Springer Texts in Statistics

James E. Gentle

Matrix Algebra

Theory, Computations and Applications in Statistics

Second Edition

Springer

Springer Texts in Statistics

Series Editors

Richard DeVeaux
Stephen E. Fienberg
Ingram Olkin

More information about this series at http://www.springer.com/series/417

James E. Gentle

Matrix Algebra

Theory, Computations and Applications in Statistics

Second Edition

Springer

James E. Gentle
Fairfax, VA, USA

ISSN 1431-875X ISSN 2197-4136 (electronic)
Springer Texts in Statistics
ISBN 978-3-319-64866-8 ISBN 978-3-319-64867-5 (eBook)
DOI 10.1007/978-3-319-64867-5

Library of Congress Control Number: 2017952371

This Springer imprint is published by Springer Nature
The registered company is Springer International Publishing AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

To María

Preface to the Second Edition

In this second edition, I have corrected all known typos and other errors; I have (it is hoped) clarified certain passages; I have added some additional material; and I have enhanced the Index.

I have added a few more comments about vectors and matrices with complex elements, although, as before, unless stated otherwise, all vectors and matrices in this book are assumed to have real elements. I have begun to use "$\det(A)$" rather than "$|A|$" to represent the determinant of A, except in a few cases. I have also expressed some derivatives as the transposes of the expressions I used formerly.

I have put more conscious emphasis on "user-friendliness" in this edition. In a book, user-friendliness is primarily a function of references, both internal and external, and of the index. As an old software designer, I've always thought that user-friendliness is very important. To the extent that internal references were present in the first edition, the positive feedback I received from users of that edition about the friendliness of those internal references ("I liked the fact that you said 'equation (x.xx) on page yy,' instead of just 'equation (x.xx)' ") encouraged me to try to make the internal references even more useful. It's only when you're "eating your own dog food," that you become aware of where details matter, and in using the first edition, I realized that the choice of entries in the Index was suboptimal. I have spent significant time in organizing it, and I hope that the user will find the Index to this edition to be very useful. I think that it has been vastly improved over the Index in the first edition.

The overall organization of chapters has been preserved, but some sections have been changed. The two chapters that have been changed most are Chaps. 3 and 12. Chapter 3, on the basics of matrices, got about 30 pages longer. It is by far the longest chapter in the book, but I just didn't see any reasonable way to break it up. In Chap. 12 of the first edition, "Software for Numerical Linear Algebra," I discussed four software systems or languages, C/C++, Fortran, Matlab, and R, and did not express any preference for one

over another. In this edition, although I occasionally mention various languages and systems, I now limit most of my discussion to Fortran and R.

There are many reasons for my preference for these two systems. R is oriented toward statistical applications. It is open source and freely distributed. As for Fortran versus C/C++, Python, or other programming languages, I agree with the statement by Hanson and Hopkins (2013, page ix), "... Fortran is currently the best computer language for numerical software." Many people, however, still think of Fortran as the language their elders (or they themselves) used in the 1970s. (On a personal note, Richard Hanson, who passed away recently, was a member of my team that designed the IMSL C Libraries in the mid 1980s. Not only was C much cooler than Fortran at the time, but the ANSI committee working on updating the Fortran language was so fractured by competing interests that approval of the revision was repeatedly delayed. Many numerical analysts who were not concerned with coolness turned to C because it provided dynamic storage allocation and it allowed flexible argument lists, and the Fortran constructs could not be agreed upon.)

Language preferences are personal, of course, and there is a strong "coolness factor" in choice of a language. Python is currently one of the coolest languages, but I personally don't like the language for most of the stuff I do.

Although this book has separate parts on applications in statistics and computational issues as before, statistical applications have informed the choices I made throughout the book, and computational considerations have given direction to most discussions.

I thank the readers of the first edition who informed me of errors. Two people in particular made several meaningful comments and suggestions. Clark Fitzgerald not only identified several typos, he made several broad suggestions about organization and coverage that resulted in an improved text (I think). Andreas Eckner found, in addition to typos, some gaps in my logic and also suggested better lines of reasoning at some places. (Although I don't follow an itemized "theorem-proof" format, I try to give reasons for any nonobvious statements I make.) I thank Clark and Andreas especially for their comments. Any remaining typos, omissions, gaps in logic, and so on are entirely my responsibility.

Again, I thank my wife, María, to whom this book is dedicated, for everything.

I used TEX via LATEX 2$_\varepsilon$ to write the book. I did all of the typing, programming, etc., myself, so all misteaks (mistakes!) are mine. I would appreciate receiving suggestions for improvement and notification of errors. Notes on this book, including errata, are available at

`http://mason.gmu.edu/~jgentle/books/matbk/`

Fairfax County, VA, USA James E. Gentle
July 14, 2017

Preface to the First Edition

I began this book as an update of *Numerical Linear Algebra for Applications in Statistics*, published by Springer in 1998. There was a modest amount of new material to add, but I also wanted to supply more of the reasoning behind the facts about vectors and matrices. I had used material from that text in some courses, and I had spent a considerable amount of class time proving assertions made but not proved in that book. As I embarked on this project, the character of the book began to change markedly. In the previous book, I apologized for spending 30 pages on the theory and basic facts of linear algebra before getting on to the main interest: *numerical* linear algebra. In this book, discussion of those basic facts takes up over half of the book.

The orientation and perspective of this book remains *numerical linear algebra for applications in statistics*. Computational considerations inform the narrative. There is an emphasis on the areas of matrix analysis that are important for statisticians, and the kinds of matrices encountered in statistical applications receive special attention.

This book is divided into three parts plus a set of appendices. The three parts correspond generally to the three areas of the book's subtitle—theory, computations, and applications—although the parts are in a different order, and there is no firm separation of the topics.

Part I, consisting of Chaps. 1 through 7, covers most of the material in linear algebra needed by statisticians. (The word "matrix" in the title of this book may suggest a somewhat more limited domain than "linear algebra"; but I use the former term only because it seems to be more commonly used by statisticians and is used more or less synonymously with the latter term.)

The first four chapters cover the basics of vectors and matrices, concentrating on topics that are particularly relevant for statistical applications. In Chap. 4, it is assumed that the reader is generally familiar with the basics of partial differentiation of scalar functions. Chapters 5 through 7 begin to take on more of an applications flavor, as well as beginning to give more consideration to computational methods. Although the details of the computations

are not covered in those chapters, the topics addressed are oriented more toward computational algorithms. Chapter 5 covers methods for decomposing matrices into useful factors.

Chapter 6 addresses applications of matrices in setting up and solving linear systems, including overdetermined systems. We should not confuse statistical inference with fitting equations to data, although the latter task is a component of the former activity. In Chap. 6, we address the more mechanical aspects of the problem of fitting equations to data. Applications in statistical data analysis are discussed in Chap. 9. In those applications, we need to make statements (i.e., assumptions) about relevant probability distributions.

Chapter 7 discusses methods for extracting eigenvalues and eigenvectors. There are many important details of algorithms for eigenanalysis, but they are beyond the scope of this book. As with other chapters in Part I, Chap. 7 makes some reference to statistical applications, but it focuses on the mathematical and mechanical aspects of the problem.

Although the first part is on "theory," the presentation is informal; neither definitions nor facts are highlighted by such words as "definition," "theorem," "lemma," and so forth. It is assumed that the reader follows the natural development. Most of the facts have simple proofs, and most proofs are given naturally in the text. No "Proof" and "Q.E.D." or "■" appear to indicate beginning and end; again, it is assumed that the reader is engaged in the development. For example, on page 341:

> If A is nonsingular and symmetric, then A^{-1} is also symmetric because $(A^{-1})^{\mathrm{T}} = (A^{\mathrm{T}})^{-1} = A^{-1}$.

The first part of that sentence could have been stated as a theorem and given a number, and the last part of the sentence could have been introduced as the proof, with reference to some previous theorem that the inverse and transposition operations can be interchanged. (This had already been shown before page 341—in an unnumbered theorem of course!)

None of the proofs are original (at least, I don't think they are), but in most cases, I do not know the original source or even the source where I first saw them. I would guess that many go back to C. F. Gauss. Most, whether they are as old as Gauss or not, have appeared somewhere in the work of C. R. Rao. Some lengthier proofs are only given in outline, but references are given for the details. Very useful sources of details of the proofs are Harville (1997), especially for facts relating to applications in linear models, and Horn and Johnson (1991), for more general topics, especially those relating to stochastic matrices. The older books by Gantmacher (1959) provide extensive coverage and often rather novel proofs. These two volumes have been brought back into print by the American Mathematical Society.

I also sometimes make simple assumptions without stating them explicitly. For example, I may write "for all i" when i is used as an index to a vector. I hope it is clear that "for all i" means only "for i that correspond to indices

of the vector." Also, my use of an expression generally implies existence. For example, if "AB" is used to represent a matrix product, it implies that "A and B are conformable for the multiplication AB." Occasionally, I remind the reader that I am taking such shortcuts.

The material in Part I, as in the entire book, was built up recursively. In the first pass, I began with some definitions and followed those with some facts that are useful in applications. In the second pass, I went back and added definitions and additional facts that led to the results stated in the first pass. The supporting material was added as close to the point where it was needed as practical and as necessary to form a logical flow. Facts motivated by additional applications were also included in the second pass. In subsequent passes, I continued to add supporting material as necessary and to address the linear algebra for additional areas of application. I sought a bare-bones presentation that gets across what I considered to be the theory necessary for most applications in the data sciences. The material chosen for inclusion is motivated by applications.

Throughout the book, some attention is given to numerical methods for computing the various quantities discussed. This is in keeping with my belief that statistical computing should be dispersed throughout the statistics curriculum and statistical literature generally. Thus, unlike in other books on matrix "theory," I describe the "modified" Gram-Schmidt method, rather than just the "classical" GS. (I put "modified" and "classical" in quotes because, to me, GS *is* MGS. History is interesting, but in computational matters, I do not care to dwell on the methods of the past.) Also, condition numbers of matrices are introduced in the "theory" part of the book, rather than just in the "computational" part. Condition numbers also relate to fundamental properties of the model and the data.

The difference between an expression and a computing method is emphasized. For example, often we may write the solution to the linear system $Ax = b$ as $A^{-1}b$. Although this is the solution (so long as A is square and of full rank), solving the linear system does not involve computing A^{-1}. We may write $A^{-1}b$, but we know we can compute the solution without inverting the matrix.

> "This is an instance of a principle that we will encounter repeatedly: *the form of a mathematical expression and the way the expression should be evaluated in actual practice may be quite different.*"

(The statement in quotes appears word for word in several places in the book.)

Standard textbooks on "matrices for statistical applications" emphasize their uses in the analysis of traditional linear models. This is a large and important field in which real matrices are of interest, and the important kinds of real matrices include symmetric, positive definite, projection, and generalized inverse matrices. This area of application also motivates much of the discussion in this book. In other areas of statistics, however, there are different matrices

of interest, including similarity and dissimilarity matrices, stochastic matrices, rotation matrices, and matrices arising from graph-theoretic approaches to data analysis. These matrices have applications in clustering, data mining, stochastic processes, and graphics; therefore, I describe these matrices and their special properties. I also discuss the geometry of matrix algebra. This provides a better intuition of the operations. Homogeneous coordinates and special operations in $\mathbb{R}^3$ are covered because of their geometrical applications in statistical graphics.

Part II addresses selected applications in data analysis. Applications are referred to frequently in Part I, and of course, the choice of topics for coverage was motivated by applications. The difference in Part II is in its orientation.

Only "selected" applications in data analysis are addressed; there are applications of matrix algebra in almost all areas of statistics, including the theory of estimation, which is touched upon in Chap. 4 of Part I. Certain types of matrices are more common in statistics, and Chap. 8 discusses in more detail some of the important types of matrices that arise in data analysis and statistical modeling. Chapter 9 addresses selected applications in data analysis. The material of Chap. 9 has no obvious definition that could be covered in a single chapter (or a single part or even a single book), so I have chosen to discuss briefly a wide range of areas. Most of the sections and even subsections of Chap. 9 are on topics to which entire books are devoted; however, I do not believe that any single book addresses all of them.

Part III covers some of the important details of numerical computations, with an emphasis on those for linear algebra. I believe these topics constitute the most important material for an introductory course in numerical analysis for statisticians and should be covered in every such course.

Except for specific computational techniques for optimization, random number generation, and perhaps symbolic computation, Part III provides the basic material for a course in statistical computing. All statisticians should have a passing familiarity with the principles.

Chapter 10 provides some basic information on how data are stored and manipulated in a computer. Some of this material is rather tedious, but it is important to have a general understanding of computer arithmetic before considering computations for linear algebra. Some readers may skip or just skim Chap. 10, but the reader should be aware that the way the computer stores numbers and performs computations has far-reaching consequences. Computer arithmetic differs from ordinary arithmetic in many ways; for example, computer arithmetic lacks associativity of addition and multiplication, and series often converge even when they are not supposed to. (On the computer, a straightforward evaluation of $\sum_{x=1}^{\infty} x$ converges!)

I emphasize the differences between the abstract number system $\mathbb{R}$, called the reals, and the computer number system $\mathbb{F}$, the floating-point numbers unfortunately also often called "real." Table 10.4 on page 492 summarizes some of these differences. All statisticians should be aware of the effects of these differences. I also discuss the differences between $\mathbb{Z}$, the abstract number

system called the integers, and the computer number system $\mathbb{I}$, the fixed-point numbers. (Appendix A provides definitions for this and other notation that I use.)

Chapter 10 also covers some of the fundamentals of algorithms, such as iterations, recursion, and convergence. It also discusses software development. Software issues are revisited in Chap. 12.

While Chap. 10 deals with general issues in numerical analysis, Chap. 11 addresses specific issues in numerical methods for computations in linear algebra.

Chapter 12 provides a brief introduction to software available for computations with linear systems. Some specific systems mentioned include the IMSL™ libraries for Fortran and C, Octave or MATLAB® (or Matlab®), and R or S-PLUS® (or S-Plus®). All of these systems are easy to use, and the best way to learn them is to begin using them for simple problems. I do not use any particular software system in the book, but in some exercises, and particularly in Part III, I do assume the ability to program in either Fortran or C and the availability of either R or S-Plus, Octave or Matlab, and Maple® or Mathematica®. My own preferences for software systems are Fortran and R, and occasionally, these preferences manifest themselves in the text.

Appendix A collects the notation used in this book. It is generally "standard" notation, but one thing the reader must become accustomed to is the lack of notational distinction between a vector and a scalar. All vectors are "column" vectors, although I usually write them as horizontal lists of their elements. (Whether vectors are "row" vectors or "column" vectors is generally only relevant for how we write expressions involving vector/matrix multiplication or partitions of matrices.)

I write algorithms in various ways, sometimes in a form that looks similar to Fortran or C and sometimes as a list of numbered steps. I believe all of the descriptions used are straightforward and unambiguous.

This book could serve as a basic reference either for courses in statistical computing or for courses in linear models or multivariate analysis. When the book is used as a reference, rather than looking for "definition" or "theorem," the user should look for items set off with bullets or look for numbered equations or else should use the Index or Appendix A, beginning on page 589.

The prerequisites for this text are minimal. Obviously, some background in mathematics is necessary. Some background in statistics or data analysis and some level of scientific computer literacy are also required. References to rather advanced mathematical topics are made in a number of places in the text. To some extent, this is because many sections evolved from class notes that I developed for various courses that I have taught. All of these courses were at the graduate level in the computational and statistical sciences, but they have had wide ranges in mathematical level. I have carefully reread the sections that refer to groups, fields, measure theory, and so on and am convinced that if the reader does not know much about these topics, the material is still understandable but if the reader is familiar with these topics, the references

add to that reader's appreciation of the material. In many places, I refer to computer programming, and some of the exercises require some programming. A careful coverage of Part III requires background in numerical programming.

In regard to the use of the book as a text, most of the book evolved in one way or another for my own use in the classroom. I must quickly admit, however, that I have never used this whole book as a text for any single course. I have used Part III in the form of printed notes as the primary text for a course in the "foundations of computational science" taken by graduate students in the natural sciences (including a few statistics students, but dominated by physics students). I have provided several sections from Parts I and II in online PDF files as supplementary material for a two-semester course in mathematical statistics at the "baby measure theory" level (using Shao, 2003). Likewise, for my courses in computational statistics and statistical visualization, I have provided many sections, either as supplementary material or as the primary text, in online PDF files or printed notes. I have not taught a regular "applied statistics" course in almost 30 years, but if I did, I am sure that I would draw heavily from Parts I and II for courses in regression or multivariate analysis. If I ever taught a course in "matrices for statistics" (I don't even know if such courses exist), this book would be my primary text because I think it covers most of the things statisticians need to know about matrix theory and computations.

Some exercises are Monte Carlo studies. I do not discuss Monte Carlo methods in this text, so the reader lacking background in that area may need to consult another reference in order to work those exercises. The exercises should be considered an integral part of the book. For some exercises, the required software can be obtained from either `statlib` or `netlib` (see the bibliography). Exercises in any of the chapters, not just in Part III, may require computations or computer programming.

Penultimately, I must make some statement about the relationship of this book to some other books on similar topics. A much important statistical theory and many methods make use of matrix theory, and many statisticians have contributed to the advancement of matrix theory from its very early days. Widely used books with derivatives of the words "statistics" and "matrices/linearalgebra" in their titles include Basilevsky (1983), Graybill (1983), Harville (1997), Schott (2004), and Searle (1982). All of these are useful books. The computational orientation of this book is probably the main difference between it and these other books. Also, some of these other books only address topics of use in linear models, whereas this book also discusses matrices useful in graph theory, stochastic processes, and other areas of application. (If the applications are only in linear models, most matrices of interest are symmetric and all eigenvalues can be considered to be real.) Other differences among all of these books, of course, involve the authors' choices of secondary topics and the ordering of the presentation.

Fairfax County, VA, USA James E. Gentle

Acknowledgments

I thank John Kimmel of Springer for his encouragement and advice on this book and other books on which he has worked with me. I especially thank Ken Berk for his extensive and insightful comments on a draft of this book. I thank my student Li Li for reading through various drafts of some of the chapters and pointing out typos or making helpful suggestions. I thank the anonymous reviewers of this edition for their comments and suggestions. I also thank the many readers of my previous book on numerical linear algebra who informed me of errors and who otherwise provided comments or suggestions for improving the exposition. Whatever strengths this book may have can be attributed in large part to these people, named or otherwise. The weaknesses can only be attributed to my own ignorance or hardheadedness.

I thank my wife, María, to whom this book is dedicated, for everything.

I used TEX via LATEX 2ε to write the book. I did all of the typing, programming, etc., myself, so all misteaks are mine. I would appreciate receiving suggestions for improvement and notification of errors.

Fairfax County, VA, USA James E. Gentle
June 12, 2007

Contents

Author Biography

James E. Gentle, PhD, is University Professor of Computational Statistics at George Mason University. He is a Fellow of the American Statistical Association (ASA) and of the American Association for the Advancement of Science. Professor Gentle has held national offices in the ASA and has served as editor and associate editor of journals of the ASA as well as for other journals in statistics and computing. He is the author of "Random Number Generation and Monte Carlo Methods" (Springer, 2003) and "Computational Statistics" (Springer, 2009).

Part I

Linear Algebra

1

Basic Vector/Matrix Structure and Notation

Vectors and matrices are useful in representing multivariate numeric data, and they occur naturally in working with linear equations or when expressing linear relationships among objects. Numerical algorithms for a variety of tasks involve matrix and vector arithmetic. An optimization algorithm to find the minimum of a function, for example, may use a vector of first derivatives and a matrix of second derivatives; and a method to solve a differential equation may use a matrix with a few diagonals for computing differences.

There are various precise ways of defining vectors and matrices, but we will generally think of them merely as linear or rectangular arrays of numbers, or scalars, on which an algebra is defined. Unless otherwise stated, we will assume the scalars are real numbers. We denote both the set of real numbers and the field of real numbers as $\mathbb{R}$. (The *field* is the set together with the two operators.) Occasionally we will take a geometrical perspective for vectors and will consider matrices to define geometrical transformations. In all contexts, however, the elements of vectors or matrices are real numbers (or, more generally, members of a field). When the elements are not members of a field (names or characters, for example) we will use more general phrases, such as "ordered lists" or "arrays".

Many of the operations covered in the first few chapters, especially the transformations and factorizations in Chap. 5, are important because of their use in solving systems of linear equations, which will be discussed in Chap. 6; in computing eigenvectors, eigenvalues, and singular values, which will be discussed in Chap. 7; and in the applications in Chap. 9.

