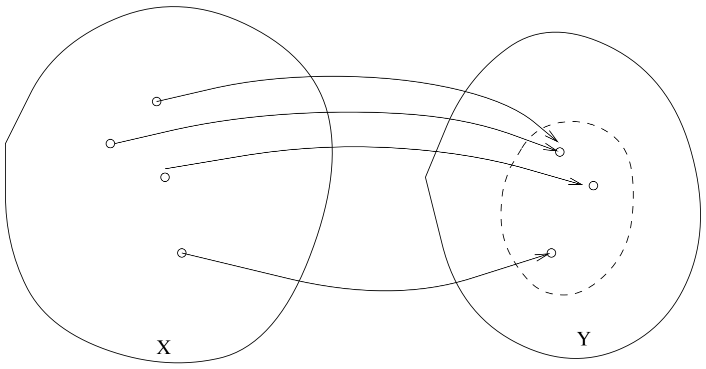
Important special attributes of a relation can be:

univocal:	$[(x, y_1) \in R]\&[(x, y_2) \in R] \Rightarrow [y_1 = y_2]$
[called a *partial function*]	
one to one (set isomorphism):	univocal & $[(x_1, y) \in R]\&[(x_2, y) \in R] \Rightarrow [x_1 = x_2]$

Functions or maps

A function (map) is a relation with the additional properties:

$$\begin{cases} \text{(1) its domain equals } X \\ \text{(2) it is univocal} \end{cases}$$



Important special attributes of a function can be:

$$\begin{cases} \text{one-to-one} \\ \text{onto} \\ \text{isomorfism = one-to-one and onto} \end{cases}$$

The set of all functions:

$$X \rightarrow Y$$

is a set of relations, hence a subset of the power set $\mathcal{P}(X \times Y)$, the set of all possible relations between X and Y.

Vector space of functions

Let X be a set and Y a vectorspace and consider the set of functions

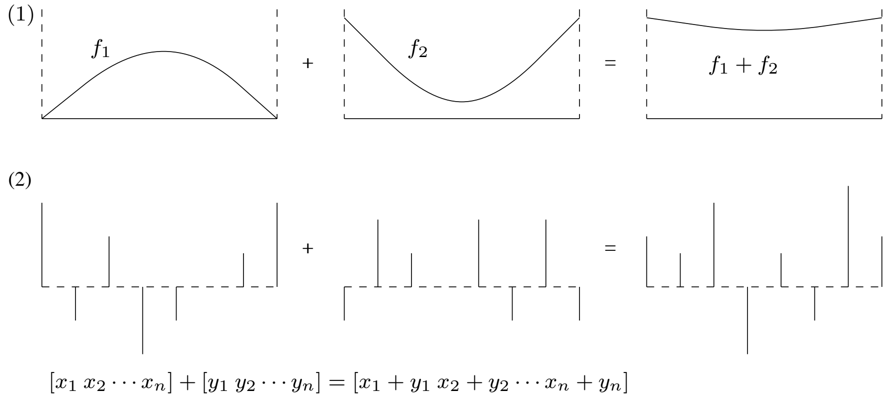
$$X \rightarrow Y.$$

We can define a new vector space on this set derived from the vector space structure of Y:

$$(f_1 + f_2)(x) = f_1(x) + f_2(x)$$

$$(af)(x) = af(x).$$

Examples:


(1)
f_1
+
f_2
=
$f_1 + f_2$
(2)
+
=
$[x_1\, x_2 \cdots x_n] + [y_1\, y_2 \cdots y_n] = [x_1 + y_1\, x_2 + y_2 \cdots x_n + y_n]$

As already mentioned, most vectors we consider can indeed be interpreted either as continous time or discrete time signals.

Linear maps

Assume now that both X **and** Y are vector spaces, then we can give a meaning to the notion 'linear map' as one that preserves the structure of vector space:

$$f(x_1 + x_2) = f(x_1) + f(x_2)$$

$$f(ax) = af(x)$$

we say that f defines a 'homomorfism of vector spaces'.

Bases

We say that a set of vectors $\{e_k\}$ in a vector space form a *basis*, if all the vectors in the space can be expressed as a unique linear combination of elements of the basis. It turns out that a basis always exists, and that all the bases of a given vector space have exactly the same number of elements. In $\mathbf{R}^n$ or $\mathbf{C}^n$ the *natural basis* is given by the elements

$$e_k = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

where the '1' is in the k^{th} position.

If

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

then

$$x = \sum_{k=1:n} x_k e_k$$

As further, related definitions and properties we mention the notion of *span* of a set $\{v_{1:k}\}$ of k vectors in a vector space V (denoted '$\bigvee v_{1:k}$'): it is the set of linear combinations $\{x : x = \sum_k \alpha_k v_k\}$ for some scalars $\{\alpha_k\}$—it is a subspace of V. We say that the set $\{v_k\}$ is linearly independent if it forms a basis for its span.

Matrices

A matrix (over the field $\mathbf{R}$ or $\mathbf{C}$) is 'a row vector of column vectors' (we use a MATLAB-like notation to indicate ranges of indices)

$$A = [a_{:,1} a_{:,2} \cdots a_{:,n}]$$

where

$$a_{:,k} = \begin{bmatrix} a_{1,k} \\ \vdots \\ a_{m,k} \end{bmatrix}.$$

and each $a_{i,k}$ is an element of the base field. We say that such a matrix has dimensions $m \times n$. (Dually the same matrix can be viewed as a column vector of row vectors.)

Given a $m \times n$ matrix A and an n-vector x, then we define the matrix vector-multiplication Ax as follows:

$$[a_{:,1} \cdots a_{:,n}] \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = a_{:,1}x_1 + \cdots + a_{:,n}x_n.$$

The vector x gives the composition recipe on the columns of A to produce the result. (A dual, equivalent definition says that a matrix is a column of rows etc...)

Matrix-matrix multiplication

can now be derived from the matrix-vector multiplication by stacking columns, in a fashion that is compatible with previous definitions:

$$[a_{:,1}\ a_{:,2} \cdots a_{:,n}] \begin{bmatrix} x_1 & y_1 & \cdots & z_1 \\ x_2 & y_2 & \cdots & z_2 \\ \vdots & \vdots & & \vdots \\ x_n & y_n & \cdots & z_n \end{bmatrix}$$

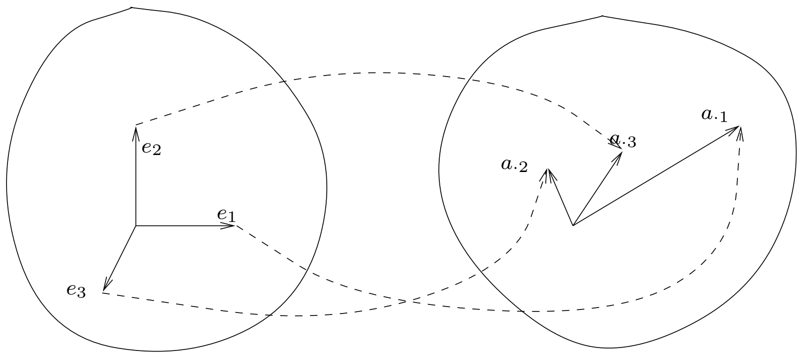
$$= \begin{bmatrix} Ax & Ay & \cdots & Az \end{bmatrix}$$

where each column is manufactured according to the matrix-vector multiplication recipe.

The dual viewpoint works equally well: one then defines row recipes forced by the right hand side on the left. Remarkably, the result is numerically the same! The product AB can be viewed as "column recipes given by the columns of B" acting on the columns of A, or, alternatively, "row recipes given by the rows of A" acting on the rows of B. Although the order of operations is different, the results are the same provided multiplication and addition are *associative* and multiplication is distributive w.r. to addition, they do not need to be commutative (hence the rules can be extended to block matrices—see further).

Linear maps represented as matrices

Maps $\mathbf{C}^n \to \mathbf{C}^m$ are represented by matrix-vector multiplications:



The way it works: map each natural basis vector $e_k \in \mathbf{C}^n$ to a column $a_{:,k} \in \mathbf{C}^m$. The matrix A build from these columns will map a general $x \in \mathbf{C}^n$ to Ax, where $A = [a_{:,1} \cdots a_{:,n}]$.

The procedure works equally well with more abstract spaces. Suppose that X and Y are such and a is a linear map between them (notation convention: we use caps (A, B, H, $\cdots$) for matrices representing linear maps, and lower case (a, b, h, $\cdots$) for (mostly) the corresponding 'abstract' operators i.e., the operator independently of the choice of basis in the domain and image spaces.). Choose bases in each space, then each vector can be represented by a 'concrete' vector of coefficients for the given basis, and we are back to the previous case. In particular, after the choice of bases, a will be represented by a 'concrete' matrix A mapping coefficients to coefficients.

This mechanism can easily be represented using formal matrix calculus, as follows. Let a basis in the domain be $\{e_k\}_{k=1:n}$, then we may assemble these abstract basis vectors (which all have the same dimension) into a vector $e := \left[\begin{array}{ccc} e_1 & \cdots & e_n \end{array}\right]$, and suppose that we have also chosen basis vectors $\{f_k\}_{k=1:m}$, collected into a matrix $f := \left[\begin{array}{ccc} f_1 & \cdots & f_m \end{array}\right]$, then we can write

$$ae = \left[\begin{array}{ccc} ae_1 & \cdots & ae_n \end{array}\right] = f \left[\begin{array}{ccc} a_{:,1} & \cdots & a_{:,n} \end{array}\right] \tag{13.1}$$

thereby defining a "concrete" matrix A representing a. Given a vector $\xi = ex$ with the numerical representation x in the e-basis, $a\xi = aex = fAx = fy$ for a concrete representation y of the image ηy with $y = Ax$—using the property that f is a basis of the image space

Operations on matrices

Important operations on matrices are:

Transpose:	$[A']_{i,j} = A_{j,i}$	(real space)
Hermitian conjugate:	$[A']_{i,j} = \bar{A}_{j,i}$	(complex space)
Addition:	$[A+B]_{i,j} = A_{i,j} + B_{i,j}$	
Scalar multiplication:	$[aA]_{i,j} = aA_{i,j}$	
Matrix multiplication:	$[AB]_{i,j} = \sum_k A_{i,k}B_{k,j}$	

Special matrices

We distinguish the following special matrices:

Zero matrix:	$0_{m\times n}$	shorthand: 0
Unit matrix:	I_n	shorthand: I

Working on blocks

Up to now we restricted the elements of matrices to scalars. The matrix calculus works equally well on more general elements, provided addition and multiplication make sense, are associative, and multiplication is distributive w.r. to addition, e.g., with block matrices provided dimensions match (but other cases of multiplication can work equally well).

Operators

Maps which correspond to square matrices, e.g. a map between $\mathbf{C}^n$ and itself or between an 'abstract' space X and itself represented by an $n \times n$ matrix, are often called *operators* (although the notion even extends to more general 'algebras' of maps, which can be combined with each other by addition and concatenation):

$$A \in \mathbf{C}^n \to \mathbf{C}^n.$$

An interesting case is a basis transformation:

$$[e_1\, e_2 \cdots e_n] \mapsto [f_1\, f_2 \cdots f_n]$$

such that $f_k = e_1 s_{1k} + e_2 s_{2k} + \cdots e_n s_{nk}$ produces a matrix S for which holds (using 'formal' multiplication):

$$[f_1 \cdots f_n] = [e_1 \cdots e_n]S$$

If this is a genuine basis transformation, then there must exist an inverse matrix S^{-1} s.t.

$$[e_1 \cdots e_n] = [f_1 \cdots f_n]S^{-1}$$

Basis transformation of a matrix representation

Suppose that a is an abstract operator, and $\eta = a\xi$, while for a concrete representation in a given basis $[e_1 \cdots e_n]$ we have:

$$\begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} = A \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$

(abreviated as $y = Ax$) with

$$\xi = [e_1 \cdots e_n] \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}, \eta = [e_1 \cdots e_n] \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$$

then in the new basis:

$$\xi = [f_1 \cdots f_n] \begin{bmatrix} \widehat{x}_1 \\ \vdots \\ \widehat{x}_n \end{bmatrix}, \eta = [f_1 \cdots f_n] \begin{bmatrix} \widehat{y}_1 \\ \vdots \\ \widehat{y}_n \end{bmatrix}$$

and consequently

$$\begin{bmatrix} \widehat{x}_1 \\ \vdots \\ \widehat{x}_n \end{bmatrix} = S^{-1} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$

$$\begin{bmatrix} \widehat{y}_1 \\ \vdots \\ \widehat{y}_n \end{bmatrix} = S^{-1} \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$$

$$\widehat{y} = S^{-1}y = S^{-1}Ax = S^{-1}AS\widehat{x} = \widehat{A}\widehat{x}$$

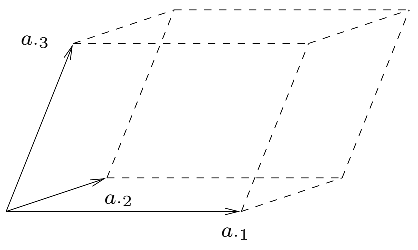
with

$$\widehat{A} = S^{-1}AS$$

by definition a *similarity transformation.*

Determinant of a square matrix

The determinant of a real $n \times n$ square matrix is *the signed volume of the n-dimensional parallellipeped which has as its edges the columns of the matrix.* (One has to be a little careful with the definition of the sign, for complex matrices one must use an extension of the definition—we skip these details).

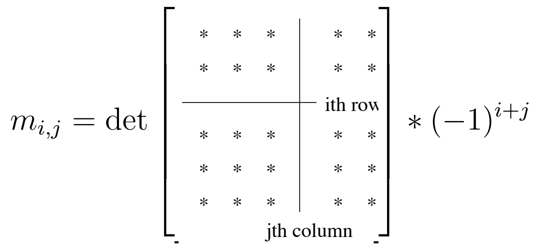


The determinant of a matrix has interesting properties:

- $\det A \in \mathbf{R}$ (or $\mathbf{C}$)
- $\det(S^{-1}AS) = \det A$
- $\det \begin{bmatrix} a_{1,1} & * & \cdots & * \\ 0 & a_{2,2} & \cdots & * \\ & & \ddots & * \\ 0 & & & a_{n,n} \end{bmatrix} = \prod_{i=1}^{n} a_{i,i}$
- $\det[a_{:,1} \cdots a_{:,i} \cdots a_{:,k} \cdots a_{:,n}] = -\det[a_{:,1} \cdots a_{:,k} \cdots a_{:,i} \cdots a_{:,n}]$
- $\det AB = \det A \cdot \det B$
- The matrix A is invertible iff $\det A \neq 0$. We call such a matrix *non-singular*.

Minors of a square matrix and minor matrix M

For each entry i, j of a square matrix A with i and $j \in 1 : n$ there is a minor $m_{i,j}$:



(i.e., cross out i$^{\text{th}}$ row and j$^{\text{th}}$ column, multiply with sign.).

This leads to the famous Cramer's rule for the inverse:
A^{-1} exists iff $\det A \neq 0$ and then:

$$A^{-1} = \frac{1}{\det A}[m_{j,i}]$$

(Note change of order of indices!)

Example:

$$\begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}^{-1} = \frac{1}{-2}\begin{bmatrix} 4 & -3 \\ -2 & 1 \end{bmatrix}$$

Another interesting property involving minors is the expansion of the determinant along a column (or, dually, a row):

$$\det A = \sum_{i=1}^{n} a_{i,j} m_{i,j}$$

(note that the result is independent of j!).

The characteristic polynomial of a matrix

Let: λ be a variable over $\mathbf{C}$ then the *characteristic polynomial* of a square matrix A is (for z a complex variable)

$$\chi_A(z) = \det(zI_n - A).$$

Example:

$$\begin{aligned} \chi_{\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}}(z) &= \det \begin{bmatrix} z-1 & -2 \\ -3 & z-4 \end{bmatrix} \\ &= (z-1)(z-4) - 6 \\ &= z^2 - 5z - 2 \end{aligned}$$

The characteristic polynomial is monic, the constant coefficient is $(-1)^n$ times the determinant, the coefficient of the n-1$^{\text{th}}$ power of z is minus the *trace* — $\text{trace}(A) = \sum_i a_{i,i}$, the sum of the diagonal entries of the matrix.

Sylvester identity:
The matrix A satisfies the following remarkable identity:

$$\chi_A(A) = 0$$

i.e. A^n depends linearly on $I, A, A^2, \cdots, A^{n-1}$ (see the section on Schur similarity for the indication of a proof).

Matrices and composition of functions

Let:

$$f : X \to Y,\ g : Y \to Z$$

then:

$$g \circ f : X \to Z : (g \circ f)(x) = g(f(x)).$$

As we already know, linear maps f and g are represented by matrices F and G after a choice of a basis. The representation of the composition becomes matrix multiplication:

$$(g \circ f)(x) = GFx.$$

Norms on vectorspaces

Let X be a linear space. A norm $\|\cdot\|$ on X is a map $\|\cdot\| : X \to \mathbf{R}^+$ which satisfies the following properties:

a. $\|x\| \geq 0$

b. $\|x\| = 0 \Leftrightarrow x = 0$

c. $\|ax\| = |a| \cdot \|x\|$

d. $\|x - z\| \leq \|x - y\| + \|y - z\|$
(triangle inequality)

The purpose of the norm is to measure the 'size' or 'length' of a vector according to some measuring rule. There are many norms possible, e.g. in $\mathbf{C}^n$: The '1' norm:

$$\|x\|_1 = \sum_{i=1}^{n} |x_i|$$

The quadratic norm:

$$\|x\|_2 = \left[\sum_{i=1}^{n} |x_i|^2\right]^{\frac{1}{2}}$$

The 'sup' norm:

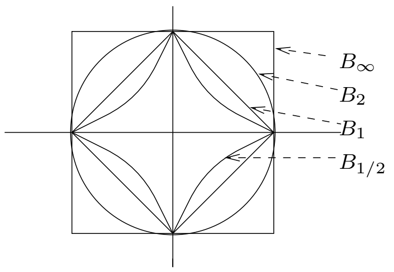
$$\|x\|_\infty = \sup_{i=1\cdots n} (|x_i|)$$

Not a norm is:

$$\|x\|_{\frac{1}{2}} = \left[\sum_{i=1}^{n} |x_i|^{\frac{1}{2}}\right]^2$$

(it does not satisfy the triangle inequality), but it can be used to measure sizes as well—it, and its siblings of the form $\|x\|_k$ with $0 < k < 1$ have an important role to play in signal processing: they strongly favor the directions of the natural basis.

Unit ball in the different norms: shown are the respective 'unit balls' $\{x : \|x\|_? = 1\}$.



An interesting question is: which norm is the strongest? (A norm that constraints more is stronger! In the definition here we have $\|x\|_\infty \leq \|x\|_2 \leq \|x\|_1$, hence $\|x\|_1$ constraints the most: it may still be pretty large when the others are already small!))

The general p-norm has the form:

$$\|x\|_p = \left[\sum |x_i|^p\right]^{\frac{1}{p}} \; (p \geq 1)$$

P-norms satisfy the following important "Hölder inequality":
Let $p \geq 1$, $q = p/(p-1)$, then

$$\left| \sum_{i=1}^{n} x_i y_i \right| \leq \|x\|_p \|y\|_q$$

Inner products

Inner products put even more structure on a vector space than norms and allow us to deal with 'orthogonality' or even more general angles!

Let: X be a vector space (over $\mathbf{C}$).