Throughout the first few chapters, we emphasize the facts that are important in statistical applications. We also occasionally refer to relevant computational issues, although computational details are addressed specifically in Part III.

J.E. Gentle, *Matrix Algebra*, Springer Texts in Statistics,
DOI 10.1007/978-3-319-64867-5_1

It is very important to understand that the form of a mathematical expression and the way the expression should be evaluated in actual practice may be quite different. We remind the reader of this fact from time to time. That there is a difference in mathematical expressions and computational methods is one of the main messages of Chaps. 10 and 11. (An example of this, in notation that we will introduce later, is the expression $A^{-1}b$. If our goal is to solve a linear system $Ax = b$, we probably should never compute the matrix inverse A^{-1} and then multiply it times b. Nevertheless, it may be entirely appropriate to write the expression $A^{-1}b$.)

1.1 Vectors

For a positive integer n, a vector (or n-vector) is an n-tuple, ordered (multi)set, or array of n numbers, called *elements* or *scalars*. The number of elements is called the *order*, or sometimes the "length", of the vector. An n-vector can be thought of as representing a point in n-dimensional space. In this setting, the "length" of the vector may also mean the Euclidean distance from the origin to the point represented by the vector; that is, the square root of the sum of the squares of the elements of the vector. This Euclidean distance will generally be what we mean when we refer to the *length* of a vector (see page 27). In general, "length" is measured by a *norm*; see Sect. 2.1.5, beginning on page 25.

We usually use a lowercase letter to represent a vector, and we use the same letter with a single subscript to represent an element of the vector.

The first element of an n-vector is the first (1^{st}) element and the last is the n^{th} element. (This statement is not a tautology; in some computer systems, the first element of an object used to represent a vector is the 0^{th} element of the object. This sometimes makes it difficult to preserve the relationship between the computer entity and the object that is of interest.) Although we are very concerned about computational issues, we will use paradigms and notation that maintain the priority of the object of interest rather than the computer entity representing it.

We may write the n-vector x as

$$x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \tag{1.1}$$

or

$$x = (x_1, \ldots, x_n). \tag{1.2}$$

We make no distinction between these two notations, although in some contexts we think of a vector as a "column", so the first notation may be more natural. The simplicity of the second notation recommends it for common use.

(And this notation does not require the additional symbol for transposition that some people use when they write the elements of a vector horizontally.)

Two vectors are equal if and only if they are of the same order and each element of one vector is equal to the corresponding element of the other.

Our view of vectors essentially associates the elements of a vector with the *coordinates* of a cartesian geometry. There are other, more abstract, ways of developing a theory of vectors that are called "coordinate-free", but we will not pursue those approaches here. For most applications in statistics, the approach based on coordinates is more useful.

Thinking of the coordinates simply as real numbers, we use the notation

$$\mathbb{R}^n \tag{1.3}$$

to denote the set of n-vectors with real elements.

This notation reinforces the notion that the coordinates of a vector correspond to the *direct product* of single coordinates. The direct product of two sets is denoted as "$\otimes$". For sets A and B, it is the set of all ordered doubletons

$$\{(a, b), \text{ s.t. } a \in A, b \in B\},$$

hence, $\mathbb{R}^n = \mathbb{R} \otimes \cdots \otimes \mathbb{R}$ (n times).

1.2 Arrays

Arrays are structured collections of elements corresponding in shape to lines, rectangles, or rectangular solids. The number of dimensions of an array is often called the *rank* of the array. Thus, a vector is an array of rank 1, and a matrix is an array of rank 2. A scalar, which can be thought of as a degenerate array, has rank 0. When referring to computer software objects, "rank" is generally used in this sense. (This term comes from its use in describing a *tensor*. A rank 0 tensor is a scalar, a rank 1 tensor is a vector, a rank 2 tensor is a *square* matrix, and so on. In our usage referring to arrays, we do not require that the dimensions be equal, however.) When we refer to "rank of an array", we mean the number of dimensions. When we refer to "rank of a matrix", we mean something different, as we discuss in Sect. 3.3. In linear algebra, this latter usage is far more common than the former.

1.3 Matrices

A matrix is a rectangular or two-dimensional array. We speak of the *rows* and *columns* of a matrix. The rows or columns can be considered to be vectors, and we often use this equivalence. An $n \times m$ matrix is one with n rows and m columns. The number of rows and the number of columns determine the

shape of the matrix. Note that the shape is the doubleton (n, m), not just a single number such as the ratio. If the number of rows is the same as the number of columns, the matrix is said to be square.

All matrices are two-dimensional in the sense of "dimension" used above. The word "dimension", however, when applied to matrices, often means something different, namely the number of columns. (This usage of "dimension" is common both in geometry and in traditional statistical applications.)

We usually use an uppercase letter to represent a matrix. To represent an element of the matrix, we usually use the corresponding lowercase letter with a subscript to denote the row and a second subscript to represent the column. If a nontrivial expression is used to denote the row or the column, we separate the row and column subscripts with a comma.

Although vectors and matrices are fundamentally quite different types of objects, we can bring some unity to our discussion and notation by occasionally considering a vector to be a "column vector" and in some ways to be the same as an $n \times 1$ matrix. (This has nothing to do with the way we may write the elements of a vector. The notation in equation (1.2) is more convenient than that in equation (1.1) and so will generally be used in this book, but its use does not change the nature of the vector in any way. Likewise, this has nothing to do with the way the elements of a vector or a matrix are stored in the computer.) When we use vectors and matrices in the same expression, however, we use the symbol "T" (for "transpose") as a superscript to represent a vector that is being treated as a $1 \times n$ matrix.

The first row is the 1st (first) row, and the first column is the 1st (first) column. (Again, we remark that computer entities used in some systems to represent matrices and to store elements of matrices as computer data sometimes index the elements beginning with 0. Furthermore, some systems use the first index to represent the column and the second index to indicate the row. We are not speaking here of the *storage order*—"row major" versus "column major"—we address that later, in Chap. 11. Rather, we are speaking of the mechanism of *referring to* the abstract entities. In image processing, for example, it is common practice to use the first index to represent the column and the second index to represent the row. In the software packages IDL and PV-Wave, for example, there are two different kinds of two-dimensional objects: "arrays", in which the indexing is done as in image processing, and "matrices", in which the indexing is done as we have described.)

The $n \times m$ matrix A can be written

$$A = \begin{bmatrix} a_{11} & \dots & a_{1m} \\ \vdots & \vdots & \vdots \\ a_{n1} & \dots & a_{nm} \end{bmatrix}. \tag{1.4}$$

We also write the matrix A above as

$$A = (a_{ij}), \tag{1.5}$$

with the indices i and j ranging over $\{1, \ldots, n\}$ and $\{1, \ldots, m\}$, respectively. We use the notation $A_{n \times m}$ to refer to the matrix A and simultaneously to indicate that it is $n \times m$, and we use the notation

$$\mathbb{R}^{n \times m} \tag{1.6}$$

to refer to the set of all $n \times m$ matrices with real elements.

We use the notation $(A)_{ij}$ to refer to the element in the i^{th} row and the j^{th} column of the matrix A; that is, in equation (1.4), $(A)_{ij} = a_{ij}$.

Two matrices are equal if and only if they are of the same shape and each element of one matrix is equal to the corresponding element of the other.

Although vectors are column vectors and the notation in equations (1.1) and (1.2) represents the same entity, that would not be the same for matrices. If $x_1, \ldots, x_n$ are scalars

$$X = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \tag{1.7}$$

and

$$Y = [x_1, \ldots, x_n], \tag{1.8}$$

then X is an $n \times 1$ matrix and Y is a $1 \times n$ matrix and $X \neq Y$ unless $n = 1$. (Y is the *transpose* of X.) Although an $n \times 1$ matrix is a different type of object from a vector, we may treat X in equation (1.7) or Y^{T} in equation (1.8) as a vector when it is convenient to do so. Furthermore, although a 1×1 matrix, a 1-vector, and a scalar are all fundamentally different types of objects, we will treat a one by one matrix or a vector with only one element as a scalar whenever it is convenient.

We sometimes use the notation a_{*j} to correspond to the j^{th} column of the matrix A and use a_{i*} to represent the (column) vector that corresponds to the i^{th} row. Using that noation, the $n \times m$ matrix A in equation (1.4) can be written as

$$A = \begin{bmatrix} a_{1*}^{\mathrm{T}} \\ \vdots \\ a_{n*}^{\mathrm{T}} \end{bmatrix}. \tag{1.9}$$

or as

$$A = [a_{*1}, \ldots, a_{*m}]. \tag{1.10}$$

One of the most important uses of matrices is as a transformation of a vector by vector/matrix multiplication. Such transformations are linear (a term that we define later). Although one can occasionally profitably distinguish matrices from linear transformations on vectors, for our present purposes there is no advantage in doing so. We will often treat matrices and linear transformations as equivalent.

Many of the properties of vectors and matrices we discuss hold for an infinite number of elements, but we will assume throughout this book that the number is finite.

1.3.1 Subvectors and Submatrices

We sometimes find it useful to work with only some of the elements of a vector or matrix. We refer to the respective arrays as "subvectors" or "submatrices". We also allow the rearrangement of the elements by row or column permutations and still consider the resulting object as a subvector or submatrix. In Chap. 3, we will consider special forms of submatrices formed by "partitions" of given matrices.

The two expressions (1.9) and (1.10) represent special partitions of the matrix A.

1.4 Representation of Data

Before we can do any serious analysis of data, the data must be represented in some structure that is amenable to the operations of the analysis. In simple cases, the data are represented by a list of scalar values. The ordering in the list may be unimportant, and the analysis may just consist of computation of simple summary statistics. In other cases, the list represents a time series of observations, and the relationships of observations to each other as a function of their order and distance apart in the list are of interest. Often, the data can be represented meaningfully in two lists that are related to each other by the positions in the lists. The generalization of this representation is a two-dimensional array in which each column corresponds to a particular type of data.

A major consideration, of course, is the nature of the individual items of data. The observational data may be in various forms: quantitative measures, colors, text strings, and so on. Prior to most analyses of data, they must be represented as real numbers. In some cases, they can be represented easily as real numbers, although there may be restrictions on the mapping into the reals. (For example, do the data naturally assume only integral values, or could any real number be mapped back to a possible observation?)

The most common way of representing data is by using a two-dimensional array in which the rows correspond to observational units ("instances") and the columns correspond to particular types of observations ("variables" or "features"). If the data correspond to real numbers, this representation is the familiar X data matrix. Much of this book is devoted to the matrix theory and computational methods for the analysis of data in this form. This type of matrix, perhaps with an adjoined vector, is the basic structure used in many

familiar statistical methods, such as regression analysis, principal components analysis, analysis of variance, multidimensional scaling, and so on.

There are other types of structures based on graphs that are useful in representing data. A *graph* is a structure consisting of two components: a set of points, called *vertices* or *nodes* and a set of pairs of the points, called *edges*. (Note that this usage of the word "graph" is distinctly different from the more common one that refers to lines, curves, bars, and so on to represent data pictorially. The phrase "graph theory" is often used, or overused, to emphasize the present meaning of the word.) A graph $\mathcal{G} = (V, E)$ with vertices $V = \{v_1, \ldots, v_n\}$ is distinguished primarily by the nature of the edge elements (v_i, v_j) in E. Graphs are identified as complete graphs, directed graphs, trees, and so on, depending on E and its relationship with V. A tree may be used for data that are naturally aggregated in a hierarchy, such as political unit, subunit, household, and individual. Trees are also useful for representing clustering of data at different levels of association. In this type of representation, the individual data elements are the terminal nodes, or "leaves", of the tree.

In another type of graphical representation that is often useful in "data mining" or "learning", where we seek to uncover relationships among objects, the vertices are the objects, either observational units or features, and the edges indicate some commonality between vertices. For example, the vertices may be text documents, and an edge between two documents may indicate that a certain number of specific words or phrases occur in both documents. Despite the differences in the basic ways of representing data, in graphical modeling of data, many of the standard matrix operations used in more traditional data analysis are applied to matrices that arise naturally from the graph.

However the data are represented, whether in an array or a network, the analysis of the data is often facilitated by using "association" matrices. The most familiar type of association matrix is perhaps a correlation matrix. We will encounter and use other types of association matrices in Chap. 8.

What You Compute and What You Don't

The applied mathematician or statistician routinely performs many computations involving vectors and matrices. Many of those computations follow the methods discussed in this text.

For a given matrix X, I will often refer to its inverse X^{-1}, its determinant $\det(X)$, its Gram $X^{\mathrm{T}}X$, a matrix formed by permuting its columns $E_{(\pi)}X$, a matrix formed by permuting its rows $XE_{(\pi)}$, and other transformations of the given matrix X. These derived objects are very important and useful. Their usefulness, however, is primarily conceptual.

When working with a real matrix X whose elements have actual known values, it is not very often that we need or want the actual values of elements

of these derived objects. Because of this, some authors try to avoid discussing or referring directly to these objects.

I do not avoid discussing the objects, but, for example, when I write $(X^{\mathrm{T}}X)^{-1}X^{\mathrm{T}}y$, I do not mean that you should compute $X^{\mathrm{T}}X$ and $X^{\mathrm{T}}y$, then compute $(X^{\mathrm{T}}X)^{-1}$, and then finally multiply $(X^{\mathrm{T}}X)^{-1}$ and $X^{\mathrm{T}}y$. I assume you know better than to do that. If you don't know it yet, I hope after reading this book, you will know why not to.

2

Vectors and Vector Spaces

In this chapter we discuss a wide range of basic topics related to vectors of real numbers. Some of the properties carry over to vectors over other fields, such as complex numbers, but the reader should not assume this. Occasionally, for emphasis, we will refer to "real" vectors or "real" vector spaces, but unless it is stated otherwise, we are assuming the vectors and vector spaces are real. The topics and the properties of vectors and vector spaces that we emphasize are motivated by applications in the data sciences.

2.1 Operations on Vectors

The elements of the vectors we will use in the following are real numbers, that is, elements of $\mathbb{R}$. We call elements of $\mathbb{R}$ *scalars*. Vector operations are defined in terms of operations on real numbers.

Two vectors can be added if they have the same number of elements. The sum of two vectors is the vector whose elements are the sums of the corresponding elements of the vectors being added. Vectors with the same number of elements are said to be *conformable* for addition. A vector all of whose elements are 0 is the *additive identity* for all conformable vectors.

We overload the usual symbols for the operations on the reals to signify the corresponding operations on vectors or matrices when the operations are defined. Hence, "+" can mean addition of scalars, addition of conformable vectors, or addition of a scalar to a vector. This last meaning of "+" may not be used in many mathematical treatments of vectors, but it is consistent with the semantics of modern computer languages such as Fortran, R, and Matlab. By the *addition of a scalar and a vector*, we mean the addition of the scalar to each element of the vector, resulting in a vector of the same number of elements.

J.E. Gentle, *Matrix Algebra*, Springer Texts in Statistics,
DOI 10.1007/978-3-319-64867-5_2

A *scalar multiple of a vector* (that is, the product of a real number and a vector) is the vector whose elements are the multiples of the corresponding elements of the original vector. Juxtaposition of a symbol for a scalar and a symbol for a vector indicates the multiplication of the scalar with each element of the vector, resulting in a vector of the same number of elements.

The basic operation in working with vectors is the addition of a scalar multiple of one vector to another vector,

$$z = ax + y, \tag{2.1}$$

where a is a scalar and x and y are vectors conformable for addition. Viewed as a single operation with three operands, this is called an *axpy operation* for obvious reasons. (Because the Fortran versions of BLAS to perform this operation were called `saxpy` and `daxpy`, the operation is also sometimes called "saxpy" or "daxpy". See Sect. 12.2.1 on page 555, for a description of the BLAS.)

The axpy operation is a *linear combination*. Such linear combinations of vectors are the basic operations in most areas of linear algebra. The composition of axpy operations is also an axpy; that is, one linear combination followed by another linear combination is a linear combination. Furthermore, any linear combination can be decomposed into a sequence of axpy operations.

A special linear combination is called a *convex combination*. For vectors x and y, it is the combination

$$ax + by, \tag{2.2}$$

where $a, b \geq 0$ and $a + b = 1$. A set of vectors that is closed with respect to convex combinations is said to be *convex*.

2.1.1 Linear Combinations and Linear Independence

If a given vector can be formed by a linear combination of one or more vectors, the set of vectors (including the given one) is said to be linearly dependent; conversely, if in a set of vectors no one vector can be represented as a linear combination of any of the others, the set of vectors is said to be *linearly independent*. In equation (2.1), for example, the vectors x, y, and z are not linearly independent. It is possible, however, that any two of these vectors are linearly independent.

Linear independence is one of the most important concepts in linear algebra.

We can see that the definition of a linearly independent set of vectors $\{v_1, \ldots, v_k\}$ is equivalent to stating that if

$$a_1 v_1 + \cdots a_k v_k = 0, \tag{2.3}$$

then $a_1 = \cdots = a_k = 0$. If the set of vectors $\{v_1, \ldots, v_k\}$ is not linearly independent, then it is possible to select a *maximal linearly independent subset*;

that is, a subset of $\{v_1, \ldots, v_k\}$ that is linearly independent and has maximum cardinality. We do this by selecting an arbitrary vector, v_{i_1}, and then seeking a vector that is independent of v_{i_1}. If there are none in the set that is linearly independent of v_{i_1}, then a maximum linearly independent subset is just the singleton, because all of the vectors must be a linear combination of just one vector (that is, a scalar multiple of that one vector). If there is a vector that is linearly independent of v_{i_1}, say v_{i_2}, we next seek a vector in the remaining set that is independent of v_{i_1} and v_{i_2}. If one does not exist, then $\{v_{i_1}, v_{i_2}\}$ is a maximal subset because any other vector can be represented in terms of these two and hence, within any subset of three vectors, one can be represented in terms of the two others. Thus, we see how to form a maximal linearly independent subset, and we see that the maximum cardinality of any subset of linearly independent vectors is unique however they are formed.

It is easy to see that the maximum number of n-vectors that can form a set that is linearly independent is n. (We can see this by assuming n linearly independent vectors and then, for any $(n+1)^{\text{th}}$ vector, showing that it is a linear combination of the others by building it up one by one from linear combinations of two of the given linearly independent vectors. In Exercise 2.1, you are asked to write out these steps.)

Properties of a set of vectors are usually invariant to a permutation of the elements of the vectors if the same permutation is applied to all vectors in the set. In particular, if a set of vectors is linearly independent, the set remains linearly independent if the elements of each vector are permuted in the same way.

If the elements of each vector in a set of vectors are separated into subvectors, linear independence of any set of corresponding subvectors implies linear independence of the full vectors. To state this more precisely for a set of three n-vectors, let $x = (x_1, \ldots, x_n)$, $y = (y_1, \ldots, y_n)$, and $z = (z_1, \ldots, z_n)$. Now let $\{i_1, \ldots, i_k\} \subseteq \{1, \ldots, n\}$, and form the k-vectors $\tilde{x} = (x_{i_1}, \ldots, x_{i_k})$, $\tilde{y} = (y_{i_1}, \ldots, y_{i_k})$, and $\tilde{z} = (z_{i_1}, \ldots, z_{i_k})$. Then linear independence of $\tilde{x}$, $\tilde{y}$, and $\tilde{z}$ implies linear independence of x, y, and z. (This can be shown directly from the definition of linear independence. It is related to equation (2.19) on page 20, which you are asked to prove in Exercise 2.5.)

2.1.2 Vector Spaces and Spaces of Vectors

Let V be a set of n-vectors such that any linear combination of the vectors in V is also in V. Such a set together with the usual vector algebra is called a *vector space*. A vector space is a *linear space*, and it necessarily includes the additive identity (the zero vector). (To see this, in the axpy operation, let $a = -1$ and $y = x$.) A vector space is necessarily convex.

The set consisting only of the additive identity, along with the axpy operation, is a vector space. It is called the "null vector space". Some people define "vector space" in a way that excludes it, because its properties do not conform to many general statements we can make about other vector spaces.

The "usual algebra" is a *linear algebra* consisting of two operations: vector addition and scalar times vector multiplication, which are the two operations comprising an axpy. It has closure of the space under the combination of those operations, commutativity and associativity of addition, an additive identity and inverses, a multiplicative identity, distribution of multiplication over both vector addition and scalar addition, and associativity of scalar multiplication and scalar times vector multiplication.