An inner product is a map $X \times X \to \mathbf{C}$ such that:

a. $(y, x) = \overline{(x, y)}$

b. $(ax + by, z) = a(x, z) + b(y, z)$

c. $(x, x) \geq 0$

d. $(x, x) = 0 \Leftrightarrow x = 0$

Hence: $\|x\| = (x, x)^{\frac{1}{2}}$ is the *Euclidean* norm $\|x\|_2$.

Question: when is a normed space also an inner product space compatible with the norm?

The answer is known: e.g., in the real case, when the parallelogram rule is satisfied:

$$\|x + y\|^2 + \|x - y\|^2 = 2\left(\|x\|^2 + \parallel y \parallel^2\right)$$

(Exercise: define the appropriate inner product in term of the norm!)

The *natural inner product* on $\mathbf{C}^n$ is given by:

$$(x, y) = \sum_{i=1}^{n} x_i \bar{y}_i = y'x = [\bar{y}_1 \cdots \bar{y}_n] \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

The *gramian* of a basis is defined as

Definition 9 *Let:* $\{f_i\}_{i=1\cdots n}$ *be a basis for* $\mathbf{C}^n$*, then its Gramian* G *is given by:*

$$\begin{aligned} G &= [(f_j, f_i)] \\ &= \begin{bmatrix} f_1' \\ \vdots \\ f_n' \end{bmatrix} [f_1 \cdots f_n] \end{aligned}$$

A basis is orthonormal when its Gramian is a unit matrix:

$$G = I_n$$

Hermitian matrix: a matrix A is symmetric (in the real case) or *hermitian* (in the complex case) iff

$$A = A'$$

i.e., in all cases: $[A']_{i,j} = \bar{A}_{j,i}$—this becomes, in the block case, $[A']_{i,j} = A'_{j,i}$

Definite matrices

Definitions:

Definition 10 *Let* $A \in \mathbf{C}^{n\times n}$ *be hermitian (or, in the real case, symmetric) then, given any inner product,* A *is positive (semi)definite if*

$$\forall x : (Ax, x) \geq 0$$

A *is strictly positive definite if*

$$\forall x \neq 0 : (Ax, x) > 0$$

(the property is independent from the inner product chosen.)

The gramian of a basis is always strictly positive definite!

Operator norms

Let X and Y be normed spaces, and

$$f : X \to Y$$

a linear map, then

$$\|f\| = \sup_{\|x\| \neq 0} \frac{\|f(x)\|_Y}{\|x\|_X} = \sup_{\|x\|=1} \|f(x)\|$$

is a valid norm on the space $X \to Y$. It measures the longest elongation of any vector on the unit ball of X under f.

Operators on an inproduct space

Let

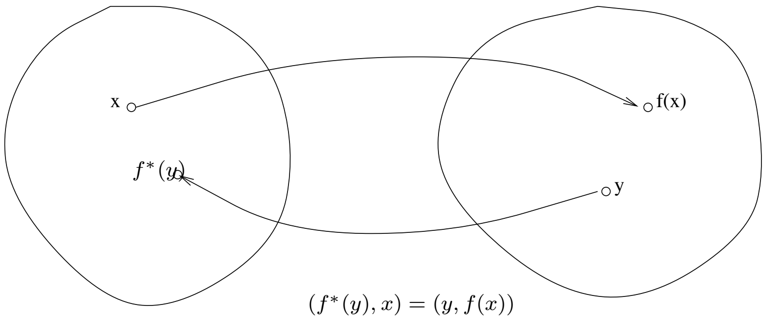
$$f : X \to Y$$

where X and Y have specific inproducts $(\cdot,\cdot)_X$ and $(\cdot,\cdot)_Y$.

The adjoint map f^* is defined as:

$$f^* : Y \to X : \forall x \forall y [(f^*(y), x)_X = (y, f(x))_Y]$$

(note that it depends on the specific inproducts!)



$(f^*(y), x) = (y, f(x))$

On matrices which represent an operator in a natural basis there is a simple expression for the adjoint: if f is $y = Ax$, and $(y, f(x)) = y'Ax$, then $y'Ax = (A'y)'x$ so that $f^*(y) = A'y$. (This also shows quite simply that the adjoint always exist and is unique, given the inproducts).

The adjoint map is very much like the original, it is in a sense its 'complex conjugate', and the composition $f^* \circ f$ is a "square" or a "covariance", depending on the context.

We say that a map is *self-adjoint* if $X = Y$ and $f = f^*$, and that it is *isometric* if

$$\forall x : \|f(x)\| = \|x\|$$

in that case:

$$f^* \circ f = I_X$$

It is said to be *co-isometric* iff its adjoint is isometric.

Unitary (Orthogonal) maps

A linear map f is *unitary* (orthogonal in the real case) if both f and f^* are isometric:

$$f^* \circ f = I_X$$

and

$$f \circ f^* = I_Y$$

In that case X and Y must have the same dimension, they are isomorphic: $X \approx Y$.

Example:

- $A = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$ is isometric with adjoint (in the natural inner product)

$$A' = [\frac{1}{\sqrt{2}}\,\frac{1}{\sqrt{2}}]$$

- The adjoint of

$$A = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}$$

  in the natural inner product, is

$$A' = \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}$$

  Both these maps are isometric and hence A is unitary, it rotates a vector over an angle of $-45°$, while A' is a rotation over $+45°$.

Norms for matrices

We have seen that the measurement of lengths of vectors can be bootstrapped to maps and hence to matrices. Let A be a matrix.

Definition: the *operator norm* or *euclidean norm* for A is (consistent with the previous definition):

$$\|A\|_E = \sup_{x \neq 0} \frac{\|Ax\|_2}{\|x\|_2}$$

It measures the greatest relative elongation of a vector x subjected to the action of A (in the natural basis and using the quadratic norm).

Properties:

- $\|A\|_E = \sup_{\|x\|=1} \|Ax\|_2$
- A is isometric if $A'A = I$, then $\|A\|_E = 1$, (the converse is *not* true!).
- Product rule: $\|FG\|_E \leq \|F\|_E \|G\|_E$.

Often, when the matrix norm is written without an index (as $\|A\|$), the euclidean norm is meant—the context should make that clear.) Note that the euclidean norm is *not* a quadratic norm (in contrast to the Frobenius norm that we define below): it does not satisfy the parallelogram rule.

Contractive matrices: A is contractive iff $\|A\|_E \leq 1$.

Positive real matrices: A is positive real if it is square and if

$$\forall x : ((A + A')x, x) \geq 0.$$

This property is abbreviated to: $A + A' \geq 0$. We say that a matrix is strictly positive real if $x'(A + A')x > 0$ for all $x \neq 0$.

If A is contractive, then $I - A'A \geq 0$.

Cayley transform: if A is positive real, then $S = (A - I)(A + I)^{-1}$ is contractive. Conversely, if $(I - S)$ is invertible and S is contractive, then $A := (I + S)(I - S)^{-1}$ is positive real.

Frobenius norm

The *Frobenius norm* is the quadratic norm of a matrix viewed as a vector, after rows or columns have been stacked into a single vector:

$$\|A\|_F = \left[\sum_{i,j=1}^{n,m} |a_{i,j}|^2 \right]^{\frac{1}{2}}$$

Properties:

- $\|A\|_F^2 = \operatorname{trace} A'A = \operatorname{trace} AA'$
- $\|A\|_E \leq \|A\|_F$, the Frobenius norm is “stronger” than the euclidean—when the Frobenius norm is small, then the euclidean norm will a fortiori be even smaller (i.e., small Frobenius $\Longrightarrow$ small euclidean).

Kernels and ranges

Definition 11 *Let A be a matrix, interpreted as a linear map $X \to Y$, then*

- *kernel of A: $\mathcal{K}(A) = \{x : Ax = 0\} \subset X$*
- *range of A: $\mathcal{R}(A) = \{y : (\exists x \in X : y = Ax)\} \subset Y$*
- *co-kernel of A = kernel of A': $\{y : A'y = 0\} \subset Y$*
- *co-range of A = range of A': $\{x : (\exists y : x = A'y)\} \subset X$*

(similar definitions hold for an abstract operator, with "adjoint" instead of "conjugate".).

Orthogonality

All vectors and subspaces considered in this subsection live in a large inner-product (Euclidean) space.

- vectors: $x \perp y \Leftrightarrow (x, y) = 0$
- spaces: $X \perp Y \Leftrightarrow (\forall x \in X)(\forall y \in Y) : (x, y) = 0$
- direct sum: $Z = X \oplus Y \Leftrightarrow (X \perp Y)\&(X, Y \text{ span } Z)$, i.e., $(\forall z \in Z)(\exists x \in X)(\exists y \in Y) : z = x+y$ (in fact, x and y are unique projections of z on respect. X and Y).

Example w.r. kernels and ranges of a map $A : X \to Y$:

$$X = \mathcal{K}(A) \oplus \mathcal{R}(A')$$

$$Y = \mathcal{K}(A') \oplus \mathcal{R}(A)$$

Projections

Let X be an euclidean space, with natural inner product.

- $P : X \to Y$ is a projection if $P^2 = P$.
- a projection P is an orthogonal projection if in addition:

$$\forall x \in X : Px \perp (I - P)x$$

- Property: P is an orthogonal projection if (1) $P^2 = P$ and (2) $P = P'$.

Application: projection on the column range of a matrix.
Let

$$A = [a_{:,1}\, a_{:,2}\, \cdots a_{:,m}]$$

such that the columns are linearly independent. Then $A'A$ is non-singular and

$$P = A(A'A)^{-1}A'$$

is the orthogonal projection on the column range of A.

Proof (sketch):

- check: $P^2 = P$
- check: $P' = P$
- check: P projects each column of A onto itself.

Eigenvalues, eigenspaces

Let A be a square $n \times n$ matrix. Then $\lambda \in \mathbf{C}$ is an eigenvalue of A and x an eigenvector, if

$$Ax = \lambda x.$$

The eigenvalues are the roots of the characteristic polynomial $\det(zI - A)$.

Schur's similarity triangularization: for any $n \times n$ square matrix A there exists an uppertriangular matrix

$$S = \begin{bmatrix} s_{11} & \cdots & s_{1n} \\ & \ddots & \vdots \\ 0 & & s_{nn} \end{bmatrix}$$

and a unitary matrix U such that

$$A = USU'.$$

The diagonal entries of S are the eigenvalues of A (including multiplicities). Schur's eigenvalue theorem is easy to prove by recursive computation of a single eigenvalue and deflation of the space. Schur's theorem can be used to show that $\chi_A(A) = 0$ where $\chi_A(z)$ is the characteristic polynomial of A.

A bit more difficult is Jordan's theorem.

Jordan's theorem

Jordan blocks are square matrices of the form

$$\Lambda = \begin{bmatrix} \lambda & 1 & & 0 \\ & \lambda & \ddots & \\ & & \ddots & 1 \\ 0 & & & \lambda \end{bmatrix}.$$

λ is the eigenvalue, if the block has dimension δ then λ has multiplicity δ but *defect* $\delta - 1$ since the block has only one eigenvector.

Jordan's theorem: A is similar to a block-diagonal matrix of Jordan blocks, i.e. there exists an invertible matrix T such that

$$A = T \begin{bmatrix} \Lambda_1 & & & 0 \\ & \Lambda_2 & & \\ & & \ddots & \\ 0 & & & \Lambda_k \end{bmatrix} T^{-1}$$

where

$$\Lambda_i = \begin{bmatrix} \lambda_i & 1 & & 0 \\ & \lambda_i & \ddots & \\ & & \ddots & 1 \\ 0 & & & \lambda_i \end{bmatrix}.$$

Because of its importance for dynamical systems, we give an indication of how a proof is constructed. First, one defines what is a called an *extended eigenspace* attached to a given eigenvalue λ. When λ is indeed an eigenvalue, then $\ker(A-\lambda)$ is not empty. Next, it may be that $\ker(A-\lambda)^2$ (which contains $\ker(A-\lambda)$) is strictly larger than $\ker(A-\lambda)$. Continuing in this fashion, one finds a smallest integer $r \geq 1$ for which $\ker(A-\lambda)^r = \ker(A-\lambda)^{(r+1)}$. One can then show that $\mathcal{K} := \ker(A-\lambda)^r = \ker(A-\lambda)^s$ for all integers $s \geq r$. $\mathcal{K}$ is by definition the *extended eigenspace* belonging to λ.

Next, let $\mathcal{R} = \operatorname{ran}(A-\lambda)^r$, and let $\mathcal{U}$ be the full space on which A is defined. One shows: (1) $\mathcal{U} := \mathcal{K} \oplus \mathcal{R}$; (2) $A\mathcal{K} \subset \mathcal{K}$ (i.e. $\mathcal{K}$ is an *invariant subspace* for A); (3) $A\mathcal{R} \subset \mathcal{R}$ (or $\mathcal{R}$ is also an invariant subspace for A); (4) λ is the sole eigenvalue of $A|_{\mathcal{K}}$; (5) λ is not an eigenvalue of $A|_{\mathcal{R}}$. The consequence of all this is that A can be transformed by similarity to a block decomposition $\begin{bmatrix} \Lambda & \\ & A_1 \end{bmatrix}$, whereby Λ has λ as sole eigenvalue and λ is not an eigenvalue of the reduced block A_1.

The next step is then to determine a structure for Λ. This is basically achieved by determining what is known as *Jordan chains*. Let $\mathcal{R}$ now be the range of $(\Lambda - \lambda)^{(r-1)} \neq 0$, with $(\Lambda - \lambda)^r = 0$, and let $t_1 \neq 0$ be a vector in $\mathcal{R}$. Then $(\Lambda - \lambda)t_1 = 0$, and t_1 is an eigenvector of Λ. Next, presuming $r > 1$, and because of the range assumption for t_1, one finds a vector t_2 such that $(\Lambda - \lambda)t_2 = t_1$, and then, presuming $r > 3$, t_3 such that $(\Lambda - \lambda)t_3 = t_2$, etc... until $t_r \neq 0$ is reached, for which $(\Lambda - \lambda)t_r = t_{r-1}$ and the chain has to end because otherwise one would have $t_1 = (\Lambda - \lambda)^r t_{r+1} = 0$, contrary to the original assumption on t_1. The vectors $t_1, t_2, \cdots, t_r$ form by definition a Jordan chain, and, because of the construction, we have the relationship

$$\Lambda \begin{bmatrix} t_1 & t_2 & \cdots & t_{r-1} & t_r \end{bmatrix} = \begin{bmatrix} t_1 & t_2 & \cdots & t_{r-1} & t_r \end{bmatrix} \begin{bmatrix} \lambda & 1 & & & \\ & \lambda & \ddots & & \\ & & \ddots & \ddots & \\ & & & \lambda & 1 \\ & & & & \lambda \end{bmatrix} \tag{13.2}$$

in which one sees a first Jordan block appearing. It is not hard to show that the just defined t_k are linearly independent, and that their span is an invariant subspace for Λ. Moreover, only t_1 can be a vector in $\mathcal{R}$. If $\dim \mathcal{R} > 1$, then another non-zero vector in $\mathcal{R}$ can be found, independent from t_1, and the construction of a Jordan chain of dimension r can be repeated. This can be done until exhaustion of the dimension of $\mathcal{R}$, and one shows that all the vectors so obtained are linearly independent (tedious, but straightforward proof.) This may not yet exhaust all possibilities. We certainly have $\text{ran}(\Lambda - \lambda)^{(r-1)} \subset \text{ran}(\Lambda - \lambda)^{(r-2)}$ (assuming $r > 1$), so it may be that the collection of vectors of type t_2 obtained in the previously constructed Jordan chains do not span the whole space $\text{ran}(\Lambda - \lambda)^{(r-2)}$, and one can find more vectors to generate (smaller) Jordan chains, and again invariant subspaces spanned by vectors that are independent of all the previous ones. This process can be continued further, gradually decreasing r, and necessarily ends after all possibilities have been exhausted, and the complete space of definition of Λ is spanned by all the (generalized eigen-)vectors determined. Although the full proof is quite technical (as one can imagine at this point), the construction of Jordan chains has independent interest and good use in the theory of stability of ordinary differential equations and the solution of Lyapunov-type equations.

Ill conditioning of multiple or clusters of eigenvalues

Look e.g. at a 'companion matrix':

$$A = \begin{bmatrix} 0 & & & -p_0 \\ 1 & \ddots & & \vdots \\ & \ddots & 0 & \vdots \\ 0 & & 1 & -p_{n-1} \end{bmatrix}$$

its characteristic polynomial is:

$$\chi_A(z) = z^n + p_{n-1}z^{n-1} + \cdots + p_0.$$

Assume now that $p(z) = (z-a)^n$ and assume a perturbation $p_\epsilon(z) = (z-a)^n - \epsilon$. The new roots of the polonymial and hence the new eigenvalues of A are:

$$a + \epsilon^{\frac{1}{n}} e^{j\pi k/n}$$

Hence: an ϵ error in the data produces an error of size $\epsilon^{\frac{1}{n}}$ in the result (take e.g. $n = 10$ and $\epsilon = 10^{-5}$, then the error is ≈ 1!)

13.2 Systems of Equations, QR algorithm

Let be given: an $n \times m$ matrix T and an n-vector b.
Asked: an m-vector x such that:

$$Tx = b$$

Distinguish the following cases: $\begin{cases} n > m & \text{more equations than unknowns} \\ n = m & \text{square} \\ n < m & \text{less equations than unknowns} \end{cases}$

Although all these cases are important, we look first at the cases $n \geq m$.

The general strategy for solution is to use orthogonal transformations on rows, because these are numerically desirable, and choose them in such a way that the system becomes simpler. Let a and b be two rows in a matrix, then we can generate linear combinations of these rows by applying a rotation matrix to the left (row recipe):

$$\begin{bmatrix} t_{11} & t_{12} \\ t_{21} & t_{22} \end{bmatrix} \begin{bmatrix} \longleftarrow & a & \longrightarrow \\ \longleftarrow & b & \longrightarrow \end{bmatrix} = \begin{bmatrix} t_{11}a + t_{12}b \\ t_{21}a + t_{22}b \end{bmatrix}$$

or embedded:

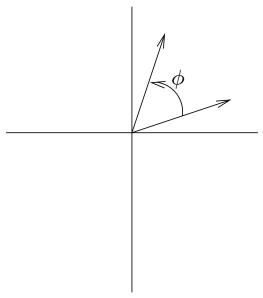
$$\begin{bmatrix} 1 & & & & & \\ & 1 & & & & \\ & & t_{11} & & t_{12} & \\ & & & 1 & & \\ & & t_{21} & & t_{22} & \\ & & & & & 1 \end{bmatrix} \begin{bmatrix} \longleftarrow & \cdot & \longrightarrow \\ \longleftarrow & \cdot & \longrightarrow \\ \longleftarrow & a & \longrightarrow \\ \longleftarrow & \cdot & \longrightarrow \\ \longleftarrow & b & \longrightarrow \\ \longleftarrow & \cdot & \longrightarrow \end{bmatrix} = \begin{bmatrix} \longleftarrow & \cdot & \longrightarrow \\ \longleftarrow & \cdot & \longrightarrow \\ & t_{11}a + t_{12}b & \\ \longleftarrow & \cdot & \longrightarrow \\ & t_{21}a + t_{22}b & \\ \longleftarrow & \cdot & \longrightarrow \end{bmatrix}$$

Jacobi transformations

Assuming real arithmetic, the Jacobi elementary transformation (often called Givens rotation) is:

$$\begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix}$$

It represents an elementary rotation of a two dimensional vector over an angle ϕ:



A Jacobi transformation can be used to annihilate an element in a row (with $c \doteq \cos\phi$ and $s \doteq \sin\phi$):

$$\begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{bmatrix} a_1 & a_2 & \cdots & a_m \\ b_1 & b_2 & \cdots & b_m \end{bmatrix} = \begin{bmatrix} ca_1 - sb_1 & ca_2 - sb_2 & \cdots \\ sa_1 + cb_1 & sa_2 + cb_2 & \cdots \end{bmatrix}$$

$$\doteq \begin{bmatrix} \sqrt{|a_1|^2 + |b_1|^2} & \star & \cdots & \star \\ 0 & \star & \cdots & \star \end{bmatrix}$$

when $sa_1 + cb_1 = 0$ or

$$\tan\phi = -\frac{b_1}{a_1}$$

which can always be done (of course, one does not compute the tangent, but rather put

$$\begin{bmatrix} c & -s \\ s & c \end{bmatrix} = \begin{bmatrix} \frac{a_1}{c} & \frac{b_1}{c} \\ -\frac{b_1}{c} & \frac{a_1}{c} \end{bmatrix} \tag{13.3}$$

with $c = \sqrt{a_1^2 + b_1^2}$!)