A vector space can also be composed of other objects, such as matrices, along with their appropriate operations. The key characteristic of a vector space is a linear algebra.

We generally use a calligraphic font to denote a vector space; $\mathcal{V}$ or $\mathcal{W}$, for example. Often, however, we think of the vector space merely in terms of the set of vectors on which it is built and denote it by an ordinary capital letter; V or W, for example. A vector space is an *algebraic structure* consisting of a set together with the axpy operation, with the restriction that the set is closed under the operation. To indicate that it is a structure, rather than just a set, we may write

$$\mathcal{V} = (V, \circ),$$

where V is just the set and $\circ$ denotes the axpy operation, or a similar linear operation under which the set is closed.

2.1.2.1 Generating Sets

Given a set G of vectors of the same order, a vector space can be formed from the set G together with all vectors that result from the axpy operation being applied to all combinations of vectors in G and all values of the real number a; that is, for all $v_i, v_j \in G$ and all real a,

$$\{av_i + v_j\}.$$

This set together with the axpy operation itself is a vector space. It is called the *space generated by* G. We denote this space as

$$\text{span}(G).$$

We will discuss generating and spanning sets further in Sect. 2.1.3.

2.1.2.2 The Order and the Dimension of a Vector Space

The vector space consisting of all n-vectors with real elements is denoted $\mathbb{R}^n$. (As mentioned earlier, the notation $\mathbb{R}^n$ can also refer to just the *set* of n-vectors with real elements; that is, to the set over which the vector space is defined.)

The *dimension of a vector space* is the maximum number of linearly independent vectors in the vector space. We denote the dimension by

$$\dim(\cdot),$$

which is a mapping $\mathbb{R}^n \to \mathbb{Z}_+$ (where $\mathbb{Z}_+$ denotes the positive integers).

The *order of a vector space* is the order of the vectors in the space. Because the maximum number of n-vectors that can form a linearly independent set is n, as we showed above, the order of a vector space is greater than or equal to the dimension of the vector space.

Both the order and the dimension of $\mathbb{R}^n$ are n. A set of m linearly independent n-vectors with real elements can generate a vector space within $\mathbb{R}^n$ of order n and dimension m.

We also may use the phrase *dimension of a vector* to mean the dimension of the vector space of which the vector is an element. This term is ambiguous, but its meaning is clear in specific contexts, such as *dimension reduction*, that we will discuss later.

2.1.2.3 Vector Spaces with an Infinite Number of Dimensions

It is possible that no finite set of vectors span a given vector space. In that case, the vector space is said to be of infinite dimension.

Many of the properties of vector spaces that we discuss hold for those with an infinite number of dimensions; but not all do, such as the equivalence of norms (see page 29).

Throughout this book, however, unless we state otherwise, we assume the vector spaces have a finite number of dimensions.

2.1.2.4 Essentially Disjoint Vector Spaces

If the only element in common between two vector spaces $\mathcal{V}$ and $\mathcal{W}$ is the additive identity, the spaces are said to be *essentially disjoint*. Essentially disjoint vector spaces necessarily have the same order.

If the vector spaces $\mathcal{V}$ and $\mathcal{W}$ are essentially disjoint, it is clear that any element in $\mathcal{V}$ (except the additive identity) is linearly independent of any set of elements in $\mathcal{W}$.

2.1.2.5 Some Special Vectors: Notation

We denote the additive identity in a vector space of order n by 0_n or sometimes by 0. This is the vector consisting of all zeros:

$$0_n = (0, \ldots, 0). \tag{2.4}$$

We call this the *zero vector*, or the *null vector*. (A vector $x \neq 0$ is called a "nonnull vector".) This vector by itself is sometimes called the *null vector space*. It is not a vector space in the usual sense; it would have dimension 0. (All linear combinations are the same.)

Likewise, we denote the vector consisting of all ones by 1_n or sometimes by 1:

$$1_n = (1, \ldots, 1). \tag{2.5}$$

We call this the *one vector* and also the "summing vector" (see page 34). This vector and all scalar multiples of it are vector spaces with dimension 1. (This is true of any single nonzero vector; all linear combinations are just scalar multiples.) Whether 0 and 1 without a subscript represent vectors or scalars is usually clear from the context.

The zero vector and the one vector are both instances of *constant vectors*; that is, vectors all of whose elements are the same. In some cases we may abuse the notation slightly, as we have done with "0" and "1" above, and use a single symbol to denote both a scalar and a vector all of whose elements are that constant; for example, if "c" denotes a scalar constant, we may refer to the vector all of whose elements are c as "c" also. These notational conveniences rarely result in any ambiguity. They also allow another interpretation of the definition of addition of a scalar to a vector that we mentioned at the beginning of the chapter.

The i^{th} *unit vector*, denoted by e_i, has a 1 in the i^{th} position and 0s in all other positions:

$$e_i = (0, \ldots, 0, 1, 0, \ldots, 0). \tag{2.6}$$

Another useful vector is the *sign vector*, which is formed from signs of the elements of a given vector. It is denoted by "sign$(\cdot)$" and for $x = (x_1, \ldots, x_n)$ is defined by

$$\begin{aligned} \operatorname{sign}(x)_i &= 1 \quad \text{if } x_i > 0, \\ &= 0 \quad \text{if } x_i = 0, \\ &= -1 \quad \text{if } x_i < 0. \end{aligned} \tag{2.7}$$

2.1.2.6 Ordinal Relations Among Vectors

There are several possible ways to form a rank ordering of vectors of the same order, but no complete ordering is entirely satisfactory. (Note the unfortunate overloading of the words "order" and "ordering" here.) If x and y are vectors of the same order and for corresponding elements $x_i > y_i$, we say x is *greater than* y and write

$$x > y. \tag{2.8}$$

In particular, if all of the elements of x are positive, we write $x > 0$.

If x and y are vectors of the same order and for corresponding elements $x_i \geq y_i$, we say x is *greater than or equal to* y and write

$$x \geq y. \tag{2.9}$$

This relationship is a *partial ordering* (see Exercise 8.2a on page 396 for the definition of partial ordering).

The expression $x \geq 0$ means that all of the elements of x are nonnegative.

2.1.2.7 Set Operations on Vector Spaces

The ordinary operations of subsetting, intersection, union, direct sum, and direct product for sets have analogs for vector spaces, and we use some of the same notation to refer to vector spaces that we use to refer to sets. The set operations themselves are performed on the individual sets to yield a set of vectors, and the resulting vector space is the space generated by that set of vectors.

Unfortunately, there are many inconsistencies in terminology used in the literature regarding operations on vector spaces. When I use a term and/or symbol, such as "union" or "$\cup$", for a structure such as a vector space, I use it in reference to the *structure*. For example, if $\mathcal{V} = (V, \circ)$ and $\mathcal{W} = (W, \circ)$ are vector spaces, then $V \cup U$ is the ordinary union of the sets; however, $\mathcal{V} \cup \mathcal{W}$ is the union of the vector spaces, and is not necessarily the same as $(U \cup W, \circ)$, which may not even be a vector space. Occasionally in the following discussion, I will try to point out common variants in usage.

The convention that I follow allows the wellknown relationships among common set operations to hold for the corresponding operations on vector spaces; for example, if $\mathcal{V}$ and $\mathcal{W}$ are vector spaces, $\mathcal{V} \subseteq \mathcal{V} \cup \mathcal{W}$, just as for sets V and W.

The properties of vector spaces are proven the same way that properties of sets are proven, after first requiring that the axpy operation have the same meaning in the different vector spaces. For example, to prove that one vector space is a subspace of another, we show that any given vector in the first vector space is necessarily in the second. To prove that two vector spaces are equal, we show that each is a subspace of the other. Some properties of vector spaces and subspaces can be shown more easily using "basis sets" for the spaces, which we discuss in Sect. 2.1.3, beginning on page 21.

Note that if $(V, \circ)$ and $(W, \circ)$ are vector spaces of the same order and U is some set formed by an operation on V and W, then $(U, \circ)$ may not be a vector space because it is not closed under the axpy operation, $\circ$. We sometimes refer to a set of vectors of the same order together with the axpy operator (whether or not the set is closed with respect to the operator) as a "space of vectors" (instead of a "vector space").

2.1.2.8 Subpaces

Given a vector space $\mathcal{V} = (V, \circ)$, if W is any subset of V, then the vector space $\mathcal{W}$ generated by W, that is, $\text{span}(W)$, is said to be a *subspace* of $\mathcal{V}$, and we denote this relationship by $\mathcal{W} \subseteq \mathcal{V}$.

If $\mathcal{W} \subseteq \mathcal{V}$ and $\mathcal{W} \neq \mathcal{V}$, then $\mathcal{W}$ is said to be a *proper subspace* of $\mathcal{V}$. If $\mathcal{W} = \mathcal{V}$, then $\mathcal{W} \subseteq \mathcal{V}$ and $\mathcal{V} \subseteq \mathcal{W}$, and the converse is also true.

The maximum number of linearly independent vectors in the subspace cannot be greater than the maximum number of linearly independent vectors in the original space; that is, if $\mathcal{W} \subseteq \mathcal{V}$, then

$$\dim(\mathcal{W}) \leq \dim(\mathcal{V}) \tag{2.10}$$

(Exercise 2.2). If $\mathcal{W}$ is a proper subspace of $\mathcal{V}$, then $\dim(\mathcal{W}) < \dim(\mathcal{V})$.

2.1.2.9 Intersections of Vector Spaces

For two vector spaces $\mathcal{V}$ and $\mathcal{W}$ of the same order with vectors formed from the same field, we define their *intersection*, denoted by $\mathcal{V} \cap \mathcal{W}$, to be the set of vectors consisting of the intersection of the sets in the individual vector spaces together with the axpy operation.

The intersection of two vector spaces of the same order that are not essentially disjoint is a vector space, as we can see by letting x and y be any vectors in the intersection $\mathcal{U} = \mathcal{V} \cap \mathcal{W}$, and showing, for any real number a, that $ax + y \in \mathcal{U}$. This is easy because both x and y must be in both $\mathcal{V}$ and $\mathcal{W}$.

Note that if $\mathcal{V}$ and $\mathcal{W}$ are essentially disjoint, then $\mathcal{V} \cap \mathcal{W} = (0, \circ)$, which, as we have said, is not a vector space in the usual sense.

Also note that

$$\dim(\mathcal{V} \cap \mathcal{W}) \leq \min(\dim(\mathcal{V}), \dim(\mathcal{W})) \tag{2.11}$$

(Exercise 2.2).

2.1.2.10 Unions and Direct Sums of Vector Spaces

Given two vector spaces $\mathcal{V}$ and $\mathcal{W}$ of the same order, we define their *union*, denoted by $\mathcal{V} \cup \mathcal{W}$, to be the vector space generated by the union of the sets in the individual vector spaces together with the axpy operation. If $\mathcal{V} = (V, \circ)$ and $\mathcal{W} = (W, \circ)$, this is the vector space generated by the set of vectors $V \cup W$; that is,

$$\mathcal{V} \cup \mathcal{W} = \text{span}(V \cup W). \tag{2.12}$$

The union of the sets of vectors in two vector spaces may not be closed under the axpy operation (Exercise 2.3b), but the union of vector spaces *is* a vector space by definition.

The vector space generated by the union of the sets in the individual vector spaces is easy to form. Since $(V, \circ)$ and $(W, \circ)$ are vector spaces (so for any vector x in either V or W, ax is in that set), all we need do is just include all simple sums of the vectors from the individual sets, that is,

$$\mathcal{V} \cup \mathcal{W} = \{v + w, \text{ s.t. } v \in \mathcal{V}, \ w \in \mathcal{W}\}. \tag{2.13}$$

It is easy to see that this is a vector space by showing that it is closed with respect to axpy. (As above, we show that for any x and y in $\mathcal{V} \cup \mathcal{W}$ and for any real number a, $ax + y$ is in $\mathcal{V} \cup \mathcal{W}$.)

(Because of the way the union of vector spaces can be formed from simple addition of the individual elements, some authors call the vector space in

equation (2.13) the "sum" of $\mathcal{V}$ and $\mathcal{W}$, and write it as $\mathcal{V}+\mathcal{W}$. Other authors, including myself, call this the *direct sum*, and denote it by $\mathcal{V}\oplus\mathcal{W}$. Some authors define "direct sum" only in the cases of vector spaces that are essentially disjoint. Still other authors define "direct sum" to be what I will call a "direct product" below.)

Despite the possible confusion with other uses of the notation, I often use the notation $\mathcal{V} \oplus \mathcal{W}$ because it points directly to the nice construction of equation (2.13). *To be clear:* to the extent that I use "direct sum" and "$\oplus$" for vector spaces $\mathcal{V}$ and $\mathcal{W}$, I will mean the direct sum

$$\mathcal{V} \oplus \mathcal{W} \equiv \mathcal{V} \cup \mathcal{W}, \tag{2.14}$$

as defined above.

Note that

$$\dim(\mathcal{V} \oplus \mathcal{W}) \ = \dim(\mathcal{V}) + \dim(\mathcal{W}) - \dim(\mathcal{V} \cap \mathcal{W}) \tag{2.15}$$

(Exercise 2.4). Therefore

$$\dim(\mathcal{V} \oplus \mathcal{W}) \geq \max(\dim(\mathcal{V}), \dim(\mathcal{W}))$$

and

$$\dim(\mathcal{V} \oplus \mathcal{W}) \leq \dim(\mathcal{V}) + \dim(\mathcal{W}).$$

2.1.2.11 Direct Sum Decomposition of a Vector Space

In some applications, given a vector space $\mathcal{V}$, it is of interest to find essentially disjoint vector spaces $\mathcal{V}_1, \ldots, \mathcal{V}_n$ such that

$$\mathcal{V} = \mathcal{V}_1 \oplus \cdots \oplus \mathcal{V}_n.$$

This is called a *direct sum decomposition* of $\mathcal{V}$. (As I mentioned above, some authors who do not use "direct sum" as I do would use the term in this context because the individual matrices are essentially disjoint.)

It is clear that if $\mathcal{V}_1, \ldots, \mathcal{V}_n$ is a direct sum decomposition of $\mathcal{V}$, then

$$\dim(\mathcal{V}) = \sum_{i=1}^{n} \dim(\mathcal{V}_i) \tag{2.16}$$

(Exercise 2.4).

A collection of essentially disjoint vector spaces $\mathcal{V}_1, \ldots, \mathcal{V}_n$ such that $\mathcal{V} = \mathcal{V}_1 \oplus \cdots \oplus \mathcal{V}_n$ is said to be *complementary with respect to* $\mathcal{V}$.

An important property of a direct sum decomposition is that it allows a unique representation of a vector in the decomposed space in terms of a sum of vectors from the individual essentially disjoint spaces; that is, if $\mathcal{V} = \mathcal{V}_1 \oplus \cdots \oplus \mathcal{V}_n$ is a direct sum decomposition of $\mathcal{V}$ and $v \in \mathcal{V}$, then there exist unique vectors $v_i \in \mathcal{V}_i$ such that

$$v = v_1 + \cdots + v_n. \tag{2.17}$$

We will prove this for the case $n = 2$. This is without loss, because additional spaces in the decomposition add nothing different.

Given the direct sum decomposition $\mathcal{V} = \mathcal{V}_1 \oplus \mathcal{V}_2$, let v be any vector in $\mathcal{V}$. Because $\mathcal{V}_1 \oplus \mathcal{V}_2$ can be formed as in equation (2.13), there exist vectors $v_1 \in \mathcal{V}_1$ and $v_2 \in \mathcal{V}_2$ such that $v = v_1 + v_2$. Now all we need to do is to show that they are unique.

Let $u_1 \in \mathcal{V}_1$ and $u_2 \in \mathcal{V}_2$ be such that $v = u_1 + u_2$. Now we have $(v - u_1) \in \mathcal{V}_2$ and $(v - v_1) \in \mathcal{V}_2$; hence $(v_1 - u_1) \in \mathcal{V}_2$. However, since $v_1, u_1 \in \mathcal{V}_1$, $(v_1 - u_1) \in \mathcal{V}_1$. Since $\mathcal{V}_1$ and $\mathcal{V}_2$ are essentially disjoint, and $(v_1 - u_1)$ is in both, it must be the case that $(v_1 - u_1) = 0$, or $u_1 = v_1$. In like manner, we show that $u_2 = v_2$; hence, the representation $v = v_1 + v_2$ is unique.

An important fact is that for any vector space $\mathcal{V}$ with dimension 2 or greater, a direct sum decomposition exists; that is, there exist essentially disjoint vector spaces $\mathcal{V}_1$ and $\mathcal{V}_2$ such that $\mathcal{V} = \mathcal{V}_1 \oplus \mathcal{V}_2$.

This is easily shown by first choosing a proper subspace $\mathcal{V}_1$ of $\mathcal{V}$ and then constructing an essentially disjoint subspace $\mathcal{V}_2$ such that $\mathcal{V} = \mathcal{V}_1 \oplus \mathcal{V}_2$. The details of these steps are made simpler by use of basis sets which we will discuss in Sect. 2.1.3, in particular the facts listed on page 22.

2.1.2.12 Direct Products of Vector Spaces and Dimension Reduction

The set operations on vector spaces that we have mentioned so far require that the vector spaces be of a fixed order. Sometimes in applications, it is useful to deal with vector spaces of different orders.

The *direct product* of the vector space $\mathcal{V}$ of order n and the vector space $\mathcal{W}$ of order m is the vector space of order $n + m$ on the set of vectors

$$\{(v_1, \ldots, v_n, w_1, \ldots, w_m), \text{ s.t. } (v_1, \ldots, v_n) \in \mathcal{V}, (w_1, \ldots, w_m) \in \mathcal{W}\}, \tag{2.18}$$

together with the axpy operator defined as the same operator in $\mathcal{V}$ and $\mathcal{W}$ applied separately to the first n and the last m elements. The direct product of $\mathcal{V}$ and $\mathcal{W}$ is denoted by $\mathcal{V} \otimes \mathcal{W}$.

Notice that while the direct sum operation is commutative, the direct product is not commutative in general.

The vectors in $\mathcal{V}$ and $\mathcal{W}$ are sometimes called "subvectors" of the vectors in $\mathcal{V} \otimes \mathcal{W}$. These subvectors are related to projections, which we will discuss in more detail in Sect. 2.2.2 (page 36) and Sect. 8.5.2 (page 358).

We can see that the direct product is a vector space using the same method as above by showing that it is closed under the axpy operation.

Note that

$$\dim(\mathcal{V} \otimes \mathcal{W}) = \dim(\mathcal{V}) + \dim(\mathcal{W}) \tag{2.19}$$

(Exercise 2.5).

Note that for integers $0 < p < n$,

$$\mathbb{R}^n = \mathbb{R}^p \otimes \mathbb{R}^{n-p}, \tag{2.20}$$

where the operations in the space $\mathbb{R}^n$ are the same as in the component vector spaces with the meaning adjusted to conform to the larger order of the vectors in $\mathbb{R}^n$. (Recall that $\mathbb{R}^n$ represents the algebraic structure consisting of the set of n-tuples of real numbers plus the special axpy operator.)

In statistical applications, we often want to do "dimension reduction". This means to find a smaller number of coordinates that cover the relevant regions of a larger-dimensional space. In other words, we are interested in finding a lower-dimensional vector space in which a given set of vectors in a higher-dimensional vector space can be approximated by vectors in the lower-dimensional space. For a given set of vectors of the form $x = (x_1, \ldots, x_n)$ we seek a set of vectors of the form $z = (z_1, \ldots, z_p)$ that almost "cover the same space". (The transformation from x to z is called a projection.)

2.1.3 Basis Sets for Vector Spaces

If each vector in the vector space $\mathcal{V}$ can be expressed as a linear combination of the vectors in some set G, then G is said to be a *generating set* or *spanning set* of $\mathcal{V}$. The number of vectors in a generating set is at least as great as the dimension of the vector space.

If all linear combinations of the elements of G are in $\mathcal{V}$, the vector space is the *space generated by* G and is denoted by $\mathcal{V}(G)$ or by $\mathrm{span}(G)$, as we mentioned on page 14. We will use either notation interchangeably:

$$\mathcal{V}(G) \equiv \mathrm{span}(G). \tag{2.21}$$

Note that G is also a generating or spanning set for $\mathcal{W}$ where $\mathcal{W} \subseteq \mathrm{span}(G)$.

A *basis* for a vector space is a set of linearly independent vectors that generate or span the space. For any vector space, a generating set consisting of the minimum number of vectors of any generating set for that space is a basis set for the space. A basis set is obviously not unique.

Note that the linear independence implies that a basis set cannot contain the 0 vector.

An important fact is

- The representation of a given vector in terms of a given basis set is unique.

To see this, let $\{v_1, \ldots, v_k\}$ be a basis for a vector space that includes the vector x, and let

$$x = c_1 v_1 + \cdots c_k v_k.$$

Now suppose

$$x = b_1 v_1 + \cdots b_k v_k,$$

so that we have

$$0 = (c_1 - b_1)v_1 + \cdots + (c_k - b_k)v_k.$$

Since $\{v_1, \ldots, v_k\}$ are independent, the only way this is possible is if $c_i = b_i$ for each i.

A related fact is that if $\{v_1, \ldots, v_k\}$ is a basis for a vector space of order n that includes the vector x and $x = c_1v_1 + \cdots c_kv_k$, then $x = 0_n$ if and only if $c_i = 0$ for each i.

For any vector space, the order of the vectors in a basis set is the same as the order of the vector space.

Because the vectors in a basis set are independent, the number of vectors in a basis set is the same as the dimension of the vector space; that is, if B is a basis set of the vector space $\mathcal{V}$, then

$$\dim(\mathcal{V}) = \#(B). \tag{2.22}$$

A simple basis set for the vector space $\mathbb{R}^n$ is the set of unit vectors $\{e_1, \ldots, e_n\}$, defined on page 16.

2.1.3.1 Properties of Basis Sets of Vector Subspaces

There are several interesting facts about basis sets for vector spaces and various combinations of the vector spaces. Verifications of these facts all follow similar arguments, and most are left as exercises.

- If B_1 is a basis set for $\mathcal{V}_1$, B_2 is a basis set for $\mathcal{V}_2$, and $\mathcal{V}_1$ and $\mathcal{V}_2$ are essentially disjoint, then $B_1 \cap B_2 = \emptyset$.
  This fact is easily seen by assuming the contrary; that is, assume that $b \in B_1 \cap B_2$. (Note that b cannot be the 0 vector.) This implies, however, that b is in both $\mathcal{V}_1$ and $\mathcal{V}_2$, contradicting the hypothesis that they are essentially disjoint.
- If B is a basis set for $\mathcal{V}$ and $\mathcal{V}_1 \subseteq \mathcal{V}$, then there exists B_1, with $B_1 \subseteq B$, such that B_1 is a basis set for $\mathcal{V}_1$.
- If B_1 is a basis set for $\mathcal{V}_1$ and B_2 is a basis set for $\mathcal{V}_2$, then $B_1 \cup B_2$ is a generating set for $\mathcal{V}_1 \oplus \mathcal{V}_2$.
  (We see this easily from the definition of $\oplus$ because any vector in $\mathcal{V}_1 \oplus \mathcal{V}_2$ can be represented as a linear combination of vectors in B_1 plus a linear combination of vectors in B_2.)
- If $\mathcal{V}_1$ and $\mathcal{V}_2$ are essentially disjoint, B_1 is a basis set for $\mathcal{V}_1$, and B_2 is a basis set for $\mathcal{V}_2$, then $B_1 \cup B_2$ is a basis set for $\mathcal{V} = \mathcal{V}_1 \oplus \mathcal{V}_2$.
  This is the case that $\mathcal{V}_1 \oplus \mathcal{V}_2$ is a direct sum decomposition of $\mathcal{V}$.
- Suppose $\mathcal{V}_1$ is a real vector space of order n_1 (that is, it is a subspace of $\mathbb{R}^{n_1}$) and B_1 is a basis set for $\mathcal{V}_1$. Now let $\mathcal{V}_2$ be a real vector space of order n_2 and B_2 be a basis set for $\mathcal{V}_2$. For each vector b_1 in B_1 form the vector

$$\tilde{b}_1 = (b_1|0, \ldots, 0) \quad \text{where there are } n_2 \text{ 0s},$$

and let $\widetilde{B}_1$ be the set of all such vectors. (The order of each $\tilde{b}_1 \in \widetilde{B}_1$ is $n_1 + n_2$.) Likewise, for each vector b_2 in B_2 form the vector

$$\tilde{b}_2 = (0, \ldots, 0|b_2) \quad \text{where there are } n_1 \text{ 0s},$$

and let $\widetilde{B}_2$ be the set of all such vectors. Then $\widetilde{B}_1 \cup \widetilde{B}_2$ is a basis for $\mathcal{V}_1 \otimes \mathcal{V}_2$.

2.1.4 Inner Products

A useful operation on two vectors x and y of the same order is the *inner product*, which we denote by $\langle x, y \rangle$ and define as

$$\langle x, y \rangle = \sum_i x_i \bar{y}_i, \tag{2.23}$$

where $\bar{z}$ represents the complex conjugate of z; that is, if $z = a + b\mathrm{i}$, then $\bar{z} = a - b\mathrm{i}$. In general, throughout this book unless stated otherwise, I assume that we are working with real numbers, and hence, $\bar{z} = z$. Most statements will hold whether the numbers are real or complex. When the statements only hold for reals, I will generally include the exception in the statement. The main differences have to do with inner products and an important property defined in terms of an inner product, called *orthogonality*.

In the case of vectors with real elements, we have

$$\langle x, y \rangle = \sum_i x_i y_i. \tag{2.24}$$

In that case (which is what we generally assume throughout this book), the inner product is a mapping

$$\mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}.$$

The inner product is also called the dot product or the scalar product. The dot product is actually a special type of inner product, and there is some ambiguity in the terminology. The dot product is the most commonly used inner product in the applications we consider, and so we will use the terms synonymously.

The inner product is also sometimes written as $x \cdot y$, hence the name dot product. Yet another notation for the inner product for real vectors is $x^{\mathrm{T}}y$, and we will see later that this notation is natural in the context of matrix multiplication. So for real vectors, we have the equivalent notations

$$\langle x, y \rangle \equiv x \cdot y \equiv x^{\mathrm{T}}y. \tag{2.25}$$

(I will mention one more notation that is equivalent for real vectors. This is the "bra·ket" notation originated by Paul Dirac, and is still used in certain

areas of application. Dirac referred to x^{T} as the "bra x", and denoted it as $\langle x|$. He referred to an ordinary vector y as the "ket y", and denoted it as $|y\rangle$. He then denoted the inner product of the vectors as $\langle x||y\rangle$, or, omitting one vertical bar, as $\langle x|y\rangle$.)

In general, the inner product is a mapping from a real vector space $\mathcal{V}$ to $\mathbb{R}$ that has the following properties:

1. Nonnegativity and mapping of the additive identity:
   if $x \neq 0$, then $\langle x, x\rangle > 0$ and $\langle 0, x\rangle = \langle x, 0\rangle = \langle 0, 0\rangle = 0$.
2. Commutativity:
   $\langle x, y\rangle = \langle y, x\rangle$.
3. Factoring of scalar multiplication in dot products:
   $\langle ax, y\rangle = a\langle x, y\rangle$ for real a.
4. Relation of vector addition to addition of dot products:
   $\langle x + y, z\rangle = \langle x, z\rangle + \langle y, z\rangle$.

These properties in fact define an *inner product* for mathematical objects for which an addition, an additive identity, and a multiplication by a scalar are defined. Notice that the operation defined in equation (2.24) is not an inner product for vectors over the complex field because, if x is complex, we can have $\langle x, x\rangle = 0$ when $x \neq 0$.

A vector space together with an inner product is called an *inner product space*.

Inner products are also defined for matrices, as we will discuss on page 97. We should note in passing that there are two different kinds of multiplication used in property 3. The first multiplication is scalar multiplication, that is, an operation from $\mathbb{R} \times \mathbb{R}^n$ to $\mathbb{R}^n$, which we have defined above, and the second multiplication is ordinary multiplication in $\mathbb{R}$, that is, an operation from $\mathbb{R} \times \mathbb{R}$ to $\mathbb{R}$. There are also two different kinds of addition used in property 4. The first addition is vector addition, defined above, and the second addition is ordinary addition in $\mathbb{R}$. The dot product can reveal fundamental relationships between the two vectors, as we will see later.

A useful property of inner products is the *Cauchy-Schwarz inequality*:

$$\langle x, y\rangle \leq \langle x, x\rangle^{\frac{1}{2}} \langle y, y\rangle^{\frac{1}{2}}. \tag{2.26}$$

This relationship is also sometimes called the Cauchy-Bunyakovskii-Schwarz inequality. (Augustin-Louis Cauchy gave the inequality for the kind of discrete inner products we are considering here, and Viktor Bunyakovskii and Hermann Schwarz independently extended it to more general inner products, defined on functions, for example.) The inequality is easy to see, by first observing that for every real number t,

$$\begin{aligned} 0 &\leq \langle (tx + y), (tx + y)\rangle \\ &= \langle x, x\rangle t^2 + 2\langle x, y\rangle t + \langle y, y\rangle \\ &= at^2 + bt + c, \end{aligned}$$

where the constants a, b, and c correspond to the dot products in the preceding equation. This quadratic in t cannot have two distinct real roots. Hence the discriminant, $b^2 - 4ac$, must be less than or equal to zero; that is,

$$\left(\frac{1}{2}b\right)^2 \leq ac.$$

By substituting and taking square roots, we get the Cauchy-Schwarz inequality. It is also clear from this proof that equality holds only if $x = 0$ or if $y = rx$, for some scalar r.

Two vectors x and y such that $\langle x, y\rangle = 0$ are said to be *orthogonal*. This term has such an intuitive meaning that we may use it prior to a careful definition and study, so I only introduce it here. We will discuss orthogonality more thoroughly in Sect. 2.1.8 beginning on page 33.

2.1.5 Norms

We consider a set of objects S that has an addition-type operator, $+$, a corresponding additive identity, 0, and a scalar multiplication; that is, a multiplication of the objects by a real (or complex) number. On such a set, a *norm* is a function, $\|\cdot\|$, from S to $\mathbb{R}$ that satisfies the following three conditions:

1. Nonnegativity and mapping of the additive identity:
   if $x \neq 0$, then $\|x\| > 0$, and $\|0\| = 0$.
2. Relation of scalar multiplication to real multiplication:
   $\|ax\| = |a|\,\|x\|$ for real a.
3. Triangle inequality:
   $\|x + y\| \leq \|x\| + \|y\|$.

(If property 1 is relaxed to require only $\|x\| \geq 0$ for $x \neq 0$, the function is called a *seminorm*.) Because a norm is a function whose argument is a vector, we also often use a functional notation such as $\rho(x)$ to represent a norm of the vector x.

Sets of various types of objects (functions, for example) can have norms, but our interest in the present context is in norms for vectors and (later) for matrices. (The three properties above in fact define a more general norm for other kinds of mathematical objects for which an addition, an additive identity, and multiplication by a scalar are defined. Norms are defined for matrices, as we will discuss later. Note that there are two different kinds of multiplication used in property 2 and two different kinds of addition used in property 3.)

A vector space together with a norm is called a *normed space*.

For some types of objects, a norm of an object may be called its "length" or its "size". (Recall the ambiguity of "length" of a vector that we mentioned at the beginning of this chapter.)

2.1.5.1 Convexity

A function $f(\cdot)$ over a convex domain S into a range R, where both S and R have an addition-type operator, $+$, corresponding additive identities, and scalar multiplication, is said to be *convex*, if, for any x and y in S, and a such that $0 \leq a \leq 1$,

$$f(ax + (1-a)y) \leq af(x) + (1-a)f(y). \tag{2.27}$$

If, for $x \neq y$ and a such that $0 < a < 1$, the inequality in (2.27) is sharp, then the function is said to be *strictly convex*.

It is clear from the triangle inequality that a norm is convex.

2.1.5.2 Norms Induced by Inner Products

There is a close relationship between a norm and an inner product. For any inner product space with inner product $\langle \cdot, \cdot \rangle$, a norm of an element of the space can be defined in terms of the square root of the inner product of the element with itself:

$$\|x\| = \sqrt{\langle x, x \rangle}. \tag{2.28}$$

Any function $\|\cdot\|$ defined in this way satisfies the properties of a norm. It is easy to see that $\|x\|$ satisfies the first two properties of a norm, nonnegativity and scalar equivariance. Now, consider the square of the right-hand side of the triangle inequality, $\|x\| + \|y\|$:

$$\begin{aligned}(\|x\| + \|y\|)^2 &= \langle x, x \rangle + 2\sqrt{\langle x, x \rangle \langle y, y \rangle} + \langle y, y \rangle \\ &\geq \langle x, x \rangle + 2\langle x, y \rangle + \langle y, y \rangle \\ &= \langle x + y,\, x + y \rangle \\ &= \|x + y\|^2; \end{aligned} \tag{2.29}$$

hence, the triangle inequality holds. Therefore, given an inner product, $\langle x, y \rangle$, then $\sqrt{\langle x, x \rangle}$ is a norm.

Equation (2.28) defines a norm given any inner product. It is called the *norm induced by the inner product*.

Norms induced by inner products have some interesting properties. First of all, they have the Cauchy-Schwarz relationship (inequality (2.26)) with their associated inner product:

$$|\langle x, y \rangle| \leq \|x\| \|y\|. \tag{2.30}$$

In the sequence of equations above for an induced norm of the sum of two vectors, one equation (expressed differently) stands out as particularly useful in later applications:

$$\|x + y\|^2 = \|x\|^2 + \|y\|^2 + 2\langle x, y \rangle. \tag{2.31}$$

If $\langle x, y\rangle = 0$ (that is, the vectors are orthogonal), equation (2.31) becomes the *Pythagorean theorem*:

$$\|x + y\|^2 = \|x\|^2 + \|y\|^2.$$

Another useful property of a norm induced by an inner product is the *parallelogram equality*:

$$2\|x\|^2 + 2\|y\|^2 = \|x + y\|^2 + \|x - y\|^2. \tag{2.32}$$

This is trivial to show, and you are asked to do so in Exercise 2.7. (It is also the case that if the parallelogram equality holds for every pair of vectors in the space, then the norm is necessarily induced by an inner product. This fact is both harder to show and less useful than its converse; I state it only because it is somewhat surprising.)

A vector space whose norm is induced by an inner product has an interesting structure; for example, the geometric properties such as projections, orthogonality, and angles between vectors that we discuss in Sect. 2.2 are defined in terms of inner products and the associated norm.

2.1.5.3 L_p Norms

There are many norms that could be defined for vectors. One type of norm is called an L_p norm, often denoted as $\|\cdot\|_p$. For $p \geq 1$, it is defined as

$$\|x\|_p = \left(\sum_i |x_i|^p\right)^{\frac{1}{p}}. \tag{2.33}$$

This is also sometimes called the *Minkowski norm* and also the *Hölder norm*. An L_p norm is also called a p-norm, or 1-norm, 2-norm, or ∞-norm (defined by a limit) in those special cases.

It is easy to see that the L_p norm satisfies the first two conditions above. For general $p \geq 1$ it is somewhat more difficult to prove the triangular inequality (which for the L_p norms is also called the Minkowski inequality), but for some special cases it is straightforward, as we will see below.

The most common L_p norms, and in fact the most commonly used vector norms, are:

- $\|x\|_1 = \sum_i |x_i|$, also called the *Manhattan norm* because it corresponds to sums of distances along coordinate axes, as one would travel along the rectangular street plan of Manhattan (except for Broadway and a few other streets and avenues).
- $\|x\|_2 = \sqrt{\sum_i x_i^2}$, also called the *Euclidean norm*, the *Euclidean length*, or just the *length* of the vector. The L_2 norm is induced by an inner product; it is the square root of the inner product of the vector with itself: $\|x\|_2 = \sqrt{\langle x, x\rangle}$. It is the only L_p norm induced by an inner product. (See Exercise 2.9.)

- $\|x\|_\infty = \max_i |x_i|$, also called the *max norm* or the *Chebyshev norm*. The L_∞ norm is defined by taking the limit in an L_p norm, and we see that it is indeed $\max_i |x_i|$ by expressing it as

$$\|x\|_\infty = \lim_{p\to\infty} \|x\|_p = \lim_{p\to\infty} \left(\sum_i |x_i|^p\right)^{\frac{1}{p}} = m \lim_{p\to\infty} \left(\sum_i \left|\frac{x_i}{m}\right|^p\right)^{\frac{1}{p}}$$

  with $m = \max_i |x_i|$. Because the quantity of which we are taking the p^{th} root is bounded above by the number of elements in x and below by 1, that factor goes to 1 as p goes to ∞.

It is easy to see that, for any n-vector x, the L_p norms have the relationships

$$\|x\|_\infty \leq \|x\|_2 \leq \|x\|_1. \tag{2.34}$$

More generally, for given x and for $p \geq 1$, we see that $\|x\|_p$ is a nonincreasing function of p.

We also have bounds that involve the number of elements in the vector:

$$\|x\|_\infty \leq \|x\|_2 \leq \sqrt{n}\|x\|_\infty, \tag{2.35}$$

and

$$\|x\|_2 \leq \|x\|_1 \leq \sqrt{n}\|x\|_2. \tag{2.36}$$

The triangle inequality obviously holds for the L_1 and L_∞ norms. For the L_2 norm it can be seen by expanding $\sum(x_i + y_i)^2$ and then using the Cauchy-Schwarz inequality (2.26) on page 24. Rather than approaching it that way, however, we will show below that the L_2 norm can be defined in terms of an inner product, and then we will establish the triangle inequality for any norm defined similarly by an inner product; see inequality (2.29). Showing that the triangle inequality holds for other L_p norms is more difficult; see Exercise 2.11.

A generalization of the L_p vector norm is the *weighted* L_p *vector norm* defined by

$$\|x\|_{wp} = \left(\sum_i w_i |x_i|^p\right)^{\frac{1}{p}}, \tag{2.37}$$

where $w_i \geq 0$ and $\sum_i w_i = 1$.

In the following, if we use the unqualified symbol $\|\cdot\|$ for a vector norm and do not state otherwise, we will mean the L_2 norm; that is, the Euclidean norm, the induced norm.

2.1.5.4 Basis Norms

If $\{v_1, \ldots, v_k\}$ is a basis for a vector space that includes a vector x with $x = c_1 v_1 + \cdots + c_k v_k$, then

$$\rho(x) = \left(\sum_i c_i^2 \right)^{\frac{1}{2}} \tag{2.38}$$

is a norm. It is straightforward to see that $\rho(x)$ is a norm by checking the following three conditions:

- $\rho(x) \geq 0$ and $\rho(x) = 0$ if and only if $x = 0$ because $x = 0$ if and only if $c_i = 0$ for all i.
- $\rho(ax) = \left(\sum_i a^2 c_i^2 \right)^{\frac{1}{2}} = |a| \left(\sum_i c_i^2 \right)^{\frac{1}{2}} = |a| \rho(x)$.
- If also $y = b_1 v_1 + \cdots + b_k v_k$, then

$$\rho(x+y) = \left(\sum_i (c_i + b_i)^2 \right)^{\frac{1}{2}} \leq \left(\sum_i c_i^2 \right)^{\frac{1}{2}} + \left(\sum_i b_i^2 \right)^{\frac{1}{2}} = \rho(x) + \rho(y).$$

The last inequality is just the triangle inequality for the L_2 norm for the vectors $(c_1, \cdots, c_k)$ and $(b_1, \cdots, b_k)$.

In Sect. 2.2.5, we will consider special forms of basis sets in which the norm in equation (2.38) is identically the L_2 norm. (This is called Parseval's identity, equation (2.60) on page 41.)

2.1.5.5 Equivalence of Norms

There is an equivalence among any two norms over a normed finite-dimensional linear space in the sense that if $\| \cdot \|_a$ and $\| \cdot \|_b$ are norms, then there are positive numbers r and s such that for any x in the space,

$$r\|x\|_b \leq \|x\|_a \leq s\|x\|_b. \tag{2.39}$$

Expressions (2.35) and (2.36) are examples of this general equivalence for three L_p norms.