QR Factorization

A cascade of unitary transformations is still a unitary transformation: let Q_{ij} be a transformation on *rows* i and j which annihilates an appropriate element (as indicated further in the example). Successive eliminations on a 4×3 matrix (in which the $\cdot$ indicates a relevant element of the matrix, and $(\cdots)$ the previous result):

$$Q_{12}\begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} \mapsto \begin{bmatrix} \star & \star & \star \\ 0 & \star & \star \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}; Q_{13}(\cdots) \mapsto \begin{bmatrix} \star & \star & \star \\ 0 & \cdot & \cdot \\ 0 & \star & \star \\ \cdot & \cdot & \cdot \end{bmatrix}; Q_{14}(\cdots)$$

$$\mapsto \begin{bmatrix} \star & \star & \star \\ 0 & \cdot & \cdot \\ 0 & \cdot & \cdot \\ 0 & \star & \star \end{bmatrix}; Q_{23}(\cdots) \mapsto \begin{bmatrix} \cdot & \cdot & \cdot \\ 0 & \star & \star \\ 0 & 0 & \star \\ 0 & \cdot & \cdot \end{bmatrix}; Q_{24}(\cdots) \mapsto \begin{bmatrix} \cdot & \cdot & \cdot \\ 0 & \star & \star \\ 0 & 0 & \cdot \\ 0 & 0 & \star \end{bmatrix}; Q_{34}(\cdots)$$

$$\mapsto \begin{bmatrix} \cdot & \cdot & \cdot \\ 0 & \cdot & \cdot \\ 0 & 0 & \star \\ 0 & 0 & 0 \end{bmatrix}$$

(The elements that have changed during the last transformation are denoted by a '$\star$'. There are no fill-ins, a purposeful zero does not get modified in subsequent operations.) The end result is:

$$Q_{34}Q_{24}Q_{23}Q_{14}Q_{13}Q_{12}T = \begin{bmatrix} R \\ 0 \end{bmatrix}$$

in which R is upper triangular.

Solving the system $Tx = b$

Let also:

$$Q_{34} \cdots Q_{12} b \doteq \beta$$

then $Tx = b$ transforms to:

$$\begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} \\ 0 & r_{2,2} & r_{2,3} \\ 0 & 0 & r_{3,3} \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \end{bmatrix}$$

.

The solution, if it exists, can now easily be analyzed:

1. R is non-singular ($r_{1,1} \neq 0, \cdots, r_{m,m} \neq 0$), then the partial set

   $$Rx = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix}$$

   has a unique exact solution for any x obtained by backsubstitution:

   $$x = \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} \\ 0 & r_{2,2} & r_{2,3} \\ 0 & 0 & r_{3,3} \end{bmatrix}^{-1} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix} = \begin{bmatrix} \cdots \\ r_{2,2}^{-1}(\beta_{2,2} - r_{2,3} r_{3,3}^{-1} \beta_3) \\ r_{3,3}^{-1} \beta_3 \end{bmatrix}$$

   (there are better methods, see further!), and:
   1. if $\beta_4 \neq 0$ there is no exact solution for the original system,
   2. if $\beta_4 = 0$ we have found the unique solution.

2. when R is singular further analysis is necessary (one or more of the diagonal entries will be zero yielding more possibilities for contradictions—we skip this case temporarily).

Least squares solutions

But... there is more, even when $\beta_4 \neq 0$:

$$x = R^{-1} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix}$$

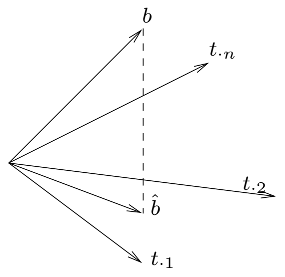
provides for a 'least squares fit' it gives the linear combination of columns of T closest to b i.e. it minimizes

$$\|Tx - b\|_2.$$

Geometric interpretation: let

$$T = [t_{:,1}\, t_{:,2} \cdots t_{:,m}]$$

then one may wonder whether b can be written as a linear combination of $t_{:,1}$ etc.?
Answer: only if $b \in \text{span}\{t_{:,1}, t_{:,2}, \cdots, t_{:,m}\}$!



Otherwise: find the "least squares fit", the combination of $t_{:,1}$ etc. which is closest to b, i.e. the orthogonal projection $\widehat{b}$ of b on the span of the columns.

Let us analyze these statements further: a QR-transformation rotates all the vectors $t_{:,i}$ *and* b over the same angles, with as result:

$$r_{:,1} \in \text{span}\{e_1\},$$

$$r_{:,2} \in \text{span}\{e_1, e_2\}$$

etc., leaving all angles and distances equal. We see that the projection of the vector β on the span of the columns of R is actually

$$\begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ 0 \end{bmatrix}$$

Hence the combination of columns that produces the least squares fit is:

$$x = R^{-1} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix}$$

Formal proof

Let Q be a unitary transformation such that

$$T = Q \begin{bmatrix} R \\ 0 \end{bmatrix}$$

in which R is upper triangular.

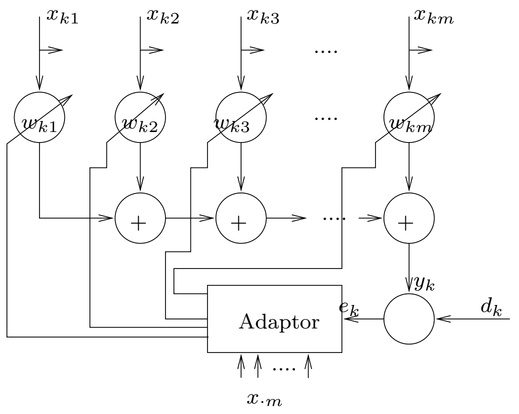
Then: $Q'Q = QQ' = 1$ and the approximation error becomes, with $Qb = \begin{bmatrix} \beta_1 \\ \beta_2 \end{bmatrix}$ conformal with the QR-factorization:

$$\|Tx - b\|_2^2 = \|Q'(Tx - b)\|_2^2 = \| \begin{bmatrix} R \\ 0 \end{bmatrix} x - \beta\|_2^2 = \|Rx - \beta_1\|_2^2 + \|\beta_2\|_2^2$$

If R is invertible a minimum is obtained for $Rx = \beta_1$ and the minimum error is $\|\beta_2\|_2$.

Application: adaptive QR

The classical adaptive filter:



At each time-index k, a data vector $x(k) = [x_{k,1} \cdots x_{k,m}]$ of dimension m comes in (e.g. from an antenna array or a delay line). We wish to estimate, at each point in time, a signal y_k as $\widehat{y}_k = \sum_i w_{k,i}x_{k,i}$ — a linear combination of the incoming data. Assume that we dispose of a "learning phase" in which the exact value d_k for y_k is known, so that the estimation error we are making $e_k = \widehat{y}_k - d_k$ is known also (typically the estimation error is due to inaccuracies and undesired signals that have been added in and which we call 'noise').

The problem is to find an "optimal" $w_{k,i}$, given the sequences received in the learning phase and the corresponding errors. We choose as optimality criterion: given the data from $t = 1$ to $t = k$, find the $w_{k,i}$ for which the total error is minimal when the new weights $w_{k,i}$ had indeed been used at all available time points $1 \leq i \leq k$ and this in least squares sense (many variations of the optimization strategy are possible).

For $i \leq k$, let $y_{k,i} = \sum_\ell x_{i\ell} w_{i\ell}$ be the output one would have obtained if $w_{k,i}$ had been used at that time and let

$$X_k = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\ \vdots & \vdots & \vdots & \vdots \\ x_{k,1} & x_{k,2} & \cdots & x_{k,m} \end{bmatrix}$$

be the 'data matrix' contained all the data collected in the period $1 \cdots k$. We wish a least squares solution of

$$X_k w_{k,:} - d_{1:k} = e_{1:k}$$

If

$$X_k = Q_k \begin{bmatrix} R_k \\ 0 \end{bmatrix},\ d_{1:k} = Q_k' \begin{bmatrix} \delta_{k,1} \\ \delta_{k,2} \end{bmatrix}$$

is a QR-factorization of X_k with conformal partitioning of d and R_k upper-triangular, and assuming R_k non-singular, we find as solution to our least squares problem:

$$w_{k\cdot} = R_k^{-1} \delta_{k,1}$$

and for the total error:

$$\sqrt{\sum_{i=1}^{k} [e_{k,i}]^2} = \sqrt{e_{1:k}' Q_k Q_k' e_{k,:}} = \|\delta_{k,2}\|_2$$

Note: the QR-factorization is done *directly* on the data matrix, no covariance is computed. This is the correct numerical way of doing things.

Recursive computation (RLS)

Suppose you know R_{k-1}, δ_{k-1}, how to find R_k, δ_k with a minimum number of computations?

We have:

$$X_k = \left[\begin{array}{ccc} & X_{k-1} & \\ \hline x_{k,1} & \cdots & x_{k,m} \end{array}\right],\ d_{1:k} = \left[\begin{array}{c} d_{1:k-1} \\ \hline d_k \end{array}\right],$$

and let us consider

$$\left[\begin{array}{c|c} Q_{k-1} & 0 \\ \hline 0 & 1 \end{array}\right]$$

as the first candidate for Q_k.

Then

$$\left[\begin{array}{c|c} Q'_{k-1} & 0 \\ \hline 0 & 1 \end{array}\right] \left[\begin{array}{c} X_{k-1} \\ \hline x_{k,:} \end{array}\right] = \left[\begin{array}{c} R_{k-1} \\ \hline 0 \\ \hline x_{k,:} \end{array}\right]$$

and

$$\left[\begin{array}{c|c} Q'_{k-1} & 0 \\ \hline 0 & 1 \end{array}\right] \left[\begin{array}{c} d_{1:k-1} \\ \hline d_k \end{array}\right] = \left[\begin{array}{c} \delta_{k-1,:} \\ \hline d_k \end{array}\right]$$

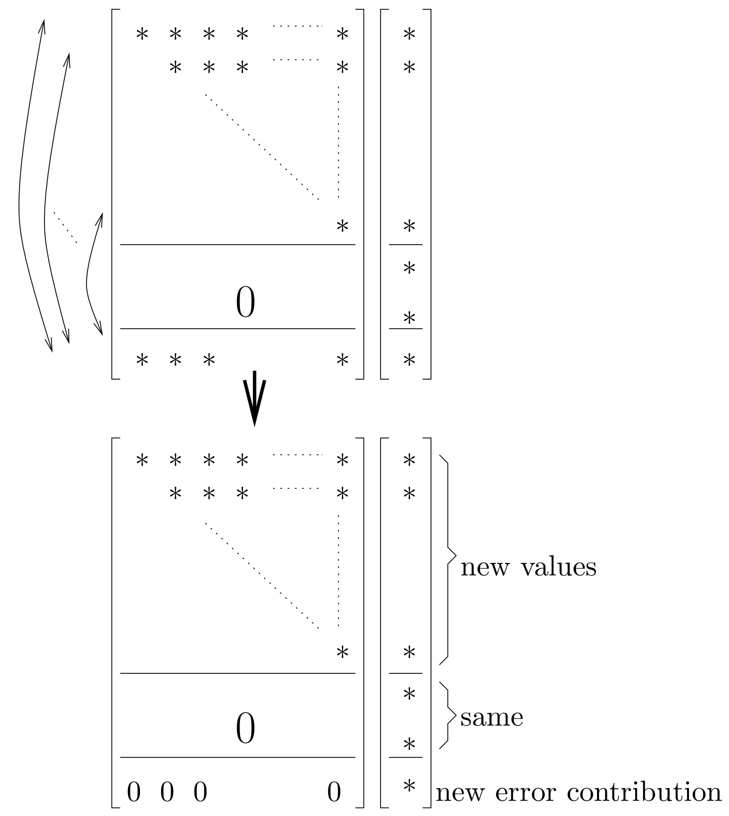
Hence we do not need Q_{k-1} anymore, the new system to be solved after the previous transformations becomes:

$$\left[\begin{array}{c} R_{k-1} \\ \hline 0 \\ \hline x_{k,:} \end{array}\right] \left[\begin{array}{c} w_{k,1} \\ \vdots \\ w_{k,m} \end{array}\right] = \left[\begin{array}{c} \delta_{k-1,:} \\ \hline d_k \end{array}\right],$$

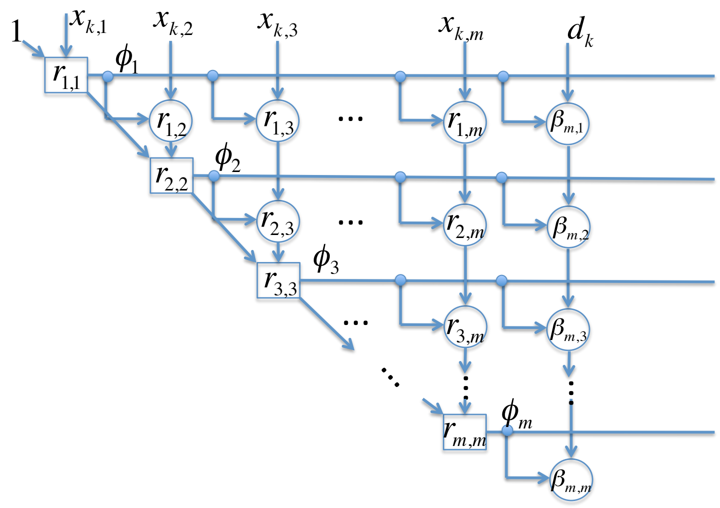
i.e.

$$\left[\begin{array}{cccc} \star & \star & \cdots & \star \\ 0 & \star & \cdots & \star \\ & & \ddots & \vdots \\ 0 & \cdots & \cdots & \star \\ \hline & & 0 & \\ \hline x_{k,1} & x_{k,2} & \cdots & x_{k,m} \end{array}\right] \left[\begin{array}{c} w_{k,1} \\ \vdots \\ w_{k,m} \end{array}\right] = \left[\begin{array}{c} \delta_{k-1,:} \\ \hline d_k \end{array}\right],$$

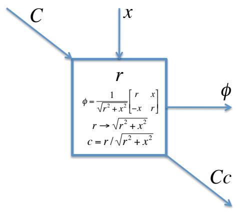
only m row transformations are needed to the find the new R_k, δ_k:



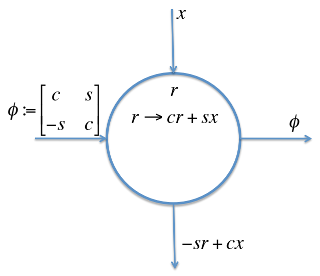
Question: how to do this computationally? A dataflow graph in which each $r_{i,j}$ is resident in a separate node would look as follows:



Initially, before the first input, all $r_{i,j} = 0$. Just before step k the $r_{i,j}$ belonging to the first (k-1) steps are resident in the nodes. There are two types of nodes:


C
x
r
$\phi = \frac{1}{\sqrt{r^2+x^2}}\begin{bmatrix} r & x \\ -x & r \end{bmatrix}$
$r \to \sqrt{r^2+x^2}$
$c = r/\sqrt{r^2+x^2}$
ϕ
Cc

vectorizing node: computes the angle ϕ from x and r



rotating node: rotates the vector $\begin{bmatrix} r \\ x \end{bmatrix}$ over ϕ.

The scheme produces $R_k, \delta_{k,k}$ and the new error:

$$\|e_k\|_2 = \sqrt{\|e_{k-1}\|_2^2 + |\delta_{k,k}|^2}$$

Reverse QR

In many applications, not the update of R_k is desired, but of $w_k = R_k^{-1}\delta_{k,1:m}$. A clever manipulation of matrices, most likely due to E. Deprettere and inspired by the old Faddeev algorithm gives a nice solution. Here is how one gets R_{k+1}^{-1} and $\delta_{k+1,1:m}$:

Observation 1: let R be an $m \times m$ invertible matrix and u an m-vector, then

$$\begin{bmatrix} R & u \\ 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} R^{-1} & -R^{-1}u \\ 0 & 1 \end{bmatrix},$$

hence, $R^{-1}u$ is implicit in the inverse shown.

Observation 2: let Q' be a unitary update which performs the following transformation (for some new $\widehat{R}$ and $\widehat{u}$, given vector x and value d):

$$Q' \begin{bmatrix} R & u \\ 0 & 1 \\ x & d \end{bmatrix} \doteq \begin{bmatrix} \widehat{R} & \widehat{u}\delta \\ 0 & \delta \\ 0 & 0 \end{bmatrix}$$

(Q is thus almost like before, the embedding is slightly different—δ is a normalizing scalar which will be discounted).

Let us call $\mathbf{R} \doteq \begin{bmatrix} R & u \\ 0 & 1 \end{bmatrix}$, similarly for $\widehat{\mathbf{R}}$, and $\xi' = [x\ d]$, then we have

$$Q' \begin{bmatrix} \mathbf{R} & 0 \\ \xi' & 1 \end{bmatrix} = \begin{bmatrix} \widehat{\mathbf{R}} & q'_{21} \\ 0 & q'_{22} \end{bmatrix} \left[\begin{array}{cc|c} I & & \\ & \delta & \\ \hline & & 1 \end{array}\right]$$

Taking inverses we find (for some a_{12} and a_{22} which originate in the process):

$$\begin{bmatrix} \mathbf{R}^{-1} & 0 \\ -\xi'\mathbf{R}^{-1} & 1 \end{bmatrix} Q \doteq \left[\begin{array}{cc|c} I & & \\ & \delta^{-1} & \\ \hline & & 1 \end{array}\right] \begin{bmatrix} \widehat{\mathbf{R}}^{-1} & a_{12} \\ 0 & a_{22} \end{bmatrix}.$$

Hence, an RQ-factorization of the known matrix on the left hand side yields an update of $\mathbf{R}^{-1}$, exclusively using new data. A data flow scheme very much like the previous one can be used.

One pass orthogonal computation of the inverse

A somewhat similar technique yields a direct orthogonal computation of the inverse of an invertible operator T of dimension $n \times n$.
Observation 1: To adopt a classical notation, let us try to solve $Tx = b$, assuming T to be invertible (the algorithm will actually show whether this is indeed the case.). Suppose we do a QR-factorization of the augmented system $\left[\begin{array}{c|c|c} T' & I & 0 \\ \hline -b' & 0 & 1 \end{array}\right]$, and let the Q-factor of dimension $(n+1) \times (n+1)$ be decomposed as $Q = \begin{bmatrix} Q_{1,1} & q_{1,2} \\ q_{2,1} & q_{2,2} \end{bmatrix}$, in which $q_{2,1}$ is a row-vector, $q_{1,2}$ a column vector and $q_{2,2}$ a scalar quantity. Then the QR-factorization will have the following form:

$$\left[\begin{array}{c|c|c} T' & I & 0 \\ \hline -b' & 0 & 1 \end{array}\right] = Q \left[\begin{array}{c|c|c} \widehat{R} & Q'_{1,1} & q'_{2,1} \\ \hline 0 & q'_{1,2} & q_{2,2} \end{array}\right]$$

It follows from the zero $(2,1)$ entry in the R-factor (on the right hand side and after pre-multiplication with Q'), that $q'_{1,2}T' - q_{2,2}b' = 0$ and hence $Tq_{1,2} = bq_{2,2}$. When T is invertible, and assuming $b \neq 0$, then $q_{2,2}$ cannot be zero, for, if it were also $Tq_{1,2} = 0$ and hence $q_{1,2} = 0$, and Q would have a full zero column, which is impossible for an orthogonal matrix. Hence, $q_{2,2}$ is invertible when T is, and we have $Tq_{1,2}q_{2,2}^{-1} = b$ giving the solution $x = q_{1,2}q_{2,2}^{-1}$.
Observation 2: from the previous observation it follows that an augmented Gentleman-Kung array will yield a one pass solution to the system of equations $Tx = b$ in "normalized form" with normalization factor $q_{2,2}$. To obtain the full inverse, one must compute n solutions, for each natural base vector e_k one. This can be done economically, using a single Gentleman-Kung array, but allowing for two phases of execution:
Phase 1: compute a QR-factorization of

$$\left[\begin{array}{c|c|c} T' & I & 0 \end{array}\right] = Q \left[\begin{array}{c|c|c} R & Q' & 0 \end{array}\right]$$

(where the last column is just a column of zeros), with an augmented (2n+1) Gentleman-Kung array. This uses the array in a "setting mode", where all the rotation angles are being determined. Next,
Phase 2: now, keeping the rotation angles determined so far, put the array in a new mode, to be called "single row compute", whereby a single global row-update operation is done, the output is computed and then the array is reset to the result of the previous step. Do this step for subsequent row inputs of

the type $\left[\begin{array}{c|c|c} e_k & 0 & 1 \end{array}\right]$. Each step will produce the subsequent columns f_k of the inverse matrix: $T^{-1} = \left[\begin{array}{ccc} f_1 & \cdots & f_n \end{array}\right]$. Also in this step new rotation angles are being produced, but they are reset to their previous value after each step.

Observation 3: the previous computation can be made more efficient, by sharing some of the vectorizations and subsequent rotations between the stages. Instead of working this out for the case treated in the precious paragraph, let us do it for the case of the data estimation, which we dealt with in section 13.2. This will lead to an attractive architecture, in which both the R-factor and its inverse can be updated in a single pass. To start with, let us assume that we dispose of an upper triangular factor R, which we have to invert. The one-pass inversion of the first basis vector $e_1' = \left[\begin{array}{cccc} 1 & 0 & \cdots & 0 \end{array}\right]$ produces

$$\left[\begin{array}{cc} Q_{1,1}' & q_{2,1}' \\ q_{1,2}' & q_{2,2} \end{array}\right]\left[\begin{array}{ccc} R' & I & 0 \\ -e_1' & 0 & 1 \end{array}\right] = \left[\begin{array}{ccc} \widehat{R} & Q_{1,1}' & q_{2,1}' \\ 0 & q_{1,2}' & q_{2,2} \end{array}\right] \tag{13.4}$$

the first column of the inverse of R as $q_{1,2}q_{2,2}^{-1}$ with just one single rotation. Starting again with the original R', one first performs one rotation on the first and second row of R', thereby eliminating $[R']_{2,1} = R_{1,2}$ leading to an update of $\widehat{R}'$ and adjoining $\hat{Q}'$ for further use:

$$\hat{Q}'\left[\begin{array}{cc} R' & I \end{array}\right] = \left[\begin{array}{cc} \widehat{R}' & \widehat{Q}' \end{array}\right] \tag{13.5}$$

(notice that $\widehat{R}'$ is not exclusively lower any more, there is a fill in at the position (1,2), while the position (2,1) has been annihilated.). This is an *update* step, and we now reset $\left[\begin{array}{cc} R' & Q' \end{array}\right] := \left[\begin{array}{cc} \hat{R}' & \widehat{Q}' \end{array}\right]$ (the original Q' was just I.). Next, there is the *compute step*, in which the -1 entry in e_2' can be eliminated against (the meanwhile new) $R_{2,2}$ - the other entries on the second row being zero, so that no fill-ins are created in this half step. The next update step now consists in further partial elimination of the entries $[R']_{1,3}$ and $[R']_{2,3}$, to a produce a solitary $[R']_{3.3}$ on row 3. This is then followed by a compute step with $-e_3'$ to produce the third row the inverse. The procedure easily continues... and yields the inverse of R'.

Putting things together

Working on the (most common and interesting) case of recursive data-updates of section 13.2, one would first input a few data vectors to create a first version of the $m \times m$ upper triangular matrix R, until there is confidence that

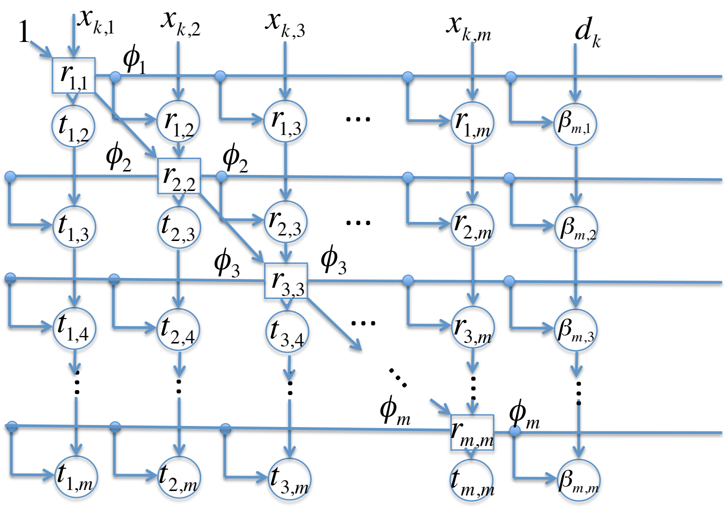


Figure 13.1: The full array. The diagonal vectorizing blocks also contain the diagonal elements of $T = R^{-1}$, namely $t_{k,k} = r_{k,k}^{-1}$. Both R and T are updated at each introduction of a new data vector x.

the matrix is indeed invertible (to be seen at the value of the diagonal entries). After this first pass, one would start working on the inverse of R, which one would locate in a complementary adjoint array. Let $T := R^{-1}$, then one would first determine T', locate it in a lower Gentleman-Kung type array (this requires running a transpose operation on the array, a somewhat awkward operation, which can be streamlined through diagonal communication), and then run through the inversion algorithm described in the previous subsection. After this initialization of T^{-1} (in transpose form), the updates using new data vectors happen in parallel between the upper and the lower array, using the same rotation arguments. The details are left to the reader, the overall array is shown in figure 13.1.

The algorithm presented may be called a true “one-pass” algorithm, because the input data flows unidirectionally through the array, first the data belonging to T, and then just the augmented input vectors e_k. It has the same overall complexity $\mathcal{O}(n^3)$ as any other method, but the avoidance of the back-substitution step is a great advantage, and the algorithm is particularly well suited for streamlined implementation on an array processor, besides being guaranteed backward stable (at the cost of some additional numerical complexity, which, however, can be cleverly be dealt with.).

The underdetermined case

This case is fundamentally different from the previous: now we have insufficient data to characterize the solution, if there is any. The situation often occurs in practice when one has to reconstruct an object (say an image) with insufficient data. Here is a short note on how orthogonal rotation can help in the RQ-form. So we have Tx ? =? b with T of dimensions $n \times m$ and $n \leq m$. An RQ-factorization of T produces

$$T = \left[\begin{array}{c|c} 0 & R \end{array}\right] \left[\begin{array}{c} Q_1 \\ \hline Q_2 \end{array}\right] \tag{13.6}$$

in which R spans the range of T and Q_1' its kernel. It may be that R is non-square, which would happen when the (partial) system is redundant or even inconsistent. Let us first assume that is not the case and R is invertible. We then have $RQ_2x = b$ and the whole collection of solutions is given by

$$x = Q_2'R^{-1}b + Q_1'Q_1v \tag{13.7}$$

in which v is an arbitrary vector with the same dimensions as b (actually one could take $w := Q_1'Q_1v$, an arbitrary vector in the kernel of T). In case T does not have full row-rank and hence R is not square a more delicate analysis is needed combining both the over- and the underdetermined case, and is left to the reader.

Francis' QR algorithm to compute the Schur eigenvalue form

A primitive version of an iterative QR algorithm to compute the Schur eigenvalue form goes as follows. Suppose that the square $n \times n$ matrix A is given. First we look for a similarity transformation with unitary matrices $U \cdot U'$ which puts A in a so called 'Hessenberg form', i.e. uppertriangular with only one additional subdiagonal, for a 4×4 matrix:

$$\begin{bmatrix} \star & \star & \star & \star \\ \star & \star & \star & \star \\ 0 & \star & \star & \star \\ 0 & 0 & \star & \star \end{bmatrix},$$

(the purpose of this step is to simplify the following procedure, it also allows refinements that enhance convergence — we skip its details except for to say that it is always possible in $(n-1)(n-2)/2$ Jacobi steps).

Assume thus that A is already in Hessenberg form, and we set $A_0 \doteq A$. A first QR factorization gives:

$$A_0 \doteq Q_0 R_0$$

and we set $A_1 \doteq R_0 Q_0$. This procedure is then repeated a number of times until A_k is nearly upper triangular (this does indeed happen sometimes - see the discussion further).

The iterative step goes as follows: assume $A_{k-1} \doteq Q_{k-1} R_{k-1}$, then

$$A_k \doteq R_{k-1} Q_{k-1}.$$

Let's analyze what we have done. A slight rewrite gives:

$$\begin{array}{rcl} Q_0 R_0 & = & A \\ Q_0 Q_1 R_1 & = & A Q_0 \\ Q_0 Q_1 Q_2 R_2 & = & A Q_0 Q_1 \\ & \cdots & \end{array}$$

This can be seen as a fixed point algorithm on the equation:

$$U\Sigma = AU$$

with $U_0 \doteq I$, we find successively:

$$\begin{array}{rcl} U_1 \Sigma_1 & = & A \\ U_2 \Sigma_2 & = & A U_1 \\ & \cdots & \end{array}$$

the algorithm detailed above produces in fact $U_k \doteq Q_0 \cdots Q_k$. If the algorithm converges, after a while we shall find that $U_k \approx U_{k+1}$ and the 'fixed point' is more or less reached.

Convergence of a fixed point algorithm is by no means assured, and even so, it is just linear. Hence, the algorithm must be improved. This is done by using at each step a clever constant diagonal 'offset' of the matrix. We refer to the literature for further information [15, 35], where it is also shown that the improved version has quadratic convergence. Given the fact that a general matrix may have complex eigenvalues, we can already see that in that case the simple version given above cannot converge, and a complex version will have to be used, based on a well-choosen complex offset. It is interesting to see that the method is related to the classical 'power method' to compute eigenvalues of a matrix. For example, if we indicate by $[\cdot]_{:,1}$ the first column of a matrix, the previous recursion gives, with

$$\mathbf{Q}_n \doteq Q_0 Q_1 \cdots Q_n$$

and $\lambda_{n+1} \doteq [R_{n+1}]_{1,1}$,

$$\lambda_{n+1}[\mathbf{Q}_{n+1}]_{:,1} = A[\mathbf{Q}_n]_{:,1}.$$

Hence, if there is an eigenvalue which is much larger in magnitude than the others, $[\mathbf{Q}_{n+1}]_{:,1}$ will converge to the corresponding eigenvector.

QZ-iterations

A further extension of the previous concerns the computation of eigenvalues of the (non singular) *pencil*

$$A - \lambda B$$

where we assume that B is invertible. The eigenvalues are values for λ and the eigenvectors are vectors x such that $(A - \lambda B)x = 0$. This actually amounts to computing the eigenvalues of AB^{-1}, but the algorithm will do so without inverting B. In a similar vein as before, we may assume that A is in Hessenberg form and B is upper triangular. The QZ iteration will determine unitary matrices Q and Z such that $A_1 \doteq QAZ$ and $B_1 \doteq QBZ$, whereby A_1 is again Hessenberg, B_1 upper triangular and A_1 is actually closer to diagonal. After a number of steps A_k will almost be triangular, and the eigenvalues of the pencil will be the ratios of the diagonal elements of A_k and B_k. We can find the eigenvectors as well if we keep track of the transformation, just as before.

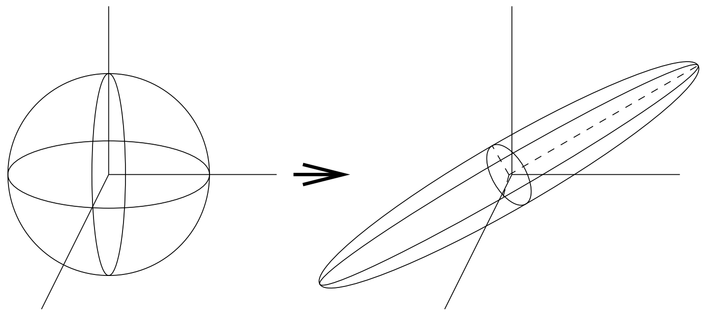
13.3 The singular value decomposition — SVD

Construction of the SVD

The all important *singular value decomposition* or SVD results from a study of the geometry of a linear transformation.

Let A be a matrix of dimensions $n \times m$, for definiteness assume $n \geq m$ (a 'tall' matrix). Consider the length of the vector Ax, $\|Ax\| = \sqrt{x'A'Ax}$, for $\|x\| = 1$.

When A is square non singular it can easily be seen that Ax moves on an ellipsoid when x moves on the unit ball. Indeed, we then have $x = A^{-1}y$ and the locus is given by $y'A^{-'}A^{-1}y = 1$, which is a bounded quadratic form in the entries of y. In general, the locus will be an ellipsoid or it will be an ellipsoid in a subspace, but the proof is more elaborate (e.g., it uses the SVD!).



The ellipsoid has a longest elongation, by definition the operator norm for A: $\sigma_1 = \|A\|$. Assume $\sigma_1 \neq 0$ (otherwise $A \equiv 0$), and take $v_1 \in \mathbf{C}^m$ a unit vector producing a longest elongation, so that $Av_1 = \sigma_1 u_1$ for some unit vector $u_1 \in \mathbf{C}^n$. It is now not too hard to show that:

$$\begin{aligned} Av_1 &= \sigma_1 u_1 \\ A'u_1 &= \sigma_1 v_1, \end{aligned}$$

and that v_1 is an eigenvector of $A'A$ with eigenvalue σ_1^2.

Proof: by construction we have $Av_1 = \sigma_1 u_1$ maximum elongation. Take any $w \perp v_1$ of unit norm and look at the effect of A on $(v_1 + \lambda w)/\sqrt{1 + |\lambda|^2}$. For very small λ the latter is $\approx (v_1 + \lambda w)(1 - \frac{1}{2}|\lambda|^2) \approx v_1 + \lambda w$, and $A(v_1 + \lambda w) = \sigma_1 u_1 + \lambda Aw$.

The norm square becomes: $v_1' A' A v_1 + \lambda v_1' A' A w + \bar{\lambda} w' A' A v_1 + O(|\lambda|^2)$ which can only be a maximum if for all $w \perp v_1$, $w' A' A v_1 = 0$. It follows that $A' u_1$ must be in the direction of v_1, easily evaluated as $A' u_1 = \sigma_1 v_1$, that σ_1^2 is an eigenvalue of $A'A$ with eigenvector v_1 and that $w \perp v_1 \Leftrightarrow Aw \perp Av_1$.

The problem can now be deflated one unit of dimension. Consider the orthogonal complement of $\mathbf{C}^m \ominus \operatorname{span}\{v_1\}$: it is a space of dimension $m - 1$, and consider the original map defined by A but now restricted to this subspace. Again, it is a linear map, and it turns out that the image is orthogonal on $\operatorname{span}(u_1)$.

Let u_2 be the unit vector in that domain for which the longest elongation σ_2 is obtained (clearly $\sigma_1 \geq \sigma_2$), and again we obtain (after some more proof) that

$$\begin{array}{rcl} Av_2 & = & \sigma_2 u_2 \\ A' u_2 & = & \sigma_2 v_2 \end{array}$$

(unless of course $\sigma_2 = 0$ and the map is henceforth zero! We already know that $v_2 \perp v_1$ and $u_2 \perp u_1$.)

The decomposition continues until an orthonormal basis for $\mathcal{R}(A')$ as $\operatorname{span}(v_1, v_2, \cdots, v_k)$ (assume the rank of A to be k) is obtained, as well as a basis for $\mathcal{R}(A)$ as $\operatorname{span}(u_1, u_2 \cdots u_k)$.

These spaces can be augmented with orthonormal bases for the kernels: $v_{k+1} \cdots v_m$ for $\mathcal{K}(A)$ and $u_{k+1} \cdots u_n$ for $\mathcal{K}(A')$. Stacking all these results produces:

$$A[v_1\, v_2 \cdots v_k\, v_{k+1} \cdots v_m] = [u_1\, u_2 \cdots u_k\, u_{k+1} \cdots u_n] \begin{bmatrix} \Sigma & 0 \\ 0 & 0 \end{bmatrix}$$

where Σ is the $k \times k$ diagonal matrix of *singular values*:

$$\Sigma = \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_k \end{bmatrix}$$

and $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_k > 0$. Alternatively:

$$A = U \begin{bmatrix} \Sigma & 0 \\ 0 & 0 \end{bmatrix} V'$$

where:

$$U = [u_1\, u_2 \cdots u_k\, u_{k+1} \cdots u_n], V = [v_1\, v_2 \cdots v_k\, v_{k+1} \cdots v_m]$$

are unitary matrices.

Singular Value Decomposition: proof

The *canonical svd form* can more easily (but maybe with less insight) be obtained directly from an eigenvalue decomposition of the Hermitean matrix $A'A$. From the form it is easy to see that

$$A'A = V \begin{bmatrix} \Sigma^2 & 0 \\ 0 & 0 \end{bmatrix} V'$$

and

$$AA' = U \begin{bmatrix} \Sigma^2 & 0 \\ 0 & 0 \end{bmatrix} U'$$

are eigenvalue decompositions of the respective (quadratic) matrices. Conversely, and starting with $A'A$ and its eigenvectors (in descending order of eigenvalues) $V = \begin{bmatrix} v_1 & \cdots & v_k & v_{k+1} & \cdots \end{bmatrix}$, with $\sigma_i > 0$ for $i = 1 \cdots k$, we may then define $u_i = Av_i\frac{1}{\sigma_i}$ for $i = 1 \cdots k$ and find $AA'u_i = \sigma_i^2 u_i$, and for the u_i with $i > k$ a basis for the orthonormal complement of the $u_i, i = 1 \cdots k$. It then follows immediately that $AV = U\Sigma$ with $U = \begin{bmatrix} u_1 & \cdots & u_k & u_{k+1} & \cdots \end{bmatrix}$. This produces an easy direct proof of the SVD.