We can prove inequality (2.39) by using the norm defined in equation (2.38). We need only consider the case $x \neq 0$, because the inequality is obviously true if $x = 0$. Let $\| \cdot \|_a$ be any norm over a given normed linear space and let $\{v_1, \ldots, v_k\}$ be a basis for the space. (Here's where the assumption of a vector space with finite dimensions comes in.) Any x in the space has a representation in terms of the basis, $x = c_1 v_1 + \cdots + c_k v_k$. Then

$$\|x\|_a = \left\| \sum_{i=1}^k c_i v_i \right\|_a \leq \sum_{i=1}^k |c_i| \, \|v_i\|_a.$$

Applying the Cauchy-Schwarz inequality to the two vectors $(c_1, \cdots, c_k)$ and $(\|v_1\|_a, \cdots, \|v_k\|_a)$, we have

$$\sum_{i=1}^k |c_i| \, \|v_i\|_a \leq \left(\sum_{i=1}^k c_i^2 \right)^{\frac{1}{2}} \left(\sum_{i=1}^k \|v_i\|_a^2 \right)^{\frac{1}{2}}.$$

Hence, with $\tilde{s} = (\sum_i \|v_i\|_a^2)^{\frac{1}{2}}$, which must be positive, we have

$$\|x\|_a \leq \tilde{s}\rho(x).$$

Now, to establish a lower bound for $\|x\|_a$, let us define a subset C of the linear space consisting of all vectors $(u_1, \ldots, u_k)$ such that $\sum |u_i|^2 = 1$. This set is obviously closed. Next, we define a function $f(\cdot)$ over this closed subset by

$$f(u) = \left\| \sum_{i=1}^{k} u_i v_i \right\|_a .$$

Because f is continuous, it attains a minimum in this closed subset, say for the vector u_*; that is, $f(u_*) \leq f(u)$ for any u such that $\sum |u_i|^2 = 1$. Let

$$\tilde{r} = f(u_*),$$

which must be positive, and again consider any x in the normed linear space and express it in terms of the basis, $x = c_1 v_1 + \cdots c_k v_k$. If $x \neq 0$, we have

$$\begin{aligned} \|x\|_a &= \left\| \sum_{i=1}^{k} c_i v_i \right\|_a \\ &= \left(\sum_{i=1}^{k} c_i^2 \right)^{\frac{1}{2}} \left\| \sum_{i=1}^{k} \left(\frac{c_i}{\left(\sum_{i=1}^{k} c_i^2 \right)^{\frac{1}{2}}} \right) v_i \right\|_a \\ &= \rho(x) f(\tilde{c}), \end{aligned}$$

where $\tilde{c} = (c_1, \cdots, c_k)/(\sum_{i=1}^{k} c_i^2)^{1/2}$. Because $\tilde{c}$ is in the set C, $f(\tilde{c}) \geq r$; hence, combining this with the inequality above, we have

$$\tilde{r}\rho(x) \leq \|x\|_a \leq \tilde{s}\rho(x).$$

This expression holds for any norm $\|\cdot\|_a$ and so, after obtaining similar bounds for any other norm $\|\cdot\|_b$ and then combining the inequalities for $\|\cdot\|_a$ and $\|\cdot\|_b$, we have the bounds in the equivalence relation (2.39). (This is an equivalence relation because it is reflexive, symmetric, and transitive. Its transitivity is seen by the same argument that allowed us to go from the inequalities involving $\rho(\cdot)$ to ones involving $\|\cdot\|_b$.)

As we have mentioned, there are some differences in the properties of vector spaces that have an infinite number of dimensions and those with finite dimensions. The equivalence of norms is one of those differences. The argument above fails in the properties of the continuous function f. (Recall, however, as we have mentioned, unless we state otherwise, we assume that the vector spaces we discuss have finite dimensions.)

2.1.6 Normalized Vectors

The Euclidean norm of a vector corresponds to the length of the vector x in a natural way; that is, it agrees with our intuition regarding "length". Although, as we have seen, this is just one of many vector norms, in most applications it is the most useful one. (I must warn you, however, that occasionally I will carelessly but naturally use "length" to refer to the order of a vector; that is, the number of elements. This usage is common in computer software packages such as R and SAS IML, and software necessarily shapes our vocabulary.)

Dividing a given vector by its length *normalizes* the vector, and the resulting vector with length 1 is said to be *normalized*; thus

$$\tilde{x} = \frac{1}{\|x\|}x \tag{2.40}$$

is a normalized vector. Normalized vectors are sometimes referred to as "unit vectors", although we will generally reserve this term for a special kind of normalized vector (see page 16). A normalized vector is also sometimes referred to as a "normal vector". I use "normalized vector" for a vector such as $\tilde{x}$ in equation (2.40) and use "normal vector" to denote a vector that is orthogonal to a subspace (as on page 34).

2.1.6.1 "Inverse" of a Vector

Because the mapping of an inner product takes the elements of one space into a different space (the inner product of vectors takes elements of $\mathbb{R}^n$ into $\mathbb{R}$), the concept of an inverse for the inner product does not make sense in the usual way. First of all, there is no identity with respect to the inner product.

Often in applications, however, inner products are combined with the usual scalar-vector multiplication in the form $\langle x, y\rangle z$; therefore, for given x, it may be of interest to determine y such that $\langle x, y\rangle = 1$, the multiplicative identity in $\mathbb{R}$. For x in $\mathbb{R}^n$ such that $x \neq 0$, the additive identity in $\mathbb{R}^n$,

$$y = \frac{1}{\|x\|^2}x = \frac{1}{\|x\|}\tilde{x} \tag{2.41}$$

uniquely satisfies $\langle x, y\rangle = 1$. Such a y is called the *Samelson inverse* of the vector x and is sometimes denoted as x^{-1} or as $[x]^{-1}$. It is also sometimes called the Moore-Penrose vector inverse because it satisfies the four properties of the definition the Moore-Penrose inverse. (See page 128, where, for example, the first property is interpreted both as $\langle x, [x]^{-1}\rangle x$ and as $x\langle [x]^{-1}, x\rangle$.)

The norm in equation (2.41) is obviously the Euclidean norm (because of the way we defined $\tilde{x}$), but the idea of the inverse could also be extended to other norms associated with other inner products.

The Samelson inverse has a nice geometric interpretation: it is the inverse point of x with respect to the unit sphere in $\mathbb{R}^n$. This inverse arises in the vector ϵ-algorithm used in accelerating convergence of vector sequences in numerical computations (see Wynn 1962).

2.1.7 Metrics and Distances

It is often useful to consider how far apart two objects are; that is, the "distance" between them. A reasonable distance measure would have to satisfy certain requirements, such as being a nonnegative real number.

A function Δ that maps any two objects in a set S to $\mathbb{R}$ is called a *metric* on S if, for all x, y, and z in S, it satisfies the following three conditions:

1. $\Delta(x, y) > 0$ if $x \neq y$ and $\Delta(x, y) = 0$ if $x = y$;
2. $\Delta(x, y) = \Delta(y, x)$;
3. $\Delta(x, y) \leq \Delta(x, z) + \Delta(z, y)$.

These conditions correspond in an intuitive manner to the properties we expect of a distance between objects.

A vector space together with a metric defined on it is called a *metric space*. A normed vector space is a metric space because the norm can induce a metric. In the following, we may speak almost interchangeably of an inner product space, a normed space, or a metric space, but we must recognize that none is a special case of another. (Recall that a normed space whose norm is the L_1 norm is not equivalent to an inner product space, for example.)

2.1.7.1 Metrics Induced by Norms

If subtraction and a norm are defined for the elements of S, the most common way of forming a metric is by using the norm. If $\|\cdot\|$ is a norm, we can verify that

$$\Delta(x, y) = \|x - y\| \tag{2.42}$$

is a metric by using the properties of a norm to establish the three properties of a metric above (Exercise 2.12).

The norm in equation (2.42) may, of course, be induced by an inner product.

The general inner products, norms, and metrics defined above are relevant in a wide range of applications. The sets on which they are defined can consist of various types of objects. In the context of real vectors, the most common inner product is the dot product; the most common norm is the Euclidean norm that arises from the dot product; and the most common metric is the one defined by the Euclidean norm, called the Euclidean distance.

2.1.7.2 Convergence of Sequences of Vectors

A sequence of real numbers $a_1, a_2, \ldots$ is said to converge to a finite number a if for any given $\epsilon > 0$ there is an integer M such that, for $k > M$, $|a_k - a| < \epsilon$, and we write $\lim_{k\to\infty} a_k = a$, or we write $a_k \to a$ as $k \to \infty$.

We define convergence of a sequence of vectors in a normed vector space in terms of the convergence of a sequence of their norms, which is a sequence of

real numbers. We say that a sequence of vectors $x_1, x_2, \ldots$ (of the same order) converges to the vector x with respect to the norm $\|\cdot\|$ if the sequence of real numbers $\|x_1 - x\|, \|x_2 - x\|, \ldots$ converges to 0. Because of the bounds (2.39), the choice of the norm is irrelevant (for finite dimensional vector spaces), and so convergence of a sequence of vectors is well-defined without reference to a specific norm. (This is one reason that equivalence of norms is an important property.)

A sequence of vectors $x_1, x_2, \ldots$ in the metric space $\mathcal{V}$ that come arbitrarily close to one another (as measured by the given metric) is called a *Cauchy sequence*. (In a Cauchy sequence $x_1, x_2, \ldots$, for any $\epsilon > 0$ there is a number N such that for $i, j > N$, $\Delta(x_i, x_j) < \epsilon$.) Intuitively, such a sequence should converge to some fixed vector in $\mathcal{V}$, but this is not necessarily the case. A metric space in which every Cauchy sequence converges to an element in the space is said to be a *complete metric space* or just a *complete space*. The space $\mathbb{R}^n$ (with any norm) is complete.

A complete normed space is called a *Banach space*, and a complete inner product space is called a *Hilbert space*. It is clear that a Hilbert space is a Banach space (because its inner product induces a norm). As we have indicated, a space with a norm induced by an inner product, such as a Hilbert space, has an interesting structure. Most of the vector spaces encountered in statistical applications are Hilbert spaces. The space $\mathbb{R}^n$ with the L_2 norm is a Hilbert space.

2.1.8 Orthogonal Vectors and Orthogonal Vector Spaces

Two vectors v_1 and v_2 such that

$$\langle v_1, v_2 \rangle = 0 \tag{2.43}$$

are said to be *orthogonal*, and this condition is denoted by $v_1 \perp v_2$. (Sometimes we exclude the zero vector from this definition, but it is not important to do so.) Normalized vectors that are all orthogonal to each other are called *orthonormal* vectors.

An Aside: Complex Vectors

If the elements of the vectors are from the field of complex numbers, orthogonality and normality are also defined as above; however, the inner product in the definition (2.43) must be as defined in equation (2.23), and the expression $x^{\mathrm{T}}y$ in equation (2.25) is not equivalent to the inner product. We will use a different notation in this case: $x^{\mathrm{H}}y$. The relationship between the two notations is

$$x^{\mathrm{H}}y = \bar{x}^{\mathrm{T}}y.$$

With this interpretation of the inner product, all of the statements below about orthogonality hold for complex numbers as well as for real numbers.

A set of nonzero vectors that are mutually orthogonal are necessarily linearly independent. To see this, we show it for any two orthogonal vectors and then indicate the pattern that extends to three or more vectors. First, suppose v_1 and v_2 are nonzero and are orthogonal; that is, $\langle v_1, v_2\rangle = 0$. We see immediately that if there is a scalar a such that $v_1 = av_2$, then a must be nonzero and we have a contradiction because $\langle v_1, v_2\rangle = a\langle v_2, v_2\rangle \neq 0$. Hence, we conclude v_1 and v_2 are independent (there is no a such that $v_1 = av_2$). For three mutually orthogonal vectors, v_1, v_2, and v_3, we consider $v_1 = av_2 + bv_3$ for a or b nonzero, and arrive at the same contradiction.

Two vector spaces $\mathcal{V}_1$ and $\mathcal{V}_2$ are said to be *orthogonal*, written $\mathcal{V}_1 \perp \mathcal{V}_2$, if each vector in one is orthogonal to every vector in the other. If $\mathcal{V}_1 \perp \mathcal{V}_2$ and $\mathcal{V}_1 \oplus \mathcal{V}_2 = \mathbb{R}^n$, then $\mathcal{V}_2$ is called the *orthogonal complement* of $\mathcal{V}_1$, and this is written as $\mathcal{V}_2 = \mathcal{V}_1^\perp$. More generally, if $\mathcal{V}_1 \perp \mathcal{V}_2$ and $\mathcal{V}_1 \oplus \mathcal{V}_2 = \mathcal{V}$, then $\mathcal{V}_2$ is called the orthogonal complement of $\mathcal{V}_1$ with respect to $\mathcal{V}$. This is obviously a symmetric relationship; if $\mathcal{V}_2$ is the orthogonal complement of $\mathcal{V}_1$, then $\mathcal{V}_1$ is the orthogonal complement of $\mathcal{V}_2$.

A vector that is orthogonal to all vectors in a given vector space is said to be orthogonal to that space or *normal* to that space. Such a vector is called a *normal vector* to that space.

If B_1 is a basis set for $\mathcal{V}_1$, B_2 is a basis set for $\mathcal{V}_2$, and $\mathcal{V}_2$ is the orthogonal complement of $\mathcal{V}_1$ with respect to $\mathcal{V}$, then $B_1 \cup B_2$ is a basis set for $\mathcal{V}$. It is a basis set because since $\mathcal{V}_1$ and $\mathcal{V}_2$ are orthogonal, it must be the case that $B_1 \cap B_2 = \emptyset$. (See the properties listed on page 22.)

If $\mathcal{V}_1 \subset \mathcal{V}$, $\mathcal{V}_2 \subset \mathcal{V}$, $\mathcal{V}_1 \perp \mathcal{V}_2$, and $\dim(\mathcal{V}_1) + \dim(\mathcal{V}_2) = \dim(\mathcal{V})$, then

$$\mathcal{V}_1 \oplus \mathcal{V}_2 = \mathcal{V}; \tag{2.44}$$

that is, $\mathcal{V}_2$ is the orthogonal complement of $\mathcal{V}_1$. We see this by first letting B_1 and B_2 be bases for $\mathcal{V}_1$ and $\mathcal{V}_2$. Now $\mathcal{V}_1 \perp \mathcal{V}_2$ implies that $B_1 \cap B_2 = \emptyset$ and $\dim(\mathcal{V}_1) + \dim(\mathcal{V}_2) = \dim(\mathcal{V})$ implies $\#(B_1) + \#(B_2) = \#(B)$, for any basis set B for $\mathcal{V}$; hence, $B_1 \cup B_2$ is a basis set for $\mathcal{V}$.

The intersection of two orthogonal vector spaces consists only of the zero vector (Exercise 2.14).

A set of linearly independent vectors can be mapped to a set of mutually orthogonal (and orthonormal) vectors by means of the Gram-Schmidt transformations (see equation (2.56) below).

2.1.9 The "One Vector"

The vector with all elements equal to 1 that we mentioned previously is useful in various vector operations. We call this the "one vector" and denote it by 1 or by 1_n. The one vector can be used in the representation of the sum of the elements in a vector:

$$1^{\mathrm{T}}x = \sum x_i. \tag{2.45}$$

The one vector is also called the "summing vector".

2.1.9.1 The Mean and the Mean Vector

Because the elements of x are real, they can be summed; however, in applications it may or may not make sense to add the elements in a vector, depending on what is represented by those elements. If the elements have some kind of essential commonality, it may make sense to compute their sum as well as their *arithmetic mean*, which for the n-vector x is denoted by $\bar{x}$ and defined by

$$\bar{x} = 1_n^{\mathrm{T}} x/n. \tag{2.46}$$

We also refer to the arithmetic mean as just the "mean" because it is the most commonly used mean.

It is often useful to think of the mean as an n-vector all of whose elements are $\bar{x}$. The symbol $\bar{x}$ is also used to denote this vector; hence, we have

$$\bar{x} = \bar{x} 1_n, \tag{2.47}$$

in which $\bar{x}$ on the left-hand side is a vector and $\bar{x}$ on the right-hand side is a scalar. We also have, for the two different objects,

$$\|\bar{x}\|^2 = n\bar{x}^2. \tag{2.48}$$

The meaning, whether a scalar or a vector, is usually clear from the context. In any event, an expression such as $x - \bar{x}$ is unambiguous; the addition (subtraction) has the same meaning whether $\bar{x}$ is interpreted as a vector or a scalar. (In some mathematical treatments of vectors, addition of a scalar to a vector is not defined, but here we are following the conventions of modern computer languages.)

2.2 Cartesian Coordinates and Geometrical Properties of Vectors

Points in a Cartesian geometry can be identified with vectors, and several definitions and properties of vectors can be motivated by this geometric interpretation. In this interpretation, vectors are directed line segments with a common origin. The geometrical properties can be seen most easily in terms of a Cartesian coordinate system, but the properties of vectors defined in terms of a Cartesian geometry have analogues in Euclidean geometry without a coordinate system. In such a system, only length and direction are defined, and two vectors are considered to be the same vector if they have the same length and direction. Generally, we will not assume that there is a "location" or "position" associated with a vector.

2.2.1 Cartesian Geometry

A Cartesian coordinate system in d dimensions is defined by d unit vectors, e_i in equation (2.6), each with d elements. A unit vector is also called a *principal axis* of the coordinate system. The set of unit vectors is orthonormal. (There is an implied number of elements of a unit vector that is inferred from the context. Also parenthetically, we remark that the phrase "unit vector" is sometimes used to refer to a vector the sum of whose squared elements is 1, that is, whose length, in the Euclidean distance sense, is 1. As we mentioned above, we refer to this latter type of vector as a "normalized vector".)

The sum of all of the unit vectors is the one vector:

$$\sum_{i=1}^{d} e_i = 1_d. \tag{2.49}$$

A point x with Cartesian coordinates $(x_1, \ldots, x_d)$ is associated with a vector from the origin to the point, that is, the vector $(x_1, \ldots, x_d)$. The vector can be written as the linear combination

$$x = x_1 e_1 + \ldots + x_d e_d \tag{2.50}$$

or, equivalently, as

$$x = \langle x, e_1 \rangle e_1 + \ldots + \langle x, e_d \rangle e_d.$$

(This is a Fourier expansion, equation (2.58) below.)

2.2.2 Projections

The *projection* of the vector y onto the nonnull vector x is the vector

$$\hat{y} = \frac{\langle x, y \rangle}{\|x\|^2} x. \tag{2.51}$$

This definition is consistent with a geometrical interpretation of vectors as directed line segments with a common origin. The projection of y onto x is the inner product of the normalized x and y times the normalized x; that is, $\langle \tilde{x}, y \rangle \tilde{x}$, where $\tilde{x} = x/\|x\|$. Notice that the order of y and x is the same.

An important property of a projection is that when it is subtracted from the vector that was projected, the resulting vector, called the "residual", is orthogonal to the projection; that is, if

$$\begin{aligned} r &= y - \frac{\langle x, y \rangle}{\|x\|^2} x \\ &= y - \hat{y} \end{aligned} \tag{2.52}$$

then r and $\hat{y}$ are orthogonal, as we can easily see by taking their inner product (see Fig. 2.2.2). Notice also that the Pythagorean relationship holds:

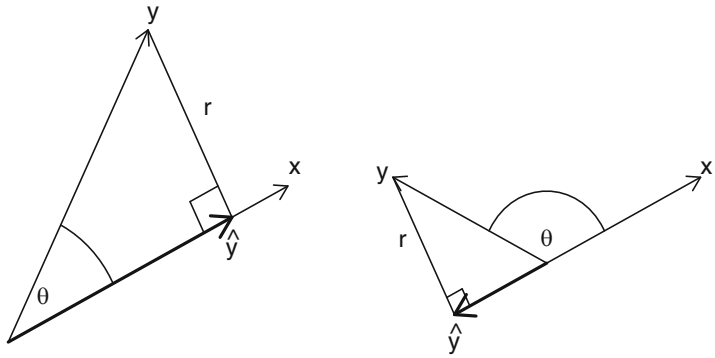


Figure 2.1. Projections and angles

$$\|y\|^2 = \|\hat{y}\|^2 + \|r\|^2. \tag{2.53}$$

As we mentioned on page 35, the mean $\bar{y}$ can be interpreted either as a scalar or as a vector all of whose elements are $\bar{y}$. As a vector, it is the projection of y onto the one vector 1_n,

$$\begin{aligned} \frac{\langle 1_n, y \rangle}{\|1_n\|^2} \, 1_n &= \frac{1_n^{\mathrm{T}} y}{n} \, 1_n \\ &= \bar{y} \, 1_n, \end{aligned}$$

from equations (2.46) and (2.51).

We will consider more general projections (that is, projections onto planes or other subspaces) on page 352, and on page 409 we will view linear regression fitting as a projection onto the space spanned by the independent variables.