The σ_i's are called singular values, and the corresponding vectors u_i, v_i are called pairs of singular vectors or Schmidt-pairs. They correspond to principal axes of appropriate ellipsoids. The collection of singular values is 'canonical' (i.e. unique), when there are multiple singular values then there are many choices possible.

Properties of the SVD

Since the SVD is central to the "geometry" of a linear transformation, it has a long list of important properties.

- $\|A\|_E = \sigma_1$, $\|A\|_F = \sqrt{\sum_{i=1\cdots k} \sigma_i^2}$.
- If A is square and A^{-1} exists, then $\|A^{-1}\|_E = \sigma_k^{-1}$.
- Matrix approximation: suppose you wish to approximate A by a matrix B of rank at most ℓ. Consider:

$$B = [u_1 \cdots u_\ell] \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_\ell \end{bmatrix} [v_1 \cdots v_\ell]^H .$$

Then

$$\|A-B\|_E = \sigma_{\ell+1}$$

and

$$\|A-B\|_F = \sqrt{\sum_{i=\ell+1\cdots k} \sigma_i^2}.$$

One shows that these are the smallest possible errors when B is varied over the matrices of rank ℓ. Moreover, the B that minimizes the Frobenius norm is unique.

- System conditioning: let A be a non-singular square $n \times n$ matrix, and consider the system of equations $Ax = b$. The condition number C gives an upper bound on $\|\delta x\|_2/\|x\|_2$ when A and b are subjected to variations δA and δb. We have:

$$(A+\delta A)(x+\delta x) = (b+\delta b)$$

Assume the variations small enough (say $O(\epsilon)$) with ϵ small, and small enough so that $A+\delta A$ is invertible, we find:

$$Ax + \delta A\ x + A\ \delta x \approx b + \delta b + O(\epsilon^2)$$

and since $Ax = b$,

$$\delta x \approx A^{-1}\delta b - A^{-1}\ \delta A\ x.$$

Hence (using the operator or $\|\cdot\|_2$ norm):

$$\begin{array}{rcl} \|\delta x\| & \le & \|A^{-1}\|\|\delta b\| + \|A^{-1}\|\|\delta A\|\|x\| \\ & \le & \|A^{-1}\|\frac{\|Ax\|}{\|b\|}\|\delta b\| + \|A^{-1}\|\|A\|\frac{\|\delta A\|}{\|A\|}\|x\| \end{array}$$

and finally, since $\|Ax\| \le \|A\|\|x\|$,

$$\frac{\|\delta x\|}{\|x\|} \le \|A^{-1}\|\|A\| \left\{ \frac{\|\delta b\|}{\|b\|} + \frac{\|\delta A\|}{\|A\|} \right\}.$$

Hence the condition number $C = \|A^{-1}\|\|A\| = \frac{\sigma_1}{\sigma_n}$.

A note on the strictness of the bounds: C is in the true sense an "attainable worst case". To attain the bound, e.g. when $\|\delta b\| = 0$, one must choose x so that $\|Ax\| = \|A\|\|x\|$ (which is the case for the first singular vector v_1), and δA so that $\|A^{-1}\delta A\ x\| = \|A^{-1}\|\|\delta A\|\|x\|$ which will be the case if $\|\delta A\ x\|$ is in the direction of the smallest singular vector of A, with an appropriate choice for $\|\delta A\|$ so that $\|\delta A\, x\| = \|\delta A\|\|x\|$. Since all this is possible, the bounds are attainable. However, it is highly unlikely that they will be attained in practical situations. Therefore, signal processing engineers prefer statistical estimates which give a better rendering of the situation, see further.

Example: given a large number K in $A = \begin{bmatrix} 1 & K \\ 0 & 1 \end{bmatrix}$, then $\sigma_1 \approx K$ and $\sigma_2 \approx K^{-1}$ so that $C \approx K^2$.

- Generalized inverses and pseudo-inverses: let's restrict the representation for A to its non-zero singular vectors, assuming its rank to be k:

  $$A = [u_1 \cdots u_k] \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_k \end{bmatrix} [v_1 \cdots v_k]' = \sum_{i=1}^{k} \sigma_i u_i v_i'$$

  (the latter being a sum of 'outer' products of vectors).

  The *Moore-Penrose* pseudo-inverse of A is given by:

  $$A^\dagger = [v_1 \cdots v_k] \begin{bmatrix} \sigma_1^{-1} & & \\ & \ddots & \\ & & \sigma_k^{-1} \end{bmatrix} [u_1 \cdots u_k]'.$$

  Its corange is the range of A and its range, the corange of A. Moreover, it satisfies the following properties:

  1. $AA^\dagger A = A$
  2. $A^\dagger A A^\dagger = A^\dagger$
  3. $A^\dagger A$ is the orthonormal projection on the corange of A
  4. $AA^\dagger$ is the orthonormal projection on the range of A.

  These properties actually characterize $A^\dagger$ uniquely. Any matrix B which satisfies (1) and (2) may be called a pseudo-inverse, but B is not unique with these properties except when A is square non-singular.

From the theory we see that the smallest norm solution of the least squares problem

$$\min_{x \in \mathcal{C}^n} \|Ax - b\|_2$$

is given by

$$x = A^\dagger b.$$

SVD and noise: estimation of signal spaces

Let X be a measured data matrix, consisting of an unknown signal S plus noise N as follows:

$$X = S + N$$

$$\begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\ & \vdots & & \\ & \vdots & & \end{bmatrix} = \begin{bmatrix} s_{1,1} & s_{1,2} & \cdots & s_{1,m} \\ s_{2,1} & s_{2,2} & \cdots & s_{2,m} \\ & \vdots & & \\ & \vdots & & \end{bmatrix} + \begin{bmatrix} N_{1,1} & N_{1,2} & \cdots & N_{1,m} \\ N_{2,1} & N_{2,2} & \cdots & N_{2,m} \\ & \vdots & & \\ & \vdots & & \end{bmatrix}$$

What is a good estimate of S given X? The answer is: only partial information (certain subspaces ...) can be well estimated. This can be seen as follows:

Properties of noise: law of large numbers (weak version)

Let

$$\nu = \frac{1}{n}\sum_{i=1}^{n} N_i$$

for some zero mean, stationary and uncorrelated white noise process $\{N_i\}$ with co-variance $\mathbf{E}(N_i N_j) = \sigma_N^2 \delta_{i,j}$.
The variance of ν is:

$$\begin{aligned} \sigma_\nu^2 &= \mathbf{E}(\tfrac{1}{n}\textstyle\sum N_i)^2 \\ &= \tfrac{1}{n^2}\textstyle\sum_{i,j} \mathbf{E}(N_i N_j) \\ &= \tfrac{\sigma_N^2}{n} \end{aligned}$$

and hence

$$\sigma_\nu = \frac{\sigma_N}{\sqrt{n}},$$

the accuracy improves with $\sqrt{n}$ through averaging. More generally, we have:

$$\frac{1}{n}N'N = \sigma_N^2 (I + O(\frac{1}{\sqrt{n}}))$$

(this result is a little harder to establish because of the different statistics involved, see textbooks on probability theory.)

Assume now S and N independent, and take a large number of samples. Then:

$$\begin{aligned}\tfrac{1}{n}X'X &= \tfrac{1}{n}(S'+N')(S+N)\\ &= \tfrac{1}{n}(S'S+N'N+N'S+S'N)\end{aligned}$$

(in the *long* direction), and suppose that $s_i, i=1,m$ are the singular values of S, then $\frac{1}{n}X'X$ equals

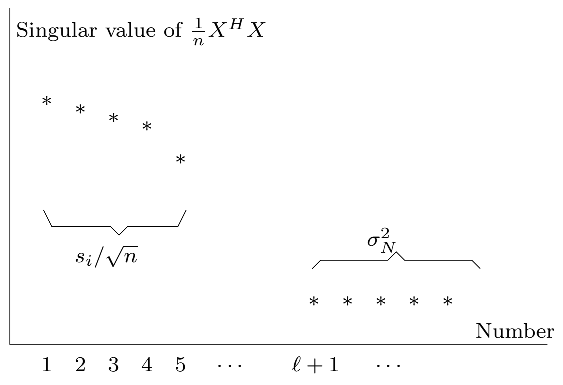
$$\left\{V_S\begin{bmatrix}\frac{s_1^2}{n} & & \\ & \ddots & \\ & & \frac{s_m^2}{n}\end{bmatrix}V_S' + \begin{bmatrix}\sigma_N^2 & & \\ & \ddots & \\ & & \sigma_N^2\end{bmatrix}\right\}\cdot\left\{I+O(\frac{1}{\sqrt{n}})\right\}.$$

A numerical error analysis of the SVD gives: $\mathrm{SVD}(A+O(\epsilon)) = \mathrm{SVD}(A) + O(\epsilon)$, and hence:

$$\frac{1}{n}X'X = V_S\begin{bmatrix}\frac{s_1^2}{n}+\sigma_N^2 & & \\ & \ddots & \\ & & \frac{s_m^2}{n}+\sigma_N^2\end{bmatrix}V_S' + O(\frac{1}{\sqrt{n}}).$$

Pisarenko discrimination

Suppose now that the original system is of rank ℓ, and we set the singular values of X out against their order, then we'll find:



We may conclude the following:

1. there is a bias σ_N^2 on the estimates of $\frac{s_i^2}{n}$

2. the error on these estimates *and* on V_S is $O(\frac{\sigma_N}{\sqrt{n}})$.

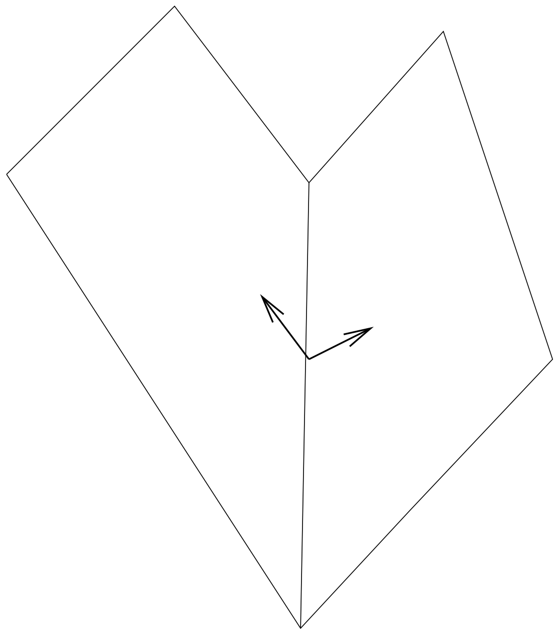
hence it benefits from the statistical averaging. This is however not true for U_S - the signal subspace - which can only be estimated $O\sigma_N$, since no averaging takes place in its estimate.

Angles between subspaces

Let

$$U = \left[\begin{array}{cccc} u_1 & u_2 & \cdots & u_k \end{array}\right]$$
$$V = \left[\begin{array}{cccc} v_1 & v_2 & \cdots & v_\ell \end{array}\right]$$

isometric matrices whose columns form bases for two spaces $\mathcal{H}_U$ and $\mathcal{H}_V$. What are the angles between these spaces?



The answer is given by the SVD of an appropriate matrix, namely $U'V$.

Let

$$U'V = A \left[\begin{array}{ccc|c} \sigma_1 & & & \\ & \ddots & & 0 \\ & & \sigma_k & \\ \hline & 0 & & 0 \end{array}\right] B'$$

be that (complete) SVD, in which A and B are unitary. The angle cosines are then given by $\cos\phi_i = \sigma_i$ and the principal vectors are given by UA and VB ($\cos\phi_i$ is the angle between the ith column of UA and VB). These are called the *principal vectors* of the intersection.

Total Least Square - TLS

Going back to our overdetermined system of equations:

$$Ax = b,$$

we have been looking for solutions of the least squares problem: an x such that $\|Ax - b\|_2$ is minimal. If the columns of A are linearly independent, then A has left inverses (in particular the pseudo-inverse defined earlier), solutions are given by any x for which $\widehat{b} \doteq Ax$ is the orthogonal projection of b on space spanned by the columns of A.

An alternative, sometimes preferable approach, is to find a modified system of equations

$$\widehat{A}x = \widehat{b}$$

which is as close as possible to the original, and such that $\hat{b}$ is actually in $\mathcal{R}(\widehat{A})$, the span of the columns of $\widehat{A}$.

What are $\widehat{A}$ and $\widehat{b}$? If the original A has m columns, then the second condition forces rank $\left[\begin{array}{cc} \widehat{A} & \widehat{b} \end{array}\right] = m$, which then has to be a rank m approximant to the augmented matrix $\left[\begin{array}{cc} A & b \end{array}\right]$. The minimal approximation in Frobenius norm is found by the SVD, now of $\left[\begin{array}{cc} A & b \end{array}\right]$. Let:

$$\begin{aligned} \left[\begin{array}{cc} A & b \end{array}\right] &= [a_{\cdot 1} \cdots a_{\cdot m}\, b] \\ &= [u_{\cdot 1} \cdots u_{\cdot m}\, u_{\cdot m+1}] \begin{bmatrix} \sigma_1 & & & \\ & \ddots & & \\ & & \sigma_m & \\ & & & \sigma_{m+1} \end{bmatrix} [v_{\cdot 1} \cdots v_{\cdot m} v_{\cdot m+1}]' \end{aligned}$$

be the desired SVD, then we define

$$\begin{aligned}\text{rank}\left[\begin{array}{cc}\widehat{A} & \widehat{b}\end{array}\right] &= [\hat{a}_{:,1}\cdots\hat{a}_{:,m}\,\hat{b}]\\ &= [u_{:,1}\cdots u_{:,m}]\left[\begin{array}{ccc}\sigma_1 & & \\ & \ddots & \\ & & \sigma_m\end{array}\right][v_{:,1}\cdots v_{:,m}]'.\end{aligned}$$

What is the value of this approximation? We know from the previous theory that the choice is such that $\|[A-\hat{A}\;\;b-\hat{b}]\|_F$ is minimal over all possible approximants of reduced rank m. This means actually that

$$\sum_{i=1}^{m}\|a_{:,i}-\hat{a}_{:,i}\|_2^2+\|b-\hat{b}\|_2^2$$

is minimal, by definition of the Frobenius norm, and this can be interpreted as follows:

The span$(a_{:,i},b)$ defines a hyperplane, such that the projections of $a_{:,i}$ and b on it are given by $\hat{a}_{:,i}$, $\hat{b}$ and the total quadratic projection error is minimal.

Pseudo-inverses revisited

let A be an $m\times n$ matrix of rank k and let

$$A=\left[\begin{array}{cc}U_1 & U_2\end{array}\right]\left[\begin{array}{cc}\Sigma & 0\\ 0 & 0\end{array}\right]\left[\begin{array}{c}V_1'\\ V_2'\end{array}\right] \tag{13.8}$$

be its SVD, with Σ collecting the non-zero singular values. The *Moore-Penrose* pseudo inverse $A^\dagger := V_1\Sigma^{-1}U_1'$ solves the optimization problem

$$A^\dagger b=\text{argmin}_{x\in\{\text{argmin}_\xi\|A\xi-b\|_2\}}(\|x\|_2) \tag{13.9}$$

for all b, or in words: of all the ξ that minimize the least square error $\|A\xi-b\|_2$, $A^\dagger b$ has minimal quadratic norm, and this property is valid no matter what b is.

A general expression for any pseudo-inverse A^+ of A (i.e., a matrix A^+ for which the two equations $A^+=A^+AA^+$ and $A=AA^+A$ hold) is given by

$$A^+=A^\dagger+V_1XU_2'+V_2YU_1'+V_2Y\Sigma XU_2'(=(V_1\Sigma^{-1}+V_2Y)\Sigma(\Sigma^{-1}U_1'+XU_2')). \tag{13.10}$$

in which X and Y are conformal but otherwise arbitrary matrices. (Proof is by direct computation on the definition.)

Not all of those will minimize $\|AA^+b - b\|_2$ for all b. We find, in fact:

$$AA^+b - b = -U_2U_2'b + U_1\Sigma XU_2'b \tag{13.11}$$

whose norm-squared is $\|U_2'b\|^2 + \|\Sigma XU_2'b\|^2$, which for all b is minimally $\|U_2'\|^2b$ when $X = 0$ (since Σ is non-singular)—Y remains arbitrary but plays a role in the norm of x..

Two important sub-cases are worth emphasizing.

The system is non-singular but overdetermined

In this case we have $A = \begin{bmatrix} U_1 & U_2 \end{bmatrix}\begin{bmatrix} \Sigma \\ 0 \end{bmatrix}V'$ (its kernel is zero), and a general pseudo-inverse is given by $A^+ = V\begin{bmatrix} \Sigma^{-1}U_1' & XU_2' \end{bmatrix}$. Clearly $\|A^+b\|_2$ will not be minimal for all b unless $X = 0$: the set of minimizers reduces to just one vector, namely $V\Sigma^{-1}\beta_1$ with $\beta_1 = U_1'b$.

The system is non-singular but underdetermined

Now we have $A = U\begin{bmatrix} \Sigma & 0 \end{bmatrix}\begin{bmatrix} V_1' \\ V_2' \end{bmatrix}$ and a general pseudo-inverse is given by $A^+ = \begin{bmatrix} V_1\Sigma^{-1} \\ V_2Y \end{bmatrix}U'$. Putting $x = A^+b$ with any Y will minimize $\|Ax - b\|_2$ but will not be of minimal norm itself (for all b). Although the more general solution obtained may not have minimal norm, it may have other properties: e.g., minimality in another norm (often ℓ_1) and be sparse.

13.4 Optimization

Let $x \in \mathbf{R}^n$ be a real vector and $\mathcal{L}x$ a real scalar function of x—often $\mathcal{L}(x) \geq 0$ is a 'cost-function' that measures how undesirable the choice of x is in a given problem. The first question one might ask is: which x's minimize $\mathcal{L}(x)$? To answer that question, let us assume that $\mathcal{L}$ is continuously differentiable, and let us define $\partial_x\mathcal{L} := \mathrm{col}(\frac{\partial\mathcal{L}}{\partial x_1}, \cdots, \frac{\partial\mathcal{L}}{\partial x_n})$. A necessary condition on x for $\mathcal{L}(x)$ to be a (local or global) minimum is

$$\partial_x\mathcal{L}(x) = 0 \tag{13.12}$$

i.e., the gradient of $\mathcal{L}(x)$ at such a minimum x_m is zero. This will be a true (local) minimum if, in addition, also the *Pfaffian* P is strictly positive definite, where P is an $n \times n$ matrix with

$$P_{i,j} := \frac{\partial^2 \mathcal{L}}{\partial x_i \partial x_j} \tag{13.13}$$

at the point x_m. If, on the other hand, the Pfaffian is strictly negative definite, then x_m makes $\mathcal{P}(x_m)$ into a local maximum. (If $P(x_m)$ is only semi-positive definite, then the situation requires more study. If it has mixed inertia, then x_m will only be a *saddle point.*)

Optimization with equality constraints

Suppose that x is subjected to some constraints of the type $f_i(x) = 0$ for $i = 1, \cdots \ell$ (and of course $\ell < k$), also assumed to be continuously differentlable, defining a manifold on which x should lie. How can the optimization criterion be adapted to this case? It takes some thinking to realize that the condition can now be formulated as follows: *the projection of the gradient of $\mathcal{L}$ at a local extremum on the manifold defined by the constraints should be zero.* This can, alternatively, be formulated as: *the gradient of $\mathcal{L}$ should locally be a linear combination of the individual gradients of the constraint functions*, or, algebraically: *there should exist coefficients λ_i $(i = 1 \cdots \ell)$ such that*

$$\partial_x \mathcal{L} = \lambda' \partial_x f (= \sum_{i=1}^{\ell} \lambda_i \partial_x f_i) \tag{13.14}$$

(this means that the local altitude line of $\mathcal{L}$ is tangent to the constraint manifold, because the hyperspace generated by $\lambda' \partial_x f$ at x_m is orthogonal to the intersection of the $f_i(x) = 0$ manifolds).