2.2.3 Angles Between Vectors

The *angle* between the nonnull vectors x and y is determined by its cosine, which we can compute from the length of the projection of one vector onto the other. Hence, denoting the angle between the nonnull vectors x and y as $\mathrm{angle}(x, y)$, we define

$$\mathrm{angle}(x, y) = \cos^{-1}\left(\frac{\langle x, y \rangle}{\|x\|\|y\|}\right), \tag{2.54}$$

with $\cos^{-1}(\cdot)$ being taken in the interval $[0, \pi]$. The cosine is $\pm\|\hat{y}\|/\|y\|$, with the sign chosen appropriately; see Fig. 2.2.2. Because of this choice of $\cos^{-1}(\cdot)$, we have that $\mathrm{angle}(y, x) = \mathrm{angle}(x, y)$—but see Exercise 2.19e on page 54.

The word "orthogonal" is appropriately defined by equation (2.43) on page 33 because orthogonality in that sense is equivalent to the corresponding geometric property. (The cosine is 0.)

Notice that the angle between two vectors is invariant to scaling of the vectors; that is, for any positive scalar a, $\text{angle}(ax, y) = \text{angle}(x, y)$.

A given vector can be defined in terms of its length and the angles θ_i that it makes with the unit vectors. The cosines of these angles are just the scaled coordinates of the vector:

$$\begin{aligned} \cos(\theta_i) &= \frac{\langle x, e_i \rangle}{\|x\| \|e_i\|} \\ &= \frac{1}{\|x\|} x_i. \end{aligned} \tag{2.55}$$

These quantities are called the *direction cosines* of the vector.

Although geometrical intuition often helps us in understanding properties of vectors, sometimes it may lead us astray in high dimensions. Consider the direction cosines of an arbitrary vector in a vector space with large dimensions. If the elements of the arbitrary vector are nearly equal (that is, if the vector is a diagonal through an orthant of the coordinate system), the direction cosine goes to 0 as the dimension increases. In high dimensions, any two vectors are "almost orthogonal" to each other; see Exercise 2.16.

The geometric property of the angle between vectors has important implications for certain operations both because it may indicate that rounding in computations will have deleterious effects and because it may indicate a deficiency in the understanding of the application.

We will consider more general projections and angles between vectors and other subspaces on page 359. In Sect. 5.3.3, we will consider rotations of vectors onto other vectors or subspaces. Rotations are similar to projections, except that the length of the vector being rotated is preserved.

2.2.4 Orthogonalization Transformations: Gram-Schmidt

Given m nonnull, linearly independent vectors, $x_1, \ldots, x_m$, it is easy to form m orthonormal vectors, $\tilde{x}_1, \ldots, \tilde{x}_m$, that span the same space. A simple way to do this is sequentially. First normalize x_1 and call this $\tilde{x}_1$. Next, project x_2 onto $\tilde{x}_1$ and subtract this projection from x_2. The result is orthogonal to $\tilde{x}_1$; hence, normalize this and call it $\tilde{x}_2$. These first two steps, which are illustrated in Fig. 2.2.4, are

$$\begin{aligned} \tilde{x}_1 &= \frac{1}{\|x_1\|} x_1, \\ \tilde{x}_2 &= \frac{1}{\|x_2 - \langle \tilde{x}_1, x_2 \rangle \tilde{x}_1\|} (x_2 - \langle \tilde{x}_1, x_2 \rangle \tilde{x}_1). \end{aligned} \tag{2.56}$$

These are called *Gram-Schmidt transformations*.

The Gram-Schmidt transformations have a close relationship to least squares fitting of overdetermined systems of linear equations and to least

squares fitting of linear regression models. For example, if equation (6.33) on page 290 is merely the system of equations in one unknown $x_1 b = x_2 - r$, and it is approximated by least squares, then the "residual vector" r is $\tilde{x}_2$ above, and of course it has the orthogonality property shown in equation (6.37) for that problem.

The Gram-Schmidt transformations can be continued with all of the vectors in the linearly independent set. There are two straightforward ways equations (2.56) can be extended. One method generalizes the second equation in an obvious way:

for $k = 2, 3 \ldots,$

$$\tilde{x}_k = \left(x_k - \sum_{i=1}^{k-1} \langle \tilde{x}_i, x_k \rangle \tilde{x}_i \right) \Bigg/ \left\| x_k - \sum_{i=1}^{k-1} \langle \tilde{x}_i, x_k \rangle \tilde{x}_i \right\|. \tag{2.57}$$

In this method, at the k^{th} step, we orthogonalize the k^{th} vector by computing its residual with respect to the plane formed by all the previous $k-1$ orthonormal vectors.

Another way of extending the transformation of equations (2.56) is, at the k^{th} step, to compute the residuals of all remaining vectors with respect just to the k^{th} normalized vector. If the initial set of vectors are linearly independent, the residuals at any stage will be nonzero. (This is fairly obvious, but you are asked to show it in Exercise 2.17.) We describe this method explicitly in Algorithm 2.1.

Algorithm 2.1 Gram-Schmidt orthonormalization of a set of linearly independent vectors, $x_1, \ldots, x_m$

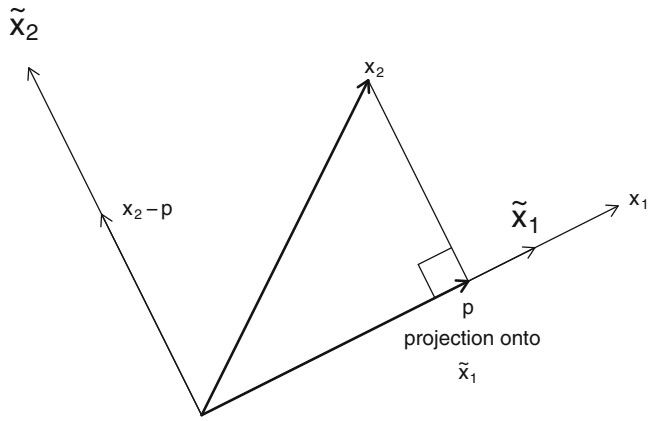


Figure 2.2. Orthogonalization of x_1 and x_2

0. For $k = 1, \ldots, m$,
   {
   set $\tilde{x}_k = x_k$.
   }
1. Ensure that $\tilde{x}_1 \neq 0$;
   set $\tilde{x}_1 = \tilde{x}_1 / \|\tilde{x}_1\|$.
2. If $m > 1$, for $k = 2, \ldots, m$,
   {
   for $j = k, \ldots, m$,
   {
   set $\tilde{x}_j = \tilde{x}_j - \langle \tilde{x}_{k-1}, \tilde{x}_j \rangle \tilde{x}_{k-1}$.
   }
   ensure that $\tilde{x}_k \neq 0$;
   set $\tilde{x}_k = \tilde{x}_k / \|\tilde{x}_k\|$.
   }

Although the method indicated in equation (2.57) is mathematically equivalent to this method, the use of Algorithm 2.1 is to be preferred for computations because it is less subject to rounding errors. (This may not be immediately obvious, although a simple numerical example can illustrate the fact—see Exercise 11.1c on page 537. We will not digress here to consider this further, but the difference in the two methods has to do with the relative magnitudes of the quantities in the subtraction. The method of Algorithm 2.1 is sometimes called the "modified Gram-Schmidt method", although I call it the "Gram-Schmidt method". I will discuss this method again in Sect. 11.2.1.3.) This is an instance of a principle that we will encounter repeatedly: *the form of a mathematical expression and the way the expression should be evaluated in actual practice may be quite different.*

These orthogonalizing transformations result in a set of orthogonal vectors that span the same space as the original set. They are not unique; if the order in which the vectors are processed is changed, a different set of orthogonal vectors will result.

Orthogonal vectors are useful for many reasons: perhaps to improve the stability of computations; or in data analysis to capture the variability most efficiently; or for dimension reduction as in principal components analysis (see Sect. 9.4 beginning on page 424); or in order to form more meaningful quantities as in a vegetative index in remote sensing. We will discuss various specific orthogonalizing transformations later.

2.2.5 Orthonormal Basis Sets

A basis for a vector space is often chosen to be an orthonormal set because it is easy to work with the vectors in such a set.

If $u_1, \ldots, u_n$ is an orthonormal basis set for a space, then a vector x in that space can be expressed as

$$x = c_1 u_1 + \cdots + c_n u_n, \tag{2.58}$$

and because of orthonormality, we have

$$c_i = \langle x, u_i \rangle. \tag{2.59}$$

(We see this by taking the inner product of both sides with u_i.) A representation of a vector as a linear combination of orthonormal basis vectors, as in equation (2.58), is called a *Fourier expansion*, and the c_i are called *Fourier coefficients*.

By taking the inner product of each side of equation (2.58) with itself, we have *Parseval's identity*:

$$\|x\|^2 = \sum c_i^2. \tag{2.60}$$

This shows that the L_2 norm is the same as the norm in equation (2.38) (on page 29) for the case of an orthogonal basis.

Although the Fourier expansion is not unique because a different orthogonal basis set could be chosen, Parseval's identity removes some of the arbitrariness in the choice; no matter what basis is used, the sum of the squares of the Fourier coefficients is equal to the square of the norm that arises from the inner product. ("The" inner product means the inner product used in defining the orthogonality.)

Another useful expression of Parseval's identity in the Fourier expansion is

$$\left\| x - \sum_{i=1}^{k} c_i u_i \right\|^2 = \langle x, x \rangle - \sum_{i=1}^{k} c_i^2 \tag{2.61}$$

(because the term on the left-hand side is 0).

The expansion (2.58) is a special case of a very useful expansion in an orthogonal basis set. In the finite-dimensional vector spaces we consider here, the series is finite. In function spaces, the series is generally infinite, and so issues of convergence are important. For different types of functions, different orthogonal basis sets may be appropriate. Polynomials are often used, and there are some standard sets of orthogonal polynomials, such as Jacobi, Hermite, and so on. For periodic functions especially, orthogonal trigonometric functions are useful.

2.2.6 Approximation of Vectors

In high-dimensional vector spaces, it is often useful to approximate a given vector in terms of vectors from a lower dimensional space. Suppose, for example, that $\mathcal{V} \subseteq \mathbb{R}^n$ is a vector space of dimension k (necessarily, $k \leq n$) and x is a given n-vector, not necessarily in $\mathcal{V}$. We wish to determine a vector $\tilde{x}$ in $\mathcal{V}$ that approximates x. Of course if $\mathcal{V} = \mathbb{R}^n$, then $x \in \mathcal{V}$, and so the problem is not very interesting. The interesting case is when $\mathcal{V} \subset \mathbb{R}^n$.

2.2.6.1 Optimality of the Fourier Coefficients

The first question, of course, is what constitutes a "good" approximation. One obvious criterion would be based on a norm of the difference of the given vector and the approximating vector. So now, choosing the norm as the Euclidean norm, we may pose the problem as one of finding $\tilde{x} \in \mathcal{V}$ such that

$$\|x - \tilde{x}\| \leq \|x - v\| \quad \forall\, v \in \mathcal{V}. \tag{2.62}$$

This difference is a *truncation error*.

Let $u_1, \ldots, u_k$ be an orthonormal basis set for $\mathcal{V}$, and let

$$\tilde{x} = c_1 u_1 + \cdots + c_k u_k, \tag{2.63}$$

where the c_i are the Fourier coefficients of x, $\langle x,\, u_i \rangle$.

Now let $v = a_1 u_1 + \cdots + a_k u_k$ be any other vector in $\mathcal{V}$, and consider

$$\begin{aligned}
\|x - v\|^2 &= \left\| x - \sum_{i=1}^{k} a_i u_i \right\|^2 \\
&= \left\langle x - \sum_{i=1}^{k} a_i u_i,\; x - \sum_{i=1}^{k} a_i u_i \right\rangle \\
&= \langle x,\, x \rangle - 2 \sum_{i=1}^{k} a_i \langle x,\, u_i \rangle + \sum_{i=1}^{k} a_i^2 \\
&= \langle x,\, x \rangle - 2 \sum_{i=1}^{k} a_i c_i + \sum_{i=1}^{k} a_i^2 + \sum_{i=1}^{k} c_i^2 - \sum_{i=1}^{k} c_i^2 \\
&= \langle x,\, x \rangle + \sum_{i=1}^{k} (a_i - c_i)^2 - \sum_{i=1}^{k} c_i^2 \\
&= \left\| x - \sum_{i=1}^{k} c_i u_i \right\|^2 + \sum_{i=1}^{k} (a_i - c_i)^2 \\
&\geq \left\| x - \sum_{i=1}^{k} c_i u_i \right\|^2 .
\end{aligned} \tag{2.64}$$

Therefore we have $\|x - \tilde{x}\| \leq \|x - v\|$, and so $\tilde{x}$ formed by the Fourier coefficients is the best approximation of x with respect to the Euclidean norm in the k-dimensional vector space $\mathcal{V}$. (For some other norm, this may not be the case.)

2.2.6.2 Choice of the Best Basis Subset

Now, posing the problem another way, we may seek the best k-dimensional subspace of $\mathbb{R}^n$ from which to choose an approximating vector. This question

is not well-posed (because the one-dimensional vector space determined by x is the solution), but we can pose a related interesting question: suppose we have a Fourier expansion of x in terms of a set of n orthogonal basis vectors, $u_1, \ldots, u_n$, and we want to choose the "best" k basis vectors from this set and use them to form an approximation of x. (This restriction of the problem is equivalent to choosing a coordinate system.) We see the solution immediately from inequality (2.64): we choose the k u_is corresponding to the k largest c_is in absolute value, and we take

$$\tilde{x} = c_{i_1} u_{i_1} + \cdots + c_{i_k} u_{i_k}, \tag{2.65}$$

where $\min(\{|c_{i_j}| \ : \ j = 1, \ldots, k\}) \geq \max(\{|c_{i_j}| \ : \ j = k+1, \ldots, n\})$.

2.2.7 Flats, Affine Spaces, and Hyperplanes

Given an n-dimensional vector space of order n, $\mathbb{R}^n$ for example, consider a system of m linear equations in the n-vector variable x,

$$\begin{aligned} c_1^{\mathrm{T}} x &= b_1 \\ &\vdots \quad \vdots \\ c_m^{\mathrm{T}} x &= b_m, \end{aligned}$$

where $c_1, \ldots, c_m$ are linearly independent n-vectors (and hence $m \leq n$). The set of points defined by these linear equations is called a *flat*. Although it is not necessarily a vector space, a flat is also called an *affine space*. An intersection of two flats is a flat.

If the equations are *homogeneous* (that is, if $b_1 = \cdots = b_m = 0$), then the point $(0, \ldots, 0)$ is included, and the flat is an $(n-m)$-dimensional subspace (also a vector space, of course). Stating this another way, a flat through the origin is a vector space, but other flats are not vector spaces.

If $m = 1$, the flat is called a *hyperplane*. A hyperplane through the origin is an $(n-1)$-dimensional vector space.

If $m = n-1$, the flat is a line. A line through the origin is a one-dimensional vector space.

2.2.8 Cones

A set of vectors that contains all nonnegative scalar multiples of any vector in the set is called a *cone*. A cone always contains the zero vector. (Some authors define a cone as a set that contains all positive scalar multiples, and in that case, the zero vector may not be included.) If a set of vectors contains all scalar multiples of any vector in the set, it is called a *double cone*.

Geometrically, a cone is just a set, possibly a finite set, of lines or half-lines. (A double cone is a set of lines.) In general, a cone may not be very interesting, but certain special cones are of considerable interest.

Given two (double) cones over the same vector space, both their union and their intersection are (double) cones. A (double) cone is in general not a vector space.

2.2.8.1 Convex Cones

A set of vectors C in a vector space $\mathcal{V}$ is a *convex cone* if, for all $v_1, v_2 \in C$ and all nonnegative real numbers $a, b \geq 0$, $av_1 + bv_2 \in C$. Such a cone is called a homogeneous convex cone by some authors. (An equivalent definition requires that the set C be a cone, and then, more in keeping with the definition of convexity, includes the requirement $a + b = 1$ along with $a, b \geq 0$ in the definition of a convex cone.)

If C is a convex cone and if $v \in C$ implies $-v \in C$, then C is called a *double convex cone*.

A (double) convex cone is in general not a vector space because, for example, $v_1 + v_2$ may not be in C.

It is clear that a (double) convex cone is a (double) cone; in fact, a convex cone is the most important type of cone. A convex cone corresponds to a solid geometric object with a single finite vertex.

An important convex cone in an n-dimensional vector space with a Cartesian coordinate system is the positive orthant together with the zero vector. This convex cone is not closed, in the sense that it does not contain some limits. The closure of the positive orthant (that is, the nonnegative orthant) is also a convex cone.

A generating set or spanning set of a cone C is a set of vectors $S = \{v_i\}$ such that for any vector v in C there exists scalars $a_i \geq 0$ so that $v = \sum a_i v_i$. If, in addition, for any scalars $b_i \geq 0$ with $\sum b_i v_i = 0$, it is necessary that $b_i = 0$ for all i, then S is a basis set for the cone. The concept of a generating set is of course more interesting in the case of a convex cone.

If a generating set of a convex cone has a finite number of elements, the cone is a *polyhedron*. For the common geometric object in three dimensions with elliptical contours and which is the basis for "conic sections", any generating set has an uncountable number of elements. Cones of this type are sometimes called "Lorentz cones".

It is easy to see from the definition that if C_1 and C_2 are convex cones over the same vector space, then $C_1 \cap C_2$ is a convex cone. On the other hand, $C_1 \cup C_2$ is not necessarily a convex cone. Of course the union of two cones, as we have seen, is a cone.

2.2.8.2 Dual Cones

Given a set of vectors S in a given vector space (in cases of interest, S is usually a cone, but not necessarily), the *dual cone* of S, denoted $C^*(S)$, is defined as

$$C^*(S) = \{v^* \text{ s.t. } \langle v^*, v\rangle \geq 0 \text{ for all } v \in S\}.$$

See Fig. 2.2.8.2 in which $S = \{v_1, v_2, v_3\}$. Clearly, the dual cone is a cone, and also $S \subseteq C^*(S)$.

If, as in the most common cases, the underlying set of vectors is a cone, say C, we generally drop the reference to an underlying set of vectors, and just denote the dual cone of C as C^*.

Geometrically, the dual cone C^* of S consists of all vectors that form nonobtuse angles with the vectors in S.

Notice that for a given set of vectors S, if $-S$ represents the set of vectors v such that $-v \in S$, then $C^*(-S) = -(C^*(S))$, or just $-C^*(S)$, which represents the set of vectors v^* such that $-v^* \in C^*(S)$.

Further, from the definition, we note that if S_1 and S_2 are sets of vectors in the same vector space such that $S_1 \subseteq S_2$ then $C^*(S_1) \subseteq C^*(S_2)$, or $C_1^* \subseteq C_2^*$.

A dual cone $C^*(S)$ is a closed convex cone. We see this by considering any $v_1^*, v_2^* \in C^*$ and real numbers $a, b \geq 0$. For any $v \in S$, it must be the case that $\langle v_1^*, v\rangle \geq 0$ and $\langle v_2^*, v\rangle \geq 0$; hence, $\langle (av_1^* + bv_2^*), v\rangle \geq 0$, that is, $av_1^* + bv_2^* \in C^*$, so C^* is a convex cone. The closure property comes from the $\geq$ condition in the definition.

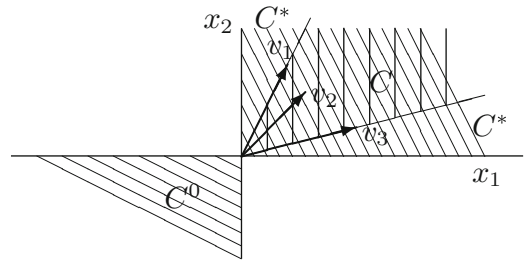


Figure 2.3. A set of vectors $\{v_1, v_2, v_3\}$, and the corresponding convex cone C, the dual cone C^*, and the polar cone C^0

2.2.8.3 Polar Cones

Given a set of vectors S in a given vector space (in cases of interest, S is usually a cone, but not necessarily), the *polar cone* of S, denoted $C^0(S)$, is defined as

$$C^0(S) = \{v^0 \text{ s.t. } \langle v^0, v\rangle \leq 0 \text{ for all } v \in S\}.$$

See Fig. 2.2.8.2.

We generally drop the reference to an underlying set of vectors, and just denote the dual cone of the set C as C^0.

From the definition, we note that if S_1 and S_2 are sets of vectors in the same vector space such that $S_1 \subseteq S_2$ then $C^0(S_1) \subseteq C^0(S_2)$, or $C_1^0 \subseteq C_2^0$.

The polar cone and the dual cone of a double cone are clearly the same.