Conditions that then produce the local extremum can then be formulated using the augmented cost function $\mathcal{L}_a = \mathcal{L} - \lambda' \partial_x f$ together with the constraints:

$$\begin{cases} \partial_x \mathcal{L}_a & = & 0 \\ f_1(x) & = & 0 \\ & \vdots & \\ f_\ell(x) & = & 0 \end{cases} \tag{13.15}$$

Quadratic optimization

A typical quadratic cost function has the form $\mathcal{L} = (x'M' + b'N')(Mx + Nb)$ with M and N given matrices and b a given vector. Such a cost function is obviously positive. Application of the previous theory requires some care. We have $\partial_x x'MM'x = 2Mx$ and $\partial_x b'N'Mx = \partial_x x'M'Nb = M'Nb$, everything being assumed to be real. It follows that the problem of unconstrained quadratic optimization with this cost function is solved by

$$2M'Mx + 2M'Nb = 0 \tag{13.16}$$

or $M'Mx = -M'Nb$, which reduces it to what has been said of pseudo-inverses already. The solution will be unique when $M'M$ is definite.

Suppose a constraint $Px = p$ is added, where P is another matrix and p an appropriately sized vector p. Constrained optimization now gives (with $\mathcal{L}_a = \mathcal{L} + \lambda'(Px - p)$ and $\partial_x \lambda' Px = P'\lambda$)

$$\left\{ \begin{array}{ccc} 2M'Mx + P'\lambda & = & -2M'Nb \\ Px & = & p \end{array} \right. \tag{13.17}$$

or, in matrix form

$$\left[\begin{array}{cc} 2M'M & P' \\ P & 0 \end{array} \right] \left[\begin{array}{c} x \\ \lambda \end{array} \right] = \left[\begin{array}{c} -2M'Nb \\ p \end{array} \right] \tag{13.18}$$

another system of linear equations, with a matrix that is not typically positive definite and hence requires further specific analysis (this latter matrix is sometimes called the 'Hamiltonian').

Chapter 14

Appendix II: Linear time-invariant filters

Klaus Diepold

Menu

Hors d'oeuvre
Properties of linearity and time-invariance

First course
Linear convolution

Second course
Cyclic convolution and cyclic Toeplitz matrices

Third course
The shift operator and its z-transform

Dessert
Diagonal expansions and discussion topics

This chapter deals with the classical theory of linear filters and systems, as it is being taught in signal processing courses. Its purpose is to provide, besides a handy summary of important properties, a connection with system theory as considered in other chapters and a motivation for the notation used in the more general setting of time-variant environments.

14.1 Properties of linearity and time-invariance

In digital signal processing and digital communications one is often interested in devising an algorithm, to be executed on a computer or a computing system, which receives as its input (scalar real or complex) sequences of the type $u = [u_k]_{k=m\cdots}$ starting at some index point $m \leq n$ (often chosen 0 or 1) and produces output sequences $y = [y_k]_{k=n\cdots}$ such that the relation between the input sequence and the output sequence is given by a function $\mathcal{T}\{\cdot\}$ (a 'behavior') with the additional properties of being time invariant and linear. Figure 14.1 represents such a computing system, in which the input sequence u is fed, and mapped by $\mathcal{T}$ to the output. This is denoted mathematically as

$$y = \mathcal{T}\{u\}.$$

$k = \cdots, -1, 0, 1, 2, \cdots$ represents the time index for elements of the discrete-time sequence.

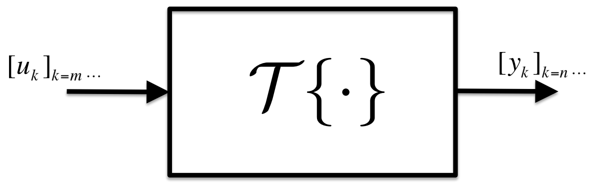


Figure 14.1: Input Output description of a linear time-invariant computing system or filter.

The system will be called *linear* (at least from an input-output perspective), when the superposition principle holds, that is, for two input sequences u^1 and u^2 and scalars α_1 and α_2 the corresponding output sequences add like

$$y^1 = \mathcal{T}\{u^1\}, \quad y^2 = \mathcal{T}\{u^2\} \quad \Rightarrow \quad \alpha_1 y^1 + \alpha_2 y^2 = \mathcal{T}\{\alpha_1 u^1 + \alpha_2 u^2\}.$$

The property of *time-invariance* states that the function $\mathcal{T}\{\cdot\}$ in invariant to shifts along the time axis, i.e. shifting the input sequence over τ units $\sigma_\tau(u) = [u_{k-\tau}]_{k=n-\tau\cdots}$ causes a corresponding shift in the output sequence $\sigma_\tau y$:

$$\mathcal{T}\{\sigma_\tau u\} = \sigma_\tau \mathcal{T}\{u\},$$

without causing further changes.

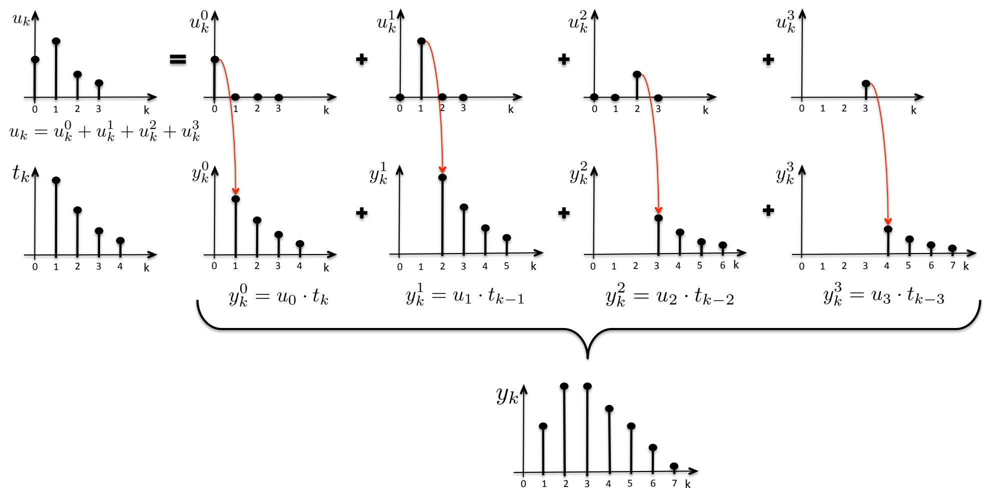
In many applications, the first significant index n in the output sequence will follow the first index m of the input sequence, but it may happen that other conventions are used, e.g., when the system is computing symmetrical running averages (often called 'smoothing'). We shall say that a system is *causal* when for each k the value of the output y_k only depends on values of inputs u_i for $i \leq k$, i.e. input values preceding or equal to k.

14.2 Exploiting linearity and time-Invariance

We can exploit the features of linearity and time-invariance to compute the output signal y_k of the system in a streamlined way, namely using only shifted versions of a special output called the *impulse response*, which is defined as the response of the system for a single input of magnitude 1 at index point 0. We show an example in Figure 14.2. We take an input sequence u_k starting at 0 and of length 4, and decompose it into the sum of individual impulses u_k^i, which are shifted in time. Each of these individual impulses generates a shifted version of the impulse response $t := [t_k]_{k=0\cdots 3}$, here assumed to be of length 4 also. Each of these shifted versions of the impulse response is weighted with the value of the corresponding input impulse u_k^i creating the individual weighted and shifted responses $y_k^i, i = 0, 1, 2, 3$. Here we exploit the time-invariance property of the LTI-system, which says that the shifted versions of the impulse response are derived from the identical impulse response t_k. Finally, the output signal y_k is generated as the sum of all the individually weighted and shifted impulse responses. For this step we exploit the linearity property of the LTI-system (superposition principle).

Putting all this together, and assuming values zero for inputs u_k and impulse response values t_k when $k \leq 0$ we can observe that the LTI-system determines the output sequence y_k as

$$y_k = \sum_{i=-\infty}^{\infty} t_{k-i} \cdot u_i, \quad k = 0, 1, \ldots m + n - 2,$$


u_k
u_k^0
u_k^1
u_k^2
u_k^3
$u_k = u_k^0 + u_k^1 + u_k^2 + u_k^3$
t_k
y_k^0
y_k^1
y_k^2
y_k^3
$y_k^0 = u_0 \cdot t_k$
$y_k^1 = u_1 \cdot t_{k-1}$
$y_k^2 = u_2 \cdot t_{k-2}$
$y_k^3 = u_3 \cdot t_{k-3}$
y_k
k

$$y_k = y_k^0 + y_k^1 + y_k^2 + y_k^3 = u_0 \cdot t_k + u_1 \cdot t_{k-1} + u_2 \cdot t_{k-2} + u_3 \cdot t_{k-3}$$

$$y_k = \sum_{i=-\infty}^{\infty} t_{k-1} u_i, \quad k = 0, 1, 2, \ldots$$

Figure 14.2: Computing the output signal of an LTI system with convolutions. The first row denotes the decomposition of an input sequence u_k into the sum of individual impulses u_k^i, which are shifted in time. Each of the individual impulses generates a shifted version of the impulse response, which weighted with the value of the corresponding impulse u_k^i. These impulse responses are shown in the second row. Finally, the output signal y_k is generated as the sum of the individually weighted and shifted impulse responses.

which is called a *linear convolution* of the sequence t with the sequence u, often written as $y = t \star u$.

14.3 Linear convolution

As the output sequence y_k is determined by the convolution operation, we now discuss how the computational task to compute the convolution of two signals can be made efficient. To this end let's consider a finite, discrete-time sequence $u_k, k = 0, 1, 2, \dots m-1$. We can use the elements of the sequence to form the vector (with $[\cdot]'$ the transposition of a vector)

$$\mathbf{u} = \begin{bmatrix} u_0 & u_1 & \dots & u_{m-1} \end{bmatrix}', \quad \mathbf{u} \in R^m.$$

We then feed this sequence as the input to a (discrete-time) linear, time-invariant system that is described by its associated impulse response $t_k, k = 1, 2, \dots m-1$, the entries of this sequence will be cast into the vector

$$\mathbf{t} = \begin{bmatrix} t_0 & t_1 & \dots & t_{n-1} \end{bmatrix}', \quad \mathbf{t} \in R^n.$$

For now we restrict the discussion to finite time series to keep things simple. Computing the convolution of the two time series amounts to evaluate the convolution sum given as

$$y_k = \sum_{i=-\infty}^{\infty} t_{k-i} \cdot u_i, \quad k = 0, 1, \dots m+n-2. \tag{14.1}$$

If the input sequence u_k has the length m and the impulse response t_k has the length n, then the length of the output signal is $N = m+n-1$. The samples of the output sequence y_k can also be summarized in a output sequence vector

$$\mathbf{y} = \begin{bmatrix} y_0 & y_1 & \dots & y_{m+n-2} \end{bmatrix}', \quad \mathbf{y} \in R^{m+n-1}.$$

The output sequence vector $\mathbf{y}$ of the linear system is computed as the *linear* convolution of the two sequences $\mathbf{t}$ and $\mathbf{u}$, denoted by

$$\mathbf{y} = \mathbf{t} \star \mathbf{u}, \quad \mathbf{y} \in R^N, \quad N = m+n-1.$$

As an example, we manually convolve an input signal u_k of length $m = 4$ with an impulse response t_k of length $n = 4$ to produce an output signal y_k of length $N = m + n - 1 = 7$,

	0	0	0	u_0	u_1	u_2	u_3	0	0	0
	t_3	t_2	t_1	t_0						
y_0				u_0t_0						
		t_3	t_2	t_1	t_0					
y_1				u_0t_1	$+u_1t_0$					
			t_3	t_2	t_1	t_0				
y_2				u_0t_2	$+u_1t_1$	$+u_0t_2$				
				t_3	t_2	t_1	t_0			
y_3				u_0t_3	$+u_1t_2$	$+u_2t_1$	$+u_3t_0$			
					t_3	t_2	t_1	t_0		
y_4					u_1t_3	$+u_2t_2$	$+u_3t_1$			
						t_3	t_2	t_1	t_0	
y_5						u_2t_3	$+u_3t_2$			
							t_3	t_2	t_1	t_0
y_6							u_3t_3			

Alternatively to this somewhat manual operation, we can re-express the convolution sum in terms of a matrix-vector multiplication $\mathbf{y} = \mathbf{t} \star \mathbf{u} = \mathbf{T}_\infty \cdot \mathbf{u}$, where the infinite matrix

$$\begin{bmatrix} \vdots \\ y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ \vdots \end{bmatrix} = \underbrace{\begin{bmatrix} \vdots & \vdots & \vdots & \vdots \\ t_0 & 0 & 0 & 0 \\ t_1 & t_0 & 0 & 0 \\ t_2 & t_1 & t_0 & 0 \\ t_3 & t_2 & t_1 & t_0 \\ 0 & t_3 & t_2 & t_1 \\ 0 & 0 & t_3 & t_2 \\ 0 & 0 & 0 & t_3 \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix}}_{\mathbf{T}_\infty} \cdot \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \end{bmatrix} \tag{14.2}$$

appears. Note that shifted versions of the impulse-response $\mathbf{t}$ have been used as columns to generate a matrix that has identical entries along diagonals.

The matrix $\mathbf{T}$ with this specific structure is a finite dimensional representation of a more general *convolution operator* and is called a *Toeplitz* matrix after the German mathematician Otto Toeplitz. The current Toeplitz matrix is a rectangular matrix, which can not be inverted nor does it have eigenvalues and eigenvectors. Furthermore, in order to enable efficient algorithms for computing convolutions we complete the matrix appropriately. We explain the details in the next section.

14.4 Cyclic convolution and cyclic Toeplitz matrices

One clever way to proceed is by considering a related but different type of matrix first, namely the *Cyclic Toeplitz* matrix $\mathbf{T}_c$ of size $m \times m$ given by

$$\mathbf{T}_c = \begin{bmatrix} t_0 & t_3 & t_2 & t_1 \\ t_1 & t_0 & t_3 & t_2 \\ t_2 & t_1 & t_0 & t_3 \\ t_3 & t_2 & t_1 & t_0 \end{bmatrix}.$$

With this cyclic Toeplitz matrix the cyclic convolution of the signals t_k and u_k is defined as

$$y_k = t_k \otimes u_k, \quad \mathbf{y} = \mathbf{T}_c \mathbf{u}.$$

Note that the application of the cyclic convolution produces a different result than the linear convolution.

Using a cyclic Toeplitz matrix, we still would like to compute the linear convolution of t_k and u_k. To this end, we can pad the vectors $\mathbf{t}$ and $\mathbf{u}$ appropriately with additional zeros, such that the additional columns in $\mathbf{T}_c$ do not modify the output vector $\mathbf{y}$. We still denote the zero-padded input vector as $\mathbf{t}$ and $\mathbf{u}$. Building a cyclic Toeplitz matrix $\mathbf{T}_c$ using the padded vector $\mathbf{t}$ the linear convolution is computed correctly as

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \end{bmatrix} = \left[\begin{array}{cccc|ccc} t_0 & 0 & 0 & 0 & t_3 & t_2 & t_1 \\ t_1 & t_0 & 0 & 0 & 0 & t_3 & t_2 \\ t_2 & t_1 & t_0 & 0 & 0 & 0 & t_3 \\ t_3 & t_2 & t_1 & t_0 & 0 & 0 & 0 \\ 0 & t_3 & t_2 & t_1 & t_0 & 0 & 0 \\ 0 & 0 & t_3 & t_2 & t_1 & t_0 & 0 \\ 0 & 0 & 0 & t_3 & t_2 & t_1 & t_0 \end{array}\right] \cdot \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \tag{14.3}$$

or in short hand

$$y = T_c \cdot u. \tag{14.4}$$

The cyclic extension of the Toeplitz matrix is in close relation with the known effects of periodic replication of the signal and its Fourier spectrum when processing signals which have been sampled in the time domain and the frequency domain.

14.5 Similarity transformation

It turns out that cyclic Toeplitz matrices have a special eigen-structure, which can be exploited to efficiently compute convolutions, using Fast Fourier Transforms. If we multiply both sides of Equation (14.4) from the left with a non-singular matrix $\mathbf{Q}$, we arrive at

$$\mathbf{Q} \cdot y = \mathbf{Q} \cdot T_c \cdot u. \tag{14.5}$$

As a next step we insert the identity matrix $\mathbf{I}_n = \mathbf{Q}^{-1}\mathbf{Q}$ between the factors T_c and u on the right hand side of Equation (14.5). This leads us to

$$\mathbf{Q} \cdot y = \mathbf{Q} \cdot T_c \cdot \mathbf{Q}^{-1}\mathbf{Q} \cdot u. \tag{14.6}$$

Inserting a few brackets for improved readability we get

$$(\mathbf{Q} \cdot y) = (\mathbf{Q} \cdot T_c \cdot \mathbf{Q}^{-1}) \cdot (\mathbf{Q} \cdot u), \tag{14.7}$$

where we can read off the following abbreviated notation

$$\mathbf{y} = \mathbf{\Lambda} \cdot \mathbf{u}, \tag{14.8}$$

where the quantities $\mathbf{y}, \mathbf{\Lambda}$ and $\mathbf{u}$ are defined as

$$\mathbf{y} := \mathbf{Q} \cdot y, \qquad \mathbf{u} := \mathbf{Q} \cdot u, \qquad \mathbf{\Lambda} := \mathbf{Q} \cdot T_c \cdot \mathbf{Q}^{-1}. \tag{14.9}$$

The identity

$$T_c = \mathbf{Q}^{-1} \cdot \mathbf{\Lambda} \cdot \mathbf{Q} \tag{14.10}$$

can easily be identified as an similarity transformation of T_c by $\mathbf{Q}$. The similarity transformation implies that the matrices T_c and $\mathbf{\Lambda}$ share the same eigenvalues.

14.6 Eigenvalue decomposition of cyclic Toeplitz matrices

The specific structure of the cyclic Toeplitz matrix is also called a *Circulant*. We want to determine a eigenvalue decomposition for a cyclic Toeplitz matrix, i.e. we want to compute the eigenvectors and the eigenvalues ($Ax = \lambda x$) for a cyclic Toeplitz matrix given as

$$T_c = \begin{bmatrix} t_0 & t_{N-1} & t_{N-2} & \dots & t_1 \\ t_1 & t_0 & t_{N-1} & \dots & t_2 \\ t_2 & t_1 & t_0 & \dots & t_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ t_{N-1} & t_{N-2} & t_{N-3} & \dots & t_0 \end{bmatrix}. \tag{14.11}$$

This is actually a pretty easy exercise once one has seen how to exploit the circulant symmetry. Let

$$C := \left[\begin{array}{ccc|c} & & & 1 \\ \hline 1 & & & \\ & \ddots & & \\ & & 1 & \end{array}\right] \tag{14.12}$$

be a permutation matrix that puts the last element first when applied to a column vector and shifts the others one notch down, then we see easily that

$$CT_c = T_cC \tag{14.13}$$

i.e., C and T_c commute. This actually expresses the circulant symmetry. Commutation relations have important effects on eigenspaces: a full eigenspace of C will be an eigenspace of T_c. Because of its simple form, eigenspaces of C are easy to find and hence produce eigenspaces of T_c[1]. Suppose $x := \begin{bmatrix} x_0 & \cdots & x_{N-1} \end{bmatrix}'$ is an eigenvector of C with eigenvalue λ, then we have $Cx = x\lambda$, which means $x_{N-1} = \lambda x_0$, $x_0 = \lambda x_1$, $\cdots$, and hence $x_0 = \lambda^N x_0$, and likewise for all the other entries. It follows that we have found a solution if $\lambda^N = 1$, hence if λ is an N^{th} root of unity. Since there are N such roots and the dimension of C is $N \times N$ we have found a complete set of eigenvalues of C.

[1]Finding eigenspaces of symmetry operators is standard practice in Physics!

For the complete set of eigenvectors of C (which will at the same time provide a complete set of eigenvectors of T_c thanks to commutation), let q denote a root of the scalar equation $q^N = 1$, and set

$$x = \begin{bmatrix} q^0 & q^1 & q^2 & \dots & q^{N-1} \end{bmatrix}'$$

$$y = T_c \cdot x = \begin{bmatrix} y_0 & y_1 & y_2 & \dots & y_{N-1} \end{bmatrix}'$$

Looking at the first entry of the vector y

$$y_0 = t_0 + t_{N-1}q + t_{N-2}q^2 + \cdots + t_1 q^{N-1}$$

we observe that y_0 satisfies the following system of equations

$$\begin{array}{rcl} y_0 & = & t_0 + t_{N-1}q + t_{N-2}q^2 + \cdots + t_1 q^{N-1} \\ y_1 = qy_0 & = & t_1 + t_0 q + t_{N-1}q^2 + \cdots + t_2 q^{N-1} \\ \vdots & & \vdots \\ y_{N-1} = q^{N-1}y_0 & = & t_{N-1} + t_{N-2}q + t_{N-3}q^2 + \cdots + t_0 q^{N-1}, \end{array}$$

which we can summarize compactly as

$$T_c \cdot x = y_0 \cdot x.$$

It follows that $\lambda = y_0$ is a characteristic root (eigenvalue) of T_c with the associated characteristic vector (eigenvector)

$$x = \begin{bmatrix} 1 & q & q^2 & \dots & q^{N-1} \end{bmatrix}'.$$

Since the equation $q^N = 1$ has N distinct roots $y_k, k = 0, 2, \dots N-1$, we see that we obtain N distinct characteristic vectors $x_{:,k}, k = 0, 1, 2, \dots N-1$. Let now $q := e^{j2\pi/N}$ be the most primitive non trivial solution to the equation $q^N = 1$, then other roots are given by q^k. Consequently, we have the complete set of characteristic roots and vectors in this way, i.e.,

$$T_c \cdot x_{:,k} = q^k \cdot x_{:,k}$$

holds, with $x_{:,k} = \begin{bmatrix} q^0 & q^k & \cdots & q^{(N-1)k} \end{bmatrix}'$. The set of all eigenvectors $x_{:,k}$ can be put together as the columns of a matrix

$$\mathbf{Q} = \begin{bmatrix} x_{:,0} & x_{:,1} & x_{:,2} & \dots & x_{:,N-1} \end{bmatrix}$$

with general entry $\mathbf{Q}_{i,k} = q^{ik}$. With this special choice for the matrix $\mathbf{Q}$ the Equation (14.10) represents the *Eigenvalue Decomposition* of $\mathbf{T}_c$ with

$$\mathbf{\Lambda} = \mathbf{Q}^{-1} \cdot T_c \cdot \mathbf{Q} = \begin{bmatrix} \lambda_0 & & & 0 \\ & \lambda_1 & & \\ & & \ddots & \\ 0 & & & \lambda_{N-1} \end{bmatrix}. \tag{14.14}$$

That is, $\mathbf{\Lambda}$ contains the eigenvalues of T_c as its diagonal entries and the matrix $\mathbf{Q}$ contains the corresponding eigenvectors. That implies that for computing the Eigenvalue decomposition of the cyclic Toeplitz matrix T_c we already have the corresponding eigenvectors given a priori as the columns of the matrix $\mathbf{Q}$. With those quantities given, computing the pertaining eigenvalues is an easy task.

14.7 The discrete Fourier transform

Let us have a closer look at the matrix $\mathbf{Q}$, which is composed of the characteristic vectors $x^i, i = 1, 2, \ldots N-1$,

$$\mathbf{Q} = \frac{1}{\sqrt{N}} \cdot \begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & q & q^2 & \ldots & q^{N-1} \\ 1 & q^2 & q^4 & \ldots & q^{2(N-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & q^{N-1} & & \ldots & q^{(N-1)^2} \end{bmatrix},$$

which we conveniently normalize by a factor $1/\sqrt{N}$ to arrive at a matrix $\mathbf{Q}$ having a number of special properties, such as (with '$\cdot^*$' indicating Hermitian transposition: $[\mathbf{Q}^*]_{i,k} := \overline{\mathbf{Q}_{k,i}}$)

- Unitarity: $\mathbf{Q}^*\mathbf{Q} = \mathbf{I}_N \quad \Rightarrow \quad \mathbf{Q}^* = \mathbf{Q}^{-1}$
- $\mathbf{Q}^2$ is a permutation matrix (different from C before, actually $[\mathbf{Q}^2]_{i,k} = \delta_{(i+k)\text{mod}N}$ with δ the Dirac impulse: $\delta_i = 1$ when $i = 0$, otherwise 0.)
- Cyclic: $\mathbf{Q}^3 = \mathbf{Q}^*, \quad \mathbf{Q}^4 = \mathbf{I}_N$.

This matrix $\mathbf{Q}$ is seen as defining the *Discrete Fourier Transform* (DFT) on sequences of length N.

In the final form, Equation 14.10 reduces the matrix-vector multiplication calculation to vector-vector multiplication. This approach has its well-known parallel by the simplification of the convolution in Equation 14.1 to multiplication of the Fourier spectrum of the impulse response $\mathbf{T}$ and the input vector $\mathbf{u}$. Actually, the diagonal eigenvalues of $\mathbf{T}_c$ in $\mathbf{\Lambda}$ are the same as the Fourier spectrum of the (zero-padded) $\mathbf{T}$ (Note that the zero-padding is not an essential part of this process). We identify the matrix $\mathbf{Q}$ to represent the *Discrete Fourier Transform* $\mathcal{F}$, such that the Fourier spectrum of the sequences $\mathbf{u}$ and $\mathbf{h}$ are determined by

$$\mathbf{\Lambda} = \mathbf{Q} \cdot T_c = \mathcal{F}\{T_c\}, \quad \mathbf{u} = \mathbf{Q} \cdot u = \mathcal{F}\{u\}, \quad \mathbf{y} = \mathbf{Q} \cdot y = \mathcal{F}\{y\} \tag{14.15}$$

and the convolution theorem associated with the Fourier Transformation given as

$$y = \mathcal{F}^{-1}\{\mathcal{F}\{T_c\} \cdot \mathcal{F}\{u\}\} = \mathcal{F}^{-1}\{\mathbf{\Lambda} \cdot \mathbf{u}\} \tag{14.16}$$

then becomes $\mathbf{y} = \mathbf{\Lambda} \cdot \mathbf{u}$ or more explicitly

$$\begin{bmatrix} \mathbf{y}_0 \\ \mathbf{y}_1 \\ \vdots \\ \mathbf{y}_{N-1} \end{bmatrix} = \begin{bmatrix} \lambda_0 & & & 0 \\ & \lambda_1 & & \\ & & \ddots & \\ 0 & & & \lambda_{N-1} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{u}_0 \\ \mathbf{u}_1 \\ \vdots \\ \mathbf{u}_{N-1} \end{bmatrix}. \tag{14.17}$$

The previously derived identities are the basis on which the technique of *Fast Convolution* is based on. For real computations of the DFT we use an efficient algorithm, called the *Fast Fourier Transform* or *FFT*. The FFT computes the eigenvalues of the cyclic Toeplitz matrix $\mathbf{T}_c$ using $\mathcal{O}(n \log n)$ arithmetic operations, which is much more efficient than a straight matrix-vector multiplication, which takes $\mathcal{O}(n^2)$ operations, if n is the length of the corresponding vectors.

14.8 Computing in the Fourier domain

The convolution operation costs $\mathcal{O}(n^2)$ operations, if n is the length of the signals we are working on. Alternatively, exploiting the convolution theorem of the Discrete Fourier Transformation we can compute the output signal of a linear time-invariant system in the frequency domain. The Discrete Fourier Transformation of a given signal can be computed very efficient way by the

Fast Fourier Transform (FFT). This leads to the so-called *Fast Convolution* method, which requires $\mathcal{O}(n \log n)$ operations, hence a significant saving in computations when comparing the Fast Convolution with the direct execution of the convolution sum. Figure 14.3 shows the detour through frequency domain for computing the convolution.

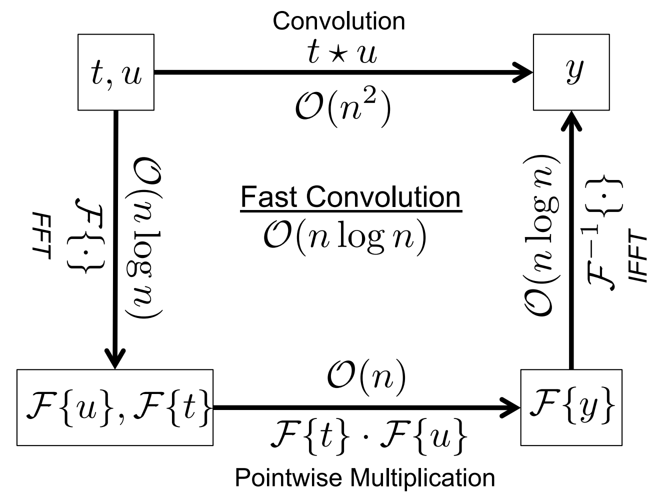


Figure 14.3: Schematic Procedure for Computing Fast Convolutions.

This frequency domain method using the FFT works very well, is well established and its efficiency can be regarded as one of the major cornerstones for the tremendous success of digital signal processing during the past 30 years. However, it is based on the assumption that the systems involved are linear and time-invariant. If one of these two assumptions is violated, then the use of frequency domain tools is no longer applicable. If the system is time-varying, then the impulse response changes with time. That implies that the columns in the convolution operator $\mathbf{T}_\infty$ are not shifted versions of a single impulse response t_k and hence the operator looses its Toeplitz property.

14.9 Linear convolution using the z-transform

A popular tool for dealing with discrete time signals and systems is the z-transformation, which is defined as

$$T(z) = \sum_{i=0}^{\infty} t_i z^i, \quad U(z) = \sum_{i=0}^{\infty} u_i z^i. \tag{14.18}$$

Please note that we use a positive exponent in our definition for the z-transform. This is a minor modification in comparison to most standard engineering text books (it appears more logical to use z as a symbol for the 'forward shift' and z^{-1} for the backward shift. This also often simplifies the notation considerably, as can be seen in the chapters on LTV sytems, where the forward shift will be written Z.). For the example this amounts to

$$T(z) = t_0 + t_1 z + t_2 z^2 + t_3 z^3, \quad U(z) = u_0 + u_1 z + u_2 z^2 + u_3 z^3 \tag{14.19}$$

The response $Y(z)$ is computed by multiplying the corresponding z-transforms

$$Y(z) = T(z) \cdot U(z). \tag{14.20}$$

For the current example this amounts to computing the coefficients of

$$Y(z) = y_0 + y_1 z + y_2 z^2 + y_3 z^3 + y_4 z^4 + y_5 z^5 + y_6 z^6. \tag{14.21}$$

The coefficients of the z-transform of $Y(z)$ appear to be identical to the previously discussed convolution of the coefficient vectors for $U(z)$ and $T(z)$.

$$Y(z) = (t_0 u_0) + (t_1 u_0 + t_0 u_1) z + (t_2 u_0 + t_1 u_1 + t_0 u_2) z^2 + (t_3 u_0 + t_2 u_1 + t_1 u_2 + t_0 u_3) z^3 + \dots$$

$$\dots + (h_3 u_1 + h_2 u_2 + h_1 u_3) z^4 + (h_3 u_2 + h_2 u_3) z^5 + (h_3 u_3) z^6$$

Note that the use of the z-transform is confined to linear <u>time-invariant</u> systems. Once we deal with time-varying systems the Fourier transformation or the z-transformation cannot be used anymore, as the convolution operator will not have a Toeplitz structure anymore.

14.10 Diagonal expansion of Toeplitz matrices

For a system to be truly time-invariant, the impulse response is invariant for all times, i.e. the corresponding Toeplitz operator describing the system must be infinite dimensional, the convolution operation then looks like

$$y = \mathbf{T} \cdot u = \begin{bmatrix} \vdots \\ y_{-1} \\ \boxed{y_0} \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ \vdots \end{bmatrix} = \begin{bmatrix} \ddots & \vdots & \vdots & \vdots & \vdots & \\ \ddots & t_{-1} & t_{-2} & t_{-3} & t_{-4} & \\ \ddots & t_0 & t_{-1} & t_{-2} & t_{-3} & \ddots \\ \ddots & t_1 & \boxed{t_0} & t_{-1} & t_{-2} & \ddots \\ \ddots & t_2 & t_1 & t_0 & t_{-1} & \ddots \\ & t_3 & t_2 & t_1 & t_0 & \ddots \\ & t_4 & t_3 & t_2 & t_1 & \ddots \\ & \vdots & \vdots & \ddots & \ddots & \ddots \end{bmatrix} \cdot \begin{bmatrix} \vdots \\ u_{-1} \\ \boxed{u_0} \\ u_1 \\ u_2 \\ \vdots \end{bmatrix} \qquad (14.22)$$

in which we have to indicate the location of the elements with index 0 in the vectors or the element with indices $(0, 0)$ in the matrices, since it is not clear anymore from the notation where these elements are located. We are looking for a more compact notation to exploit the structure of the infinite dimensional Toeplitz matrix. To this end we now introduce a more general 'shift operator'.

14.10.1 The shift operator

The Toeplitz matrices in equation (14.22) have a very particular structure, that is, every column of the matrix is a down-shifted version of the previous column. We intend to exploit this shift-structure to establish a notation and a formalism, which can extend from linear time-invariant to time-variant systems. To this end we introduce a general *shift operator*, which we denote by the symbol Z. The shift operator has the function to shift all entries of a column vector downward by one position. We can give this forward shift in time or 'causal' shift the interpretation of a temporal delay, that is,

$$(Zu)_k = u_{k-1}, \quad k = -\infty, \ldots, \infty.$$

For our matrix-oriented notation we will use a matrix representation for the shift operator, which acts from the left on a given vector u and pushes all its entries down by one notch

$$Zu = Z\begin{bmatrix} \vdots \\ u_{-1} \\ \boxed{u_0} \\ u_1 \\ u_2 \\ \vdots \end{bmatrix} \mapsto \begin{bmatrix} \vdots \\ u_{-2} \\ \boxed{u_{-1}} \\ u_0 \\ u_1 \\ \vdots \end{bmatrix}, \qquad Z^{-1}u = Z^{-1}\begin{bmatrix} \vdots \\ u_{-1} \\ \boxed{u_0} \\ u_1 \\ u_2 \\ \vdots \end{bmatrix} \mapsto \begin{bmatrix} \vdots \\ u_0 \\ \boxed{u_1} \\ u_2 \\ u_3 \\ \vdots \end{bmatrix}. \tag{14.23}$$

The rectangular box indicates the reference of time origin. Likewise, the inverse also holds, that is, Z^{-1} acting on the vector u pushes the entries of the vector up by one notch. It is worth noting that at this point we assume time series to run uniformly from index $-\infty$ to $+\infty$ (we shall soon modify this convention) and that on such series the shift operator Z is in a generalized sense orthogonal, i.e. $Z'Z = ZZ' = I$. Shifts on infinite dimensional vectors do not truncate the vectors (as is often assumed with shifts on finite vectors). The matrix Z itself can actually be represented by an infinite dimensional lower (causal) matrix

$$Z = \begin{bmatrix} \ddots & & & & & \\ \ddots & 0 & & & & \\ & 1 & 0 & & & \\ & & 1 & \boxed{0} & & \\ & & & 1 & 0 & \\ & & & & \ddots & \ddots \end{bmatrix}, \qquad Z^{-1} = Z' = \begin{bmatrix} \ddots & \ddots & & & & \\ & 0 & 1 & & & \\ & & 0 & 1 & & \\ & & & \boxed{0} & 1 & \\ & & & & 0 & \ddots \\ & & & & & \ddots \end{bmatrix}.$$

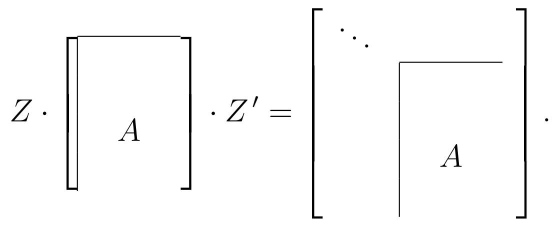
Similarly, the shift operator and its inverse can also be applied to a row vector from the right $u'Z$

$$\begin{bmatrix} \dots & u_{-1} & \boxed{u_0} & u_1 & u_2 & \dots \end{bmatrix} \cdot Z = \begin{bmatrix} \dots & u_{-1} & u_0 & \boxed{u_1} & u_2 & \dots \end{bmatrix}.$$

Accordingly, a shift by one position to the right is achieved by post-multiplication of the row vector $\mathbf{u}'$ with the inverse shift operator, that is, we then have

$$\begin{bmatrix} \dots & u_{-1} & \boxed{u_0} & u_1 & u_2 & \dots \end{bmatrix} \cdot Z' = \begin{bmatrix} \dots & u_{-2} & \boxed{u_{-1}} & u_0 & u_1 & \dots \end{bmatrix}$$

We can also apply the shift operator simultaneously from the left and from the right to a matrix A such as ZAZ', which has the effect to push the entries of A down one slot along the diagonal, which looks like



We can push down the matrix along its diagonal by k slots if we apply the shift operators from the left and from the right k times $Z^k A (Z')^k$.

These are first steps towards using infinite dimensional vectors and matrices, but at this point a word of caution is necessary. The notion of 'orthogonality' is only properly definite on finite vector spaces, thanks to the fact that a finite dimensional vector always has a finite quadratic norm. The same is not true with infinitely indexed vectors. To properly define orthogonality (and inner products etc...) in such a context, one would have to restrict the kind of vectors one may handle, so as to keep all vector-matrix multiplications and inner products under consideration finite. This will be a necessary central theme in our further developments, and we shall see that we can handle this issue nicely in most cases, but that is a concern for later.