From the definitions, it is clear in any case that the polar cone C^0 can be formed by multiplying all of the vectors in the corresponding dual cone C^* by -1, and so $C^0 = -C^*$.

The relationships of the polar cone to the dual cone and the properties we have established for a dual cone immediately imply that a polar cone is also a convex cone.

Another interesting property of polar cones is that for any set of vectors S in a given vector space, $S \subseteq (C^0)^0$. We generally write $(C^0)^0$ as just C^{00}. (The precise notation of course is $C^0(C^0(S))$.) We see this by first taking any $v \in S$. Therefore, if $v^0 \in C^0$ then $\langle v, v^0 \rangle \leq 0$, which implies $v \in (C^0)^0$, because

$$C^{00} = \{v \text{ s.t. } \langle v, v^0 \rangle \leq 0 \text{ for all } v^0 \in C^0\}.$$

2.2.8.4 Additional Properties

As noted above, a cone is a very loose and general structure. In my definition, the vectors in the set do not even need to be in the same vector space. A convex cone, on the other hand is a useful structure, and the vectors in a convex cone must be in the same vector space.

Most cones of interest, in particular, dual cones and polar cones are not necessarily vector spaces.

Although the definitions of dual cones and polar cones can apply to any set of vectors, they are of the most interest in the case in which the underlying set of vectors is a cone in the nonnegative orthant of a Cartesian coordinate system on $\mathbb{R}^n$ (the set of n-vectors all of whose elements are nonnegative). In that case, the dual cone is just the full nonnegative orthant, and the polar cone is just the nonpositive orthant (the set of all vectors all of whose elements are nonpositive).

The whole nonnegative orthant itself is a convex cone, and as we have seen for any convex cone within that orthant, the dual cone is the full nonnegative orthant.

Because the nonnegative orthant is its own dual, and hence is said to be "self-dual". (There is an extension of the property of self-duality that we will not discuss here.)

Convex cones occur in many optimization problems. The feasible region in a linear programming problem is generally a convex polyhedral cone, for example.

2.2.9 Cross Products in $\mathbb{R}^3$

The vector space $\mathbb{R}^3$ is especially interesting because it serves as a useful model of the real world, and many physical processes can be represented as vectors in it.

For the special case of the vector space $\mathbb{R}^3$, another useful vector product is the cross product, which is a mapping from $\mathbb{R}^3 \times \mathbb{R}^3$ to $\mathbb{R}^3$. Before proceeding,

we note an overloading of the term "cross product" and of the symbol "×" used to denote it. If A and B are sets, the *set cross product* or the *set Cartesian product* of A and B is the set consisting of all doubletons (a, b) where a ranges over all elements of A, and b ranges independently over all elements of B. Thus, $\mathbb{R}^3 \times \mathbb{R}^3$ is the set of all pairs of all real 3-vectors.

The *vector cross product* of the 3-vectors

$$x = (x_1, x_2, x_3)$$

and

$$y = (y_1, y_2, y_3),$$

written $x \times y$, is defined as

$$x \times y = (x_2y_3 - x_3y_2,\ x_3y_1 - x_1y_3,\ x_1y_2 - x_2y_1). \tag{2.66}$$

(We also use the term "cross products" in a different way to refer to another type of product formed by several inner products; see page 359.) The cross product has the following properties, which are immediately obvious from the definition:

1. Self-nilpotency:
   $x \times x = 0$, for all x.
2. Anti-commutativity:
   $x \times y = -y \times x$.
3. Factoring of scalar multiplication;
   $ax \times y = a(x \times y)$ for real a.
4. Relation of vector addition to addition of cross products:
   $(x + y) \times z = (x \times z) + (y \times z)$.

The cross product has the important property (sometimes taken as the definition),

$$x \times y = \|x\|\|y\| \sin(\text{angle}(y, x))e, \tag{2.67}$$

where e is a vector such that $\|e\| = 1$ and $\langle e, x\rangle = \langle e, y\rangle = 0$, and $\text{angle}(y, x)$ is interpreted as the "smallest angle through which y would be rotated to become a nonnegative multiple of x". (See Exercise 2.19e on page 54.)

In the definition of angles between vectors given on page 37, $\text{angle}(y, x) = \text{angle}(x, y)$. As we pointed out there, sometimes it is important to distinguish the direction of the angle, and this is the case in equation (2.67), as in many applications in $\mathbb{R}^3$. The direction of angles in $\mathbb{R}^3$ often is used to determine the orientation of the principal axes in a coordinate system. The coordinate system is often defined to be "right-handed" (see Exercise 2.19f).

The cross product is useful in modeling phenomena in nature, which naturally are often represented as vectors in $\mathbb{R}^3$. The cross product is also useful in "three-dimensional" computer graphics for determining whether a given surface is visible from a given perspective and for simulating the effect of lighting on a surface.

2.3 Centered Vectors and Variances and Covariances of Vectors

In this section, we define some scalar-valued functions of vectors that are analogous to functions of random variables averaged over their probabilities or probability density. The functions of vectors discussed here are the same as the ones that define sample statistics. This short section illustrates the properties of norms, inner products, and angles in terms that should be familiar to the reader.

These functions, and transformations using them, are useful for applications in the data sciences. It is important to know the effects of various transformations of data on data analysis.

2.3.1 The Mean and Centered Vectors

When the elements of a vector have some kind of common interpretation, the sum of the elements or the mean (equation (2.46)) of the vector may have meaning. In this case, it may make sense to *center* the vector; that is, to subtract the mean from each element. For a given vector x, we denote its centered counterpart as x_c:

$$x_c = x - \bar{x}. \tag{2.68}$$

We refer to any vector whose sum of elements is 0 as a centered vector; note, therefore, for any centered vector x_c,

$$1^T x_c = 0;$$

or, indeed, for any constant vector a, $a^T x_c = 0$.

From the definitions, it is easy to see that

$$(x + y)_c = x_c + y_c \tag{2.69}$$

(see Exercise 2.20). Interpreting $\bar{x}$ as a vector, and recalling that it is the projection of x onto the one vector, we see that x_c is the residual in the sense of equation (2.52). Hence, we see that x_c and $\bar{x}$ are orthogonal, and the Pythagorean relationship holds:

$$\|x\|^2 = \|\bar{x}\|^2 + \|x_c\|^2. \tag{2.70}$$

From this we see that the length of a centered vector is less than or equal to the length of the original vector. (Notice that equation (2.70) is just the formula familiar to data analysts, which with some rearrangement is $\sum(x_i - \bar{x})^2 = \sum x_i^2 - n\bar{x}^2$.)

For any scalar a and n-vector x, expanding the terms, we see that

$$\|x - a\|^2 = \|x_c\|^2 + n(a - \bar{x})^2, \tag{2.71}$$

where we interpret $\bar{x}$ as a scalar here. An implication of this equation is that for all values of a, $\|x - a\|$ is minimized if $a = \bar{x}$.

Notice that a nonzero vector when centered may be the zero vector. This leads us to suspect that some properties that depend on a dot product are not invariant to centering. This is indeed the case. The angle between two vectors, for example, is not invariant to centering; that is, in general,

$$\text{angle}(x_c, y_c) \neq \text{angle}(x, y) \tag{2.72}$$

(see Exercise 2.21).

2.3.2 The Standard Deviation, the Variance, and Scaled Vectors

We also sometimes find it useful to scale a vector by both its length (normalize the vector) and by a function of its number of elements. We denote this *scaled* vector as x_s and define it as

$$x_s = \sqrt{n-1}\,\frac{x}{\|x_c\|}. \tag{2.73}$$

For comparing vectors, it is usually better to center the vectors prior to any scaling. We denote this *centered and scaled* vector as x_{cs} and define it as

$$x_{cs} = \sqrt{n-1}\,\frac{x_c}{\|x_c\|}. \tag{2.74}$$

Centering and scaling is also called *standardizing*. Note that the vector is centered before being scaled. The angle between two vectors is not changed by scaling (but, of course, it may be changed by centering).

The multiplicative inverse of the scaling factor,

$$s_x = \|x_c\|/\sqrt{n-1}, \tag{2.75}$$

is called the *standard deviation* of the vector x. The standard deviation of x_c is the same as that of x; in fact, the standard deviation is invariant to the addition of any constant. The standard deviation is a measure of how much the elements of the vector vary. If all of the elements of the vector are the same, the standard deviation is 0 because in that case $x_c = 0$.

The square of the standard deviation is called the *variance*, denoted by V:

$$\begin{aligned} \mathrm{V}(x) &= s_x^2 \\ &= \frac{\|x_c\|^2}{n-1}. \end{aligned} \tag{2.76}$$

(In perhaps more familiar notation, equation (2.76) is just $\mathrm{V}(x) = \sum(x_i - \bar{x})^2/(n-1)$.) From equation (2.70), we see that

$$\mathrm{V}(x) = \frac{1}{n-1}\left(\|x\|^2 - \|\bar{x}\|^2\right).$$

(The terms "mean", "standard deviation", "variance", and other terms we will mention below are also used in an analogous, but slightly different, manner to refer to properties of *random variables*. In that context, the terms to refer to the quantities we are discussing here would be preceded by the word "sample", and often for clarity I will use the phrases "sample standard deviation" and "sample variance" to refer to what is defined above, especially if the elements of x are interpreted as independent realizations of a random variable. Also, recall the two possible meanings of "mean", or $\bar{x}$; one is a vector, and one is a scalar, as in equation (2.47).)

If a and b are scalars (or b is a vector with all elements the same), the definition, together with equation (2.71), immediately gives

$$\mathrm{V}(ax + b) = a^2\mathrm{V}(x).$$

This implies that for the scaled vector x_{s},

$$\mathrm{V}(x_{\mathrm{s}}) = 1.$$

If a is a scalar and x and y are vectors with the same number of elements, from the equation above, and using equation (2.31) on page 26, we see that the variance following an axpy operation is given by

$$\mathrm{V}(ax + y) = a^2\mathrm{V}(x) + \mathrm{V}(y) + 2a\frac{\langle x_{\mathrm{c}},\ y_{\mathrm{c}}\rangle}{n-1}. \tag{2.77}$$

While equation (2.76) appears to be relatively simple, evaluating the expression for a given x may not be straightforward. We discuss computational issues for this expression on page 502. This is an instance of a principle that we will encounter repeatedly: *the form of a mathematical expression and the way the expression should be evaluated in actual practice may be quite different.*

2.3.3 Covariances and Correlations Between Vectors

If x and y are n-vectors, the *covariance* between x and y is

$$\mathrm{Cov}(x, y) = \frac{\langle x - \bar{x},\ y - \bar{y}\rangle}{n-1}. \tag{2.78}$$

By representing $x - \bar{x}$ as $x - \bar{x}1$ and $y - \bar{y}$ similarly, and expanding, we see that $\mathrm{Cov}(x, y) = (\langle x,\ y\rangle - n\bar{x}\bar{y})/(n-1)$. Also, we see from the definition of covariance that $\mathrm{Cov}(x, x)$ is the variance of the vector x, as defined above.

From the definition and the properties of an inner product given on page 24, if x, y, and z are conformable vectors, we see immediately that

- $\mathrm{Cov}(x, y) = 0$
  if $\mathrm{V}(x) = 0$ or $\mathrm{V}(y) = 0$;
- $\mathrm{Cov}(ax, y) = a\mathrm{Cov}(x, y)$
  for any scalar a;

- $\mathrm{Cov}(y, x) = \mathrm{Cov}(x, y)$;
- $\mathrm{Cov}(y, y) = \mathrm{V}(y)$; and
- $\mathrm{Cov}(x + z, y) = \mathrm{Cov}(x, y) + \mathrm{Cov}(z, y)$,
  in particular,
  - $\mathrm{Cov}(x + y, y) = \mathrm{Cov}(x, y) + \mathrm{V}(y)$, *and*
  - $\mathrm{Cov}(x + a, y) = \mathrm{Cov}(x, y)$
    for any scalar a.

Using the definition of the covariance, we can rewrite equation (2.77) as

$$\mathrm{V}(ax + y) = a^2\mathrm{V}(x) + \mathrm{V}(y) + 2a\mathrm{Cov}(x, y). \tag{2.79}$$

The covariance is a measure of the extent to which the vectors point in the same direction. A more meaningful measure of this is obtained by the covariance of the centered and scaled vectors. This is the *correlation* between the vectors, which if $\|x_c\| \neq 0$ and $\|y_c\| \neq 0$,

$$\begin{aligned} \mathrm{Cor}(x, y) &= \mathrm{Cov}(x_{cs}, y_{cs}) \\ &= \left\langle \frac{x_c}{\|x_c\|}, \frac{y_c}{\|y_c\|} \right\rangle \\ &= \frac{\langle x_c, y_c \rangle}{\|x_c\|\|y_c\|}. \end{aligned} \tag{2.80}$$

If $\|x_c\| = 0$ or $\|y_c\| = 0$, we define $\mathrm{Cor}(x, y)$ to be 0. We see immediately from equation (2.54) that the correlation is the cosine of the angle between x_c and y_c:

$$\mathrm{Cor}(x, y) = \cos(\mathrm{angle}(x_c, y_c)). \tag{2.81}$$

(Recall that this is not the same as the angle between x and y.)

An equivalent expression for the correlation, so long as $\mathrm{V}(x) \neq 0$ and $\mathrm{V}(y) \neq 0$, is

$$\mathrm{Cor}(x, y) = \frac{\mathrm{Cov}(x, y)}{\sqrt{\mathrm{V}(x)\mathrm{V}(y)}}. \tag{2.82}$$

It is clear that the correlation is in the interval $[-1, 1]$ (from the Cauchy-Schwarz inequality). A correlation of -1 indicates that the vectors point in opposite directions, a correlation of 1 indicates that the vectors point in the same direction, and a correlation of 0 indicates that the vectors are orthogonal.

While the covariance is equivariant to scalar multiplication, the absolute value of the correlation is invariant to it; that is, the correlation changes only as the sign of the scalar multiplier,

$$\mathrm{Cor}(ax, y) = \mathrm{sign}(a)\mathrm{Cor}(x, y), \tag{2.83}$$

for any scalar a.

Exercises

2.1. Write out the step-by-step proof that the maximum number of n-vectors that can form a set that is linearly independent is n.

2.2. Prove inequalities (2.10) and (2.11).

2.3. a) Give an example of a vector space and a subset of the set of vectors in it such that that subset together with the axpy operation is *not* a vector space.

b) Give an example of two vector spaces such that the union of the sets of vectors in them together with the axpy operation is *not* a vector space.

2.4. Prove the equalities (2.15) and (2.16).
Hint: Use of basis sets makes the details easier.

2.5. Prove (2.19).

2.6. Let $\{v_i\}_{i=1}^n$ be an orthonormal basis for the n-dimensional vector space $\mathcal{V}$. Let $x \in \mathcal{V}$ have the representation

$$x = \sum b_i v_i.$$

Show that the Fourier coefficients b_i can be computed as

$$b_i = \langle x, v_i \rangle.$$

2.7. Show that if the norm is induced by an inner product that the parallelogram equality, equation (2.32), holds.

2.8. Let $p = \frac{1}{2}$ in equation (2.33); that is, let $\rho(x)$ be defined for the n-vector x as

$$\rho(x) = \left(\sum_{i=1}^{n} |x_i|^{1/2} \right)^2.$$

Show that $\rho(\cdot)$ is not a norm.

2.9. Show that the L_1 norm is not induced by an inner product.
Hint: Find a counterexample that does not satisfy the parallelogram equality (equation (2.32)).

2.10. Prove equation (2.34) and show that the bounds are sharp by exhibiting instances of equality. (Use the fact that $\|x\|_\infty = \max_i |x_i|$.)

2.11. Prove the following inequalities.

a) Prove Hölder's inequality: for any p and q such that $p \geq 1$ and $p + q = pq$, and for vectors x and y of the same order,

$$\langle x, y \rangle \leq \|x\|_p \|y\|_q.$$

b) Prove the triangle inequality for any L_p norm. (This is sometimes called Minkowski's inequality.)
Hint: Use Hölder's inequality.

2.12. Show that the expression defined in equation (2.42) on page 32 is a metric.

2.13. Show that equation (2.53) on page 37 is correct.

2.14. Show that the intersection of two orthogonal vector spaces consists only of the zero vector.

2.15. From the definition of direction cosines in equation (2.55), it is easy to see that the sum of the squares of the direction cosines is 1. For the special case of $\mathbb{R}^3$, draw a sketch and use properties of right triangles to show this geometrically.

2.16. In $\mathbb{R}^2$ with a Cartesian coordinate system, the diagonal directed line segment through the positive quadrant (orthant) makes a 45° angle with each of the positive axes. In 3 dimensions, what is the angle between the diagonal and each of the positive axes? In 10 dimensions? In 100 dimensions? In 1000 dimensions? We see that in higher dimensions any two lines are almost orthogonal. (That is, the angle between them approaches 90°.) What are some of the implications of this for data analysis?

2.17. Show that if the initial set of vectors are linearly independent, all residuals in Algorithm 2.1 are nonzero. (For given $k \geq 2$, all that is required is to show that

$$\tilde{x}_k - \langle \tilde{x}_{k-1}, \tilde{x}_k \rangle \tilde{x}_{k-1} \neq 0$$

if $\tilde{x}_k$ and $\tilde{x}_{k-1}$ are linearly independent. Why?)

2.18. Convex cones.

a) I defined a convex cone as a set of vectors (not necessarily a cone) such that for any two vectors v_1, v_2 in the set and for any nonnegative real numbers $a, b \geq 0$, $av_1 + bv_2$ is in the set. Then I stated that an equivalent definition requires first that the set be a cone, and then includes the requirement $a + b = 1$ along with $a, b \geq 0$ in the definition of a convex cone. Show that the two definitions are equivalent.

b) The restriction that $a + b = 1$ in the definition of a convex cone is the kind of restriction that we usually encounter in definitions of convex objects. Without this restriction, it may seem that the linear combinations may get "outside" of the object. Show that this is not the case for convex cones.
In particular in the two-dimensional case, show that if $x = (x_1, x_2)$, $y = (y_1, y_2)$, with $x_1/x_2 < y_1/y_2$ and $a, b \geq 0$, then

$$x_1/x_2 \leq (ax_1 + by_1)/(ax_2 + by_2) \leq y_1/y_2.$$

This should also help to give a geometrical perspective on convex cones.

c) Show that if C_1 and C_2 are convex cones over the same vector space, then $C_1 \cap C_2$ is a convex cone. Give a counterexample to show that $C_1 \cup C_2$ is not necessarily a convex cone.

2.19. $\mathbb{R}^3$ and the cross product.

a) Is the cross product associative? Prove or disprove.

b) For $x, y \in \mathbb{R}^3$, show that the area of the triangle with vertices $(0, 0, 0)$, x, and y is $\|x \times y\|/2$.

c) For $x, y, z \in \mathbb{R}^3$, show that

$$\langle x,\ y \times z\rangle = \langle x \times y,\ z\rangle.$$

This is called the "triple scalar product".

d) For $x, y, z \in \mathbb{R}^3$, show that

$$x \times (y \times z) = \langle x,\ z\rangle y - \langle x,\ y\rangle z.$$

This is called the "triple vector product". It is in the plane determined by y and z.

e) The magnitude of the angle between two vectors is determined by the cosine, formed from the inner product. Show that in the special case of $\mathbb{R}^3$, the angle is also determined by the sine and the cross product, and show that this method can determine both the magnitude and the *direction* of the angle; that is, the way a particular vector is rotated into the other.

f) In a Cartesian coordinate system in $\mathbb{R}^3$, the principal axes correspond to the unit vectors $e_1 = (1, 0, 0)$, $e_2 = (0, 1, 0)$, and $e_3 = (0, 0, 1)$. This system has an indeterminate correspondence to a physical three-dimensional system; if the plane determined by e_1 and e_2 is taken as horizontal, then e_3 could "point upward" or "point downward". A simple way that this indeterminacy can be resolved is to require that the principal axes have the orientation of the thumb, index finger, and middle finger of the right hand when those digits are spread in orthogonal directions, where e_1 corresponds to the index finger, e_2 corresponds to the middle finger, and e_3 corresponds to the thumb. This is called a "right-hand" coordinate system.
Show that in a right-hand coordinate system, if we interpret the angle between e_i and e_j to be measured in the direction from e_i to e_j, then $e_3 = e_1 \times e_2$ and $e_3 = -e_2 \times e_1$.

2.20. Using equations (2.46) and (2.68), establish equation (2.69).

2.21. Show that the angle between the centered vectors x_c and y_c is not the same in general as the angle between the uncentered vectors x and y of the same order.