14.10.2 Superposition of diagonals

Using the shift operator introduced in the previous section we can represent an infinite dimensional Toeplitz Operator (14.22) as the superposition of diagonals, such as

$$T \;= \cdots + \begin{bmatrix} \ddots & \ddots & & & & \\ & \ddots & t_{-1} & & & \\ & & \boxed{0} & t_{-1} & & \\ & & & 0 & t_{-1} & \\ & & & & \ddots & \ddots \\ & & & & & \ddots \end{bmatrix} + \begin{bmatrix} \ddots & \ddots & & & & \\ \ddots & t_0 & 0 & & & \\ & 0 & \boxed{t_0} & 0 & & \\ & & 0 & t_0 & 0 & \\ & & & 0 & t_0 & \ddots \\ & & & & \ddots & \ddots \end{bmatrix}$$

$$+ \begin{bmatrix} \ddots & & & & & \\ \ddots & \ddots & & & & \\ & t_1 & 0 & & & \\ & & t_1 & \boxed{0} & & \\ & & & t_1 & 0 & \\ & & & & \ddots & \ddots \end{bmatrix} + \ldots$$

This principle can be modified by denoting the values $t_i, i = -\infty, \infty$ along the main diagonal and providing the additional information by how many sub-diagonals this diagonal needs to be pushed down or pushed up. Pushing down by k diagonals is represented by the k-th power of the shift operator Z, pushing up diagonals is accomplished be negative powers of Z. This amounts to writing the Toeplitz operator as

$$T \;= \cdots + Z^{-1} \cdot \begin{bmatrix} \ddots & & & & \\ & t_{-1} & & & \\ & & \boxed{t_{-1}} & & \\ & & & t_{-1} & \\ & & & & \ddots \end{bmatrix} + \begin{bmatrix} \ddots & & & & \\ & t_0 & & & \\ & & \boxed{t_0} & & \\ & & & t_0 & \\ & & & & \ddots \end{bmatrix}$$

$$+Z \cdot \begin{bmatrix} \ddots & & & & \\ & t_1 & & & \\ & & \boxed{t_1} & & \\ & & & t_1 & \\ & & & & \ddots \end{bmatrix} + \ldots$$

$$= \cdots + Z^{-1}\text{diag}[t_{-1}] + Z^0\text{diag}[t_0] + Z^1\text{diag}[t_1] + Z^2\text{diag}[t_2] + Z^3\text{diag}[t_3] + \ldots$$
$$= \textstyle\sum_{i=-\infty}^{\infty} Z^i \cdot \text{diag}[t_i] = \sum_{i=-\infty}^{\infty} \text{diag}[t_i] \cdot Z^i.$$

(the last equality because Toeplitz operators commute with the Z as defined here). This notation reminds the reader of the z-transform of the time series $\{t_i\}, -\infty < i < \infty$,

$$T(z) = \sum_{i=-\infty}^{\infty} t_i z^i,$$

which can be thought of as an efficient manipulation tool when working with diagonals of infinite dimensional Toeplitz-Operators. There exists an 1:1 relationship between the set of power series and the set of infinite dimensional Toeplitz-Operators. The validity of this mechanism relies on the fact that the values along a diagonal are all the same, which is a consequence of dealing with time-invariant systems. Also note that in the context of the z-transformation the variable z is considered to be a complex variable, whereas the short operator is a real valued, infinite dimensional matrix.

14.11 Causality

We consider the linear, time-invariant system, which is given in terms of the Toeplitz $\mathbf{T}$ matrix. We feed the system with the input signal $\mathbf{u}$, which starts with the value u_0.

$$y = T \cdot u = \begin{bmatrix} \vdots \\ y_{-1} \\ \boxed{y_0} \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ \vdots \end{bmatrix} = \begin{bmatrix} \ddots & \vdots & \vdots & \vdots & \vdots & \\ \ddots & t_{-1} & t_{-2} & t_{-3} & t_{-4} & \\ \ddots & \boxed{t_0} & t_{-1} & t_{-2} & t_{-3} & \ddots \\ \ddots & t_1 & t_0 & t_{-1} & t_{-2} & \ddots \\ \ddots & t_2 & t_1 & t_0 & t_{-1} & \ddots \\ & t_3 & t_2 & t_1 & t_0 & \ddots \\ & t_4 & t_3 & t_2 & t_1 & \ddots \\ & \vdots & \vdots & \ddots & \ddots & \ddots \end{bmatrix} \cdot \begin{bmatrix} \vdots \\ 0 \\ \boxed{u_0} \\ u_1 \\ u_2 \\ \vdots \end{bmatrix}.$$

We can see that the system produces a valid output signal

$$y_{-1} = t_{-1}u_0 + t_{-2}u_1 + t_{-3}u_2 + \ldots,$$

using values that occur in time even before the actual input signal: it behaves in a non-causal way. For a linear system to behave (strictly) causally, the corresponding Toeplitz operator has to be (strictly) lower triangular.

The set of all causal systems corresponds with the set of all lower triangular matrices and is denoted by the symbol $\mathcal{L}$. The set of all upper triangular matrices, which we denote by the symbol $\mathcal{U}$, corresponds with the set of anti-causal systems. We use the symbol $\mathcal{D}$ to denote the set of main diagonal matrices, i.e.

$$\mathcal{D} = \mathcal{U} \cap \mathcal{L}.$$

In Equation 14.24 we see a lower triangular Toeplitz matrix $\mathbf{T}_{\mathcal{L}}$ describing a causal system

$$y = T_{\mathcal{L}} \cdot u = \begin{bmatrix} \vdots \\ 0 \\ \boxed{y_0} \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ \vdots \end{bmatrix} = \begin{bmatrix} \ddots & \vdots & \vdots & \vdots & \vdots & \\ \ddots & 0 & 0 & 0 & 0 & \\ \ddots & \boxed{t_0} & 0 & 0 & 0 & \ddots \\ \ddots & t_1 & t_0 & 0 & 0 & \ddots \\ \ddots & t_2 & t_1 & t_0 & 0 & \ddots \\ & t_3 & t_2 & t_1 & t_0 & \ddots \\ & t_4 & t_3 & t_2 & t_1 & \ddots \\ & \vdots & \vdots & \ddots & \ddots & \ddots \end{bmatrix} \cdot \begin{bmatrix} \vdots \\ 0 \\ \boxed{u_0} \\ u_1 \\ u_2 \\ u_3 \\ \vdots \end{bmatrix}. \quad (14.24)$$

Using the shift operator and the diagonal expansion discussed in the previous section let us have a look at the following example for a simple FIR filter

$$T = \frac{1}{2}Z^{-1} - I + \frac{1}{2}Z \qquad \Leftrightarrow \qquad \mathbf{T} = \begin{bmatrix} \ddots & \ddots & & & & \\ \ddots & -1 & 1/2 & & & \\ & 1/2 & \boxed{-1} & 1/2 & & \\ & & 1/2 & -1 & 1/2 & \\ & & & -1/2 & -1 & \ddots \\ & & & & \ddots & \ddots \end{bmatrix}.$$

This tridiagonal system is non-causal, as it is able to produce output signals that lie temporally before the input signals. Take the causal part of this simple FIR filter, which is the lower triangular part of the tridiagonal matrix

$$T_{\mathcal{L}} = \frac{1}{2}Z - I \qquad \Leftrightarrow \qquad T_{\mathcal{L}} = \begin{bmatrix} \ddots & & & & & \\ \ddots & -1 & & & & \\ & 1/2 & \boxed{-1} & & & \\ & & 1/2 & -1 & & \\ & & & 1/2 & -1 & \\ & & & & \ddots & \ddots \end{bmatrix}.$$

The strictly anti-causal part of the previously introduced FIR filter is described in terms of the upper triangular part of the tridiagonal matrix $\mathbf{T}$, i.e. by

$$T_{Z^{-1}\mathcal{U}} = \frac{1}{2}Z^{-1} \qquad \Leftrightarrow \qquad T_{Z^{-1}\mathcal{U}} = \begin{bmatrix} \ddots & & & & & \\ \ddots & 0 & 1/2 & & & \\ & & \boxed{0} & 1/2 & & \\ & & & 0 & 1/2 & \\ & & & & 0 & \\ & & & & \ddots & \ddots \end{bmatrix},$$

where $Z^{-1}\mathcal{U}$ denotes the strictly upper triangular part of a matrix and hence the strictly anti-causal systems. We can identify the anti-causal parts of the system to be associated with negative exponents of the shift operator. (One does not write T as a 'function' of Z anymore, as Z does not take numerical values and properly represents a matrix in its own right.)

14.12 Discussion items

- *Affine systems:* strictly speaking, a system that contains a constant offset is not a linear system. For example: the system defined by the moving average equation $y_n = a_0u_n + a_1u_{n-1} + d_0$ is not a linear system according to the usual (and our) definition. Linearity can be saved by writing $y_n = a_0u_n + a_1u_{n-1} + d_0v_n$ and taking for v_n the constant value 1. The system now has two input variables, which have to be handled separately, and the linear theory can be applied, but not without some consequences requiring extra care! E.g., when would such a system be 'stable'?

- The discrete Fourier transformation matrix $\mathbf{Q}$ has an interesting structure, in particular when the 'modulo' relation $q^N = 1$ is used. For example, when N is the product of two prime numbers, $N = pq$ with p and q prime, then the matrix decomposes in a kind of 'tensor product', very much related to the FFT. Consider e.g., the cases $N = 2$, $N = 4$, $N = 3*4$. The decomposition strategy can be repeated for more general products of primes. All this produces interesting examples of orthogonal matrices!

- In the transfer function literature, one distinguishes special cases of transfer functions: *moving average* and *autoregressive*. In the first case, the output is an average of past input signals and the transfer function becomes polynomial in the shift z. In the second, it becomes the inverse of a polynomial in z. The more general scalar case is often called *iir* for *infinite impulse response*. It can of course be considered autoregressive in the moving average given by the numerator coefficient!

- Much of the theory on transfer functions and convolution generalizes to systems with multiple inputs/outputs. One then uses matrices instead of scalars, but the basic definitions: linearity, shift invariance and causality remain the same.

- A finite Toeplitz matrix does not simply commute with the normal shift. One way out has been used in this chapter: make things cyclic; but the price to be paid is hefty: one has to use circulant matrices, which become unwieldy in the time-invariant case. The case of eigenspaces for Toeplitz matrices is considerably more complicated [16, 37]. Results for finite Toeplitz matrices cannot easily be extended to infinite matrices.

14.13 Notes

- The elementary theory presented in this chapter is of course hyper-traditional for anyone who has studied electrical engineering. It is given here because (1) it may not be familiar to students from other directions, in particular numerical mathematics, and (2) the presentation is done in such a way that the non time-invariant definitions given in the following chapters appear to be natural extensions. In particular, the various operators acting on signals representing systems (the shift, the transfer function, convolution) have natural matrix interpretations, which in the way the material is treated classically often disappears behind a curtain of transforms. In the time-variant theory these will play an essential role, so much that our 'modern' treatment makes the transform calculus (almost) obsolete. Nonetheless, time-invariance remains a very important special case, with its opportunities and difficulties. To start with the latter: time-invariance often requires the solution of a 'fixed point problem', to keep whatever solution found for a given problem time-invariant. In particular, this is the case when one has to solve Lyapunov-Stein or Riccati equations, e.g., in optimization or control problems. The opportunities are mostly due to enhanced numerical efficiency. The FFT plays a very big role in achieving this. In the literature, compromises have developed, which we shall encounter in various guises: semi-separability (equivalent to finite state time-variant) or 'close to time-invariance' as developed by Kailath and his group [14, 23], whereby other structural properties of transfer operators are exploited beside pure time-invariance.

- Strictly speaking, only a doubly infinitely indexed Toeplitz matrix may be called 'time-invariant'. Although this becomes, by necessity, an operator working on an infinite dimensional space (with all the problems this brings), one may still observe that such an operator commutes with the shift, and hence, it will share full eigenspaces with the shift. Such a theory, generalization of the eigenspace theory we used in this chapter, quickly runs into difficulties, because eigenfunctions of the shift may not belong to the original space of definition. For example the discrete-time function u defined by $u_k = e^{ka}$ with a some complex number is technically an eigenfunction because $zu = e^{-a}u$ and has eigenvalue e^{-a}, but it is unbounded except when a is purely imaginary, and even in that case it has infinite quadratic norm. Nonetheless, one can develop a discrete-time Fourier theory for such systems and most of the property will go through provided the functions are handled with some care. In particular, Fourier transform commutes convolution into multiplication of the transforms, which is one of the main advantages, not only mathematically but also physically. The same happens in the time-continuous case and has lead to the extensive use of the spectrum as a medium on which many 'channels' can be shared, each one restricted to a specific frequency domain. Much of the needed mathematical development is called 'harmonic analysis' and requires not-so-elementary mathematics, for which there are excellent textbooks [31, 20]. We shall see in the following chapters that most of the basic system theoretical problems (for estimation, control or system inversion) can be treated without recourse to transform theory. In most chapters we make efforts to extent the discrete-time, time-variant results to the time-invariant and/or continuous-time case.

Index

Bibliography

[1] B.D.O. Anderson and J.B. Moore. New results in linear system stability. *SIAM J. Control*, 7(3):398–414, August 1969.

[2] B.D.O. Anderson and J.B. Moore. *Optimal Filtering*. Prentice Hall, 1979.

[3] J. Anneveling and P. Dewilde. Object-oriented data management based on abstract data types. *Software Practice and Experience*, 17 (11), Nov. 1987.

[4] W. Arveson. Interpolation problems in nest algebras. *J. Functional Anal.*, 20:208–233, 1975.

[5] J.A. Ball and J.W. Helton. Inner-outer factorization of nonlinear operators. *J. Funct. Anal.*, 104:363–413, 1992.

[6] J.A. Ball and J.W. Helton. Inne-outer factorization of nonlinear operators. *Journal of Functional Analysis*, Vol. 104, No. 2:363–413, March 1992.

[7] R. Bellman. The theory of dynamic programming. *Bull. Am. Math. Soc.*, Vol. 60:503–516, 1954.

[8] George Cybenko. A general orthogonalization technique with applications to time series analysis and signal processing. *Mathematics of Computation*, 40(161):323–336, 1983.

[9] E. Deprettere. Mixed-form time-variant lattice recursions. In *Outils et Modèles Mathématiques pour l'Automatique, l'Analyse de Systèmes et le Traitement du Signal*, Paris, 1981. CNRS.

[10] P. Dewilde and A.-J. van der Veen. *Time-varying Systems and Computations*. Kluwer, out of print but freely available at ens.ewi.tudelft.nl, 1998.

[11] P.M. Dewilde. A course on the algebraic Schur and Nevanlinna-Pick interpolation problems. In Ed. F. Deprettere and A.J. van der Veen, editors, *Algorithms and Parallel VLSI Architectures*. Elsevier, 1991.

[12] P. Van Dooren. A unitary method for deadbeat control. *Proceedings MTNS*, 1983.

[13] Lawrence C. Evans. *An Introduction to Mathematical Control Theory Version 0.2*. Published on the web.

[14] B. Friedlander, M. Morf, T. Kailath, and L. Ljung. New inversion formulas for matrices classified in terms of their distance from Toeplitz matrices. *Lin. Alg. Appl.*, 23:31–60, 1979.

[15] G.H. Golub and Ch.F. Van Loan. *Matrix Computations*. The John Hopkins University Press, Baltimore, Maryland, 1983.

[16] U. Grenander and G. Szegö. *Toeplitz forms and their Applications*. Univ. of Cal. Press, 1958.

[17] P.R. Halmos. *Naive Set Theory*. Van Nostrand, 1960.

[18] L.M.J. Hautus. A simple proof of heymann's lemma. *IEEE Trans. on Automatic Control*, 22, no. 5:885–886, 1977.

[19] B.L. Ho and R.E. Kalman. Effective construction of linear, state-variable models from input/output functions. *Regelungstechnik*, 14:545–548, 1966.

[20] K. Hoffman. *Banach Spaces of Analytic Functions*. Prentice-Hall, Englewood Cliffs, NJ, 1962.

[21] T. Kailath. *Lectures on Linear Least-Squares Estimation*. Springer Verlag, CISM Courses and Lectures No. 140, Wien, New York, 1976.

[22] T. Kailath. A view of three decades of linear filtering theory. *IEEE Trans. on Information Theory*, 20(2):145–181, March 1974.

[23] T. Kailath, S.Y. Kung, and M. Morf. Displacement ranks of matrices and linear equations. *J. Math. Anal. Appl.*, 68(2):395–407, 1979.

[24] Thomas Kailath, Ali H. Sayed, and Babak Hassibi. *Linear Estimation*. Prentice Hall, 2000.

[25] R.E. Kalman. A new approach to linear filtering and prediction problems. *J. Basic Engineering*, 82:34–45, March 1960.

[26] R.E. Kalman, P.L. Falb, and M.A. Arbib. *Topics in Mathematical System Theory*. Int. Series in Pure and Applied Math. McGraw-Hill, 1970.

[27] David G. Luenberger. *Introduction to Dynamic Systems*. John Wiley & Sons, 1979.

[28] M. Morf and T. Kailath. Square-root algorithms for Least-Squares Estimation. *IEEE Trans. Automat. Control*, 20(4):487–497, 1975.

[29] J. K. Rice and M. Verhaegen. Distributed control: A sequentially semi-separable approach for spatially heterogeneous linear sysems. *IEEE Transactions on Automatic Control*, 54:1270–1284, 2009.

[30] John.R. Ringrose. On some algebras of operators. *Proceedings of the London Mathematical Society*, Third Series, 15, 1965.

[31] W. Rudin. *Real and Complex Analysis*. McGraw-Hill, New York, 1966.

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

[33] L.M. Silverman and H.E. Meadows. Equivalence and synthesis of time-variable linear systems. In *Proc. 4-th Allerton Conf. Circuit and Systems Theory*, pages 776–784, 1966.

[34] G.W. Stewart. *Matrix Algorithms, Vol. I: Basic Decompositions*. SIAM, 1998.

[35] G.W. Stewart. *Matrix Algorithms, Vol. II: Eigensystems*. SIAM, 2001.

[36] G. Strang. *Introduction to linear algebra*. Wellesley-Cambridge Press, Wellesley, MA, 1993.

[37] W.F. Trench. On the eigenvalue problem for toeplitz band matrices. *Linear Algebra and its Applications*, 64:199–214, 1985.

[38] Arjan van der Schacht. *L_2-Gain and Passivity Techniques in Nonlinear Control*. Springer, 2000.

[39] A.J. van der Veen. *Time-Varying System Theory and Computational Modeling: Realization, Approximation, and Factorization*. PhD thesis, Delft University of Technology, Delft, The Netherlands, June 1993.

[40] M. Verhaegen. Subspace model identification. part III: Analysis of the ordinary output-error state space model identification algorithm. *Int. J. Control*, 58:555–586, 1994.

[41] M. Verhaegen and P. Dewilde. Subspace model identification. part I: The output-error state space model identification class of algorithms. *Int. J. Control*, 56(5):1187–1210, 1992.

[42] M. Verhaegen and P. Dewilde. Subspace model identification. part II: Analysis of the elementary output-error state space model identification algorithm. *Int. J. Control*, 56(5):1211–1241, 1992.

[43] J.H. Wilkinson and C. Reinsch. *Linear Algebra, vol.2, Handbook for Automatic Computation*. Springer, 1971.

[44] Jan Willems. The behavioral approach to open and interconnected systems. *IEEE Control Systems Magazine*, 10.1109/MCS.2007.906923(December):46–99, 2007.