2.22. Formally prove equation (2.77) (and hence equation (2.79)).

2.23. Let x and y be any vectors of the same order over the same field.

a) Prove

$$(\mathrm{Cov}(x, y))^2 \leq \mathrm{V}(x)\mathrm{V}(y).$$

b) Hence, prove

$$-1 \leq \mathrm{Cor}(x, y)) \leq 1.$$

3

Basic Properties of Matrices

In this chapter, we build on the notions introduced on page 5, and discuss a wide range of basic topics related to matrices with real elements. Some of the properties carry over to matrices with complex elements, but the reader should not assume this. Occasionally, for emphasis, we will refer to "real" matrices, but unless it is stated otherwise, we are assuming the matrices are real.

The topics and the properties of matrices that we choose to discuss are motivated by applications in the data sciences. In Chap. 8, we will consider in more detail some special types of matrices that arise in regression analysis and multivariate data analysis, and then in Chap. 9 we will discuss some specific applications in statistics.

3.1 Basic Definitions and Notation

It is often useful to treat the rows or columns of a matrix as vectors. Terms such as linear independence that we have defined for vectors also apply to rows and/or columns of a matrix. The vector space generated by the columns of the $n \times m$ matrix A is of order n and of dimension m or less, and is called the *column space* of A, the *range* of A, or the *manifold* of A. This vector space is denoted by

$$\mathcal{V}(A)$$

or

$$\text{span}(A).$$

I make no distinction between these two notations. The notation $\mathcal{V}(\cdot)$ emphasizes that the result is a vector space. Note that if $A \in \mathbb{R}^{n \times m}$, then $\mathcal{V}(A) \subseteq \mathbb{R}^n$.

J.E. Gentle, *Matrix Algebra*, Springer Texts in Statistics,
DOI 10.1007/978-3-319-64867-5_3

The argument of $\mathcal{V}(\cdot)$ or span$(\cdot)$ can also be a set of vectors instead of a matrix. Recall from Sect. 2.1.3 that if G is a set of vectors, the symbol span(G) denotes the vector space generated by the vectors in G.

We also define the *row space* of A to be the vector space of order m (and of dimension n or less) generated by the rows of A; notice, however, the preference given to the column space.

Many of the properties of matrices that we discuss hold for matrices with an infinite number of elements, but throughout this book we will assume that the matrices have a finite number of elements, and hence the vector spaces are of finite order and have a finite number of dimensions.

Given an $n \times m$ matrix A with elements a_{ij}, the $m \times n$ matrix with elements a_{ji} is called the *transpose* of A. We use a superscript "T" to denote the transpose of a matrix; thus, if $A = (a_{ij})$, then

$$A^{\mathrm{T}} = (a_{ji}). \tag{3.1}$$

(In other literature, the transpose is often denoted by a prime, as in $A' = (a_{ji}) = A^{\mathrm{T}}$.)

If, in the matrix A with elements a_{ij} for all i and j, $a_{ij} = a_{ji}$, A is said to be *symmetric*. A symmetric matrix is necessarily square. A matrix A such that $a_{ij} = -a_{ji}$ is said to be *skew symmetric*. Obviously, the diagonal entries of a skew symmetric matrix must be 0. If $a_{ij} = \bar{a}_{ji}$ (where $\bar{a}$ represents the conjugate of the complex number a), A is said to be *Hermitian*. A Hermitian matrix is also necessarily square with real elements on the diagonal, and, of course, a real symmetric matrix is Hermitian. A Hermitian matrix is also called a *self-adjoint* matrix.

3.1.1 Multiplication of a Matrix by a Scalar

Similar to our definition of multiplication of a vector by a scalar, we define the multiplication of a matrix A by a scalar c as

$$cA = (ca_{ij}).$$

3.1.2 Diagonal Elements: diag(·) and vecdiag(·)

The a_{ii} elements of a matrix are called *diagonal elements*. An element a_{ij} with $i < j$ is said to be "above the diagonal", and one with $i > j$ is said to be "below the diagonal". The vector consisting of all of the a_{ii}'s is called the *principal diagonal* or just the diagonal. This definition of principal diagonal applies whether or not the matrix is square.

We denote the principal diagonal of a matrix A by diag(A) or by vecdiag(A). The latter notation is sometimes used because, as we will see on page 60, diag$(\dot{)}$ is also used for an argument that is a vector (and the function produces a matrix). The diag or vecdiag function defined here is a mapping $\mathbb{R}^{n \times m} \to \mathbb{R}^{\min(n,m)}$.

If A is an $n \times m$ matrix, and $k = \min(n, m)$,

$$\text{diag}(A) = (a_{11}, \ldots, a_{kk}). \tag{3.2}$$

As noted above, we may also denote this as vecdiag(A), but I will generally use the notation "diag($\cdot$)".

Note from the definition that

$$\text{diag}(A^{\mathrm{T}}) = \text{diag}(A), \tag{3.3}$$

and this is true whether or not A is square.

The diagonal begins in the first row and first column (that is, a_{11}), and ends at a_{kk}, where k is the minimum of the number of rows and the number of columns.

For $c = \pm 1, \ldots$, the elements $a_{i,i+c}$ are called "codiagonals" or "minor diagonals". The codiagonals $a_{i,i+1}$ are called "supradiagonals", and the codiagonals $a_{i,i-1}$ are called "infradiagonals" If the matrix has m columns, the $a_{i,m+1-i}$ elements of the matrix are called *skew diagonal elements*. We use terms similar to those for diagonal elements for elements above and below the skew diagonal elements. These phrases are used with both square and nonsquare matrices.

3.1.3 Diagonal, Hollow, and Diagonally Dominant Matrices

If all except the principal diagonal elements of a matrix are 0, the matrix is called a *diagonal matrix*. A diagonal matrix is the most common and most important type of "sparse matrix". If all of the principal diagonal elements of a matrix are 0, the matrix is called a *hollow matrix*. A skew symmetric matrix is hollow, for example. If all except the principal skew diagonal elements of a matrix are 0, the matrix is called a *skew diagonal matrix*.

An $n \times m$ matrix A for which

$$|a_{ii}| > \sum_{j \neq i}^{m} |a_{ij}| \quad \text{for each } i = 1, \ldots, n \tag{3.4}$$

is said to be row *diagonally dominant*; and a matrix A for which $|a_{jj}| > \sum_{i \neq j}^{n} |a_{ij}|$ for each $j = 1, \ldots, m$ is said to be column diagonally dominant. (Some authors refer to this as *strict* diagonal dominance and use "diagonal dominance" without qualification to allow the possibility that the inequalities in the definitions are not strict.) Most interesting properties of such matrices hold whether the dominance is by row or by column. If A is symmetric, row and column diagonal dominances are equivalent, so we refer to row or column diagonally dominant symmetric matrices without the qualification; that is, as just diagonally dominant.

3.1.4 Matrices with Special Patterns of Zeroes

If all elements below the diagonal are 0, the matrix is called an *upper triangular matrix*; and a *lower triangular matrix* is defined similarly. If all elements of a column or row of a triangular matrix are zero, we still refer to the matrix as triangular, although sometimes we speak of its form as *trapezoidal*. Another form called trapezoidal is one in which there are more columns than rows, and the additional columns are possibly nonzero. The four general forms of triangular or trapezoidal matrices are shown below, using an intuitive notation with X and 0 to indicate the pattern.

$$\begin{bmatrix} \mathtt{X} & \mathtt{X} & \mathtt{X} \\ \mathtt{0} & \mathtt{X} & \mathtt{X} \\ \mathtt{0} & \mathtt{0} & \mathtt{X} \end{bmatrix} \quad \begin{bmatrix} \mathtt{X} & \mathtt{X} & \mathtt{X} \\ \mathtt{0} & \mathtt{X} & \mathtt{X} \\ \mathtt{0} & \mathtt{0} & \mathtt{0} \end{bmatrix} \quad \begin{bmatrix} \mathtt{X} & \mathtt{X} & \mathtt{X} \\ \mathtt{0} & \mathtt{X} & \mathtt{X} \\ \mathtt{0} & \mathtt{0} & \mathtt{X} \\ \mathtt{0} & \mathtt{0} & \mathtt{0} \end{bmatrix} \quad \begin{bmatrix} \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} \\ \mathtt{0} & \mathtt{X} & \mathtt{X} & \mathtt{X} \\ \mathtt{0} & \mathtt{0} & \mathtt{X} & \mathtt{X} \end{bmatrix}$$

In this notation, X indicates that the element is possibly not zero. It does not mean each element is the same. In some cases, X and 0 may indicate "submatrices", which we discuss in the section on partitioned matrices.

If all elements are 0 except $a_{i,i+c_k}$ for some small number of integers c_k, the matrix is called a *band matrix* (or *banded matrix*). In many applications, $c_k \in \{-w_l, -w_l + 1, \ldots, -1, 0, 1, \ldots, w_u - 1, w_u\}$. In such a case, w_l is called the *lower band width* and w_u is called the *upper band width*. These patterned matrices arise in time series and other stochastic process models as well as in solutions of differential equations, and so they are very important in certain applications. Although it is often the case that interesting band matrices are symmetric, or at least have the same number of codiagonals that are nonzero, neither of these conditions always occurs in applications of band matrices. If all elements below the principal skew diagonal elements of a matrix are 0, the matrix is called a *skew upper triangular matrix*. A common form of Hankel matrix, for example, is the skew upper triangular matrix (see page 390). Notice that the various terms defined here, such as triangular and band, also apply to nonsquare matrices.

Band matrices occur often in numerical solutions of partial differential equations. A band matrix with lower and upper band widths of 1 is a *tridiagonal matrix*. If all diagonal elements and all elements $a_{i,i\pm1}$ are nonzero, a tridiagonal matrix is called a "matrix of type 2". The inverse of a covariance matrix that occurs in common stationary time series models is a matrix of type 2 (see page 385).

Using the intuitive notation of X and 0 as above, a band matrix may be written as

$$\begin{bmatrix} \mathtt{X} & \mathtt{X} & \mathtt{0} & \cdots & \mathtt{0} & \mathtt{0} \\ \mathtt{X} & \mathtt{X} & \mathtt{X} & \cdots & \mathtt{0} & \mathtt{0} \\ \mathtt{0} & \mathtt{X} & \mathtt{X} & \cdots & \mathtt{0} & \mathtt{0} \\ & & \ddots & & \ddots & \\ \mathtt{0} & \mathtt{0} & \mathtt{0} & \cdots & \mathtt{X} & \mathtt{X} \end{bmatrix}.$$

Computational methods for matrices may be more efficient if the patterns are taken into account.

A matrix is in upper *Hessenberg form*, and is called a *Hessenberg matrix*, if it is upper triangular except for the first subdiagonal, which may be nonzero. That is, $a_{ij} = 0$ for $i > j + 1$:

$$\begin{bmatrix} \mathtt{X} & \mathtt{X} & \mathtt{X} & \cdots & \mathtt{X} & \mathtt{X} \\ \mathtt{X} & \mathtt{X} & \mathtt{X} & \cdots & \mathtt{X} & \mathtt{X} \\ \mathtt{0} & \mathtt{X} & \mathtt{X} & \cdots & \mathtt{X} & \mathtt{X} \\ \mathtt{0} & \mathtt{0} & \mathtt{X} & \cdots & \mathtt{X} & \mathtt{X} \\ \vdots & \vdots & & \ddots & \vdots & \vdots \\ \mathtt{0} & \mathtt{0} & \mathtt{0} & \cdots & \mathtt{X} & \mathtt{X} \end{bmatrix}.$$

A symmetric matrix that is in Hessenberg form is necessarily *tridiagonal*.

Hessenberg matrices arise in some methods for computing eigenvalues (see Chap. 7).

Many matrices of interest are *sparse*; that is, they have a large proportion of elements that are 0. The matrices discussed above are generally not considered sparse. ("A large proportion" is subjective, but generally means more than 75%, and in many interesting cases is well over 95%.) Efficient and accurate computations often require that the sparsity of a matrix be accommodated explicitly.

3.1.5 Matrix Shaping Operators

In order to perform certain operations on matrices and vectors, it is often useful first to reshape a matrix. The most common reshaping operation is the transpose, which we define in this section. Sometimes we may need to rearrange the elements of a matrix or form a vector into a special matrix. In this section, we define three operators for doing this.

3.1.5.1 Transpose

As defined above, the *transpose* of a matrix is the matrix whose i^{th} row is the i^{th} column of the original matrix and whose j^{th} column is the j^{th} row of the original matrix. We note immediately that

$$(A^{\mathrm{T}})^{\mathrm{T}} = A. \tag{3.5}$$

If the elements of the matrix are from the field of complex numbers, the *conjugate transpose*, also called the *adjoint*, is more useful than the transpose. ("Adjoint" is also used to denote another type of matrix, so we will generally avoid using that term. This meaning of the word is the origin of the other term for a Hermitian matrix, a "self-adjoint matrix".) We use a superscript "H" to denote the conjugate transpose of a matrix; thus, if $A = (a_{ij})$, then

$$A^{\mathrm{H}} = (\bar{a}_{ji}). \tag{3.6}$$

We also use a similar notation for vectors. (The conjugate transpose is often denoted by an asterisk, as in $A^* = (\bar{a}_{ji}) = A^{\mathrm{H}}$. This notation is more common if a prime is used to denote the transpose. We sometimes use the notation A^* to denote a g_2 inverse of the matrix A; see page 128.) As with the transponse, $(A^{\mathrm{H}})^{\mathrm{H}} = A$. If (and only if) all of the elements of A are all real, then $A^{\mathrm{H}} = A^{\mathrm{T}}$.

If (and only if) A is symmetric, $A = A^{\mathrm{T}}$; if (and only if) A is skew symmetric, $A^{\mathrm{T}} = -A$; and if (and only if) A is Hermitian, $A = A^{\mathrm{H}}$ (and, in that case, all of the diagonal elements are real).

3.1.5.2 Diagonal Matrices and Diagonal Vectors: diag(·) (Again)

A square diagonal matrix can be specified by a constructor function that operates on a vector and forms a diagonal matrix with the elements of the vector along the diagonal. We denote that constructor function by diag(·), just as we used this name to denote a somewhat similar function on page 57.

$$\mathrm{diag}\big((d_1, d_2, \ldots, d_n)\big) = \begin{bmatrix} d_1 & 0 & \cdots & 0 \\ 0 & d_2 & \cdots & 0 \\ & & \ddots & \\ 0 & 0 & \cdots & d_n \end{bmatrix}. \tag{3.7}$$

(Notice that the argument of diag here is a vector; that is why there are two sets of parentheses in the expression above, although sometimes we omit one set without loss of clarity.) The diag function defined here is a mapping $\mathbb{R}^n \to \mathbb{R}^{n\times n}$. Later we will extend this definition slightly.

A very important diagonal matrix has all 1s along the diagonal. If it has n diagonal elements, it is denoted by I_n; so $I_n = \mathrm{diag}(1_n)$. This is called the *identity matrix of order n*. The size is often omitted, and we call it the identity matrix, and denote it by I.

Note that we have overloaded diag(·), which we defined on page 57 with a matrix argument, to allow its argument to be a vector. (Recall that vecdiag(·) is the same as diag(·) when the argument is a matrix.) Both the R and Matlab computing systems, for example, use this overloading; that is, they each provide a single function (called `diag` in each case).

Note further that over $\mathbb{R}^n$ and $\mathbb{R}^{n\times n}$, diag(·) is its own inverse; that is, if v is a vector,

$$\mathrm{diag}(\mathrm{diag}(v)) = v, \tag{3.8}$$

and if A is a square matrix,

$$\mathrm{diag}(\mathrm{diag}(A)) = A. \tag{3.9}$$

3.1.5.3 Forming a Vector from the Elements of a Matrix: vec(·) and vech(·)

It is sometimes useful to consider the elements of a matrix to be elements of a single vector. The most common way this is done is to string the columns of the matrix end-to-end into a vector. The vec(·) function does this:

$$\text{vec}(A) = (a_1^{\mathrm{T}}, a_2^{\mathrm{T}}, \ldots, a_m^{\mathrm{T}}), \tag{3.10}$$

where $a_1, a_2, \ldots, a_m$ are the column vectors of the matrix A. The vec function is also sometimes called the "pack" function. (A note on the notation: the right side of equation (3.10) is the notation for a column vector with elements a_i^{T}; see Chap. 1.) The vec function is a mapping $\mathbb{R}^{n\times m} \to \mathbb{R}^{nm}$.

For a symmetric matrix A with elements a_{ij}, the "vech" function stacks the unique elements into a vector:

$$\text{vech}(A) = (a_{11}, a_{21}, \ldots, a_{m1}, a_{22}, \ldots, a_{m2}, \ldots, a_{mm}). \tag{3.11}$$

There are other ways that the unique elements could be stacked that would be simpler and perhaps more useful (see the discussion of symmetric storage mode on page 548), but equation (3.11) is the standard definition of vech(·). The vech function is a mapping $\mathbb{R}^{n\times n} \to \mathbb{R}^{n(n+1)/2}$.

3.1.6 Partitioned Matrices

We often find it useful to partition a matrix into submatrices; for example, in many applications in data analysis, it is often convenient to work with submatrices of various types representing different subsets of the data.

We usually denote the submatrices with capital letters with subscripts indicating the relative positions of the submatrices. Hence, we may write

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \tag{3.12}$$

where the matrices A_{11} and A_{12} have the same number of rows, A_{21} and A_{22} have the same number of rows, A_{11} and A_{21} have the same number of columns, and A_{12} and A_{22} have the same number of columns. Of course, the submatrices in a partitioned matrix may be denoted by different letters. Also, for clarity, sometimes we use a vertical bar to indicate a partition:

$$A = [\, B \,|\, C \,].$$

The vertical bar is used just for clarity and has no special meaning in this representation.

The term "submatrix" is also used to refer to a matrix formed from a given matrix by deleting various rows and columns of the given matrix. In this terminology, B is a submatrix of A if for each element b_{ij} there is an a_{kl} with

$k \geq i$ and $l \geq j$ such that $b_{ij} = a_{kl}$; that is, the rows and/or columns of the submatrix are not necessarily contiguous in the original matrix. A more precise notation specifies the rows and columns of the original matrix. For example, $A_{(i_1,\ldots,i_k)(j_1,\ldots,j_l)}$ denotes the submatrix of A formed by rows $i_1, \ldots, i_k$ and columns $j_1, \ldots, j_l$. When the entire rows are included, $A_{(i_1,\ldots,i_k)(*)}$ denotes the submatrix of A formed from rows $i_1, \ldots, i_k$; and $A_{(*)(j_1,\ldots,j_l)}$ denotes the submatrix formed from columns $j_1, \ldots, j_l$ with elements from all rows. Finally, a_{i*} denotes the vector whose elements correspond to those in the i^{th} row of the matrix A. We sometimes emphasize that it is a *vector* by writing it in the form a_{i*}^{T}. Likewise, a_{*j} denotes the vector whose elements correspond to those in the j^{th} column of A. See page 599 for a summary of this notation. This kind of subsetting is often done in data analysis, for example, in variable selection in linear regression analysis.

A square submatrix whose principal diagonal elements are elements of the principal diagonal of the given matrix is called a *principal submatrix*. If A_{11} in the example above is square, it is a principal submatrix, and if A_{22} is square, it is also a principal submatrix. Sometimes the term "principal submatrix" is restricted to square submatrices. If a matrix is diagonally dominant, then it is clear that any principal submatrix of it is also diagonally dominant.

A principal submatrix that contains the $(1, 1)$ element and whose rows and columns are contiguous in the original matrix is called a *leading principal submatrix*. If A_{11} is square, it is a leading principal submatrix in the example above.

Partitioned matrices may have useful patterns. A "block diagonal" matrix is one of the form

$$\begin{bmatrix} \mathtt{X} & \mathtt{0} & \cdots & \mathtt{0} \\ \mathtt{0} & \mathtt{X} & \cdots & \mathtt{0} \\ & & \ddots & \\ \mathtt{0} & \mathtt{0} & \cdots & \mathtt{X} \end{bmatrix},$$

where $\mathtt{0}$ represents a submatrix with all zeros and $\mathtt{X}$ represents a general submatrix with at least some nonzeros.

The diag$(\cdot)$ function previously introduced for a vector is also defined for a list of matrices:

$$\mathrm{diag}(A_1, A_2, \ldots, A_k)$$

denotes the block diagonal matrix with submatrices $A_1, A_2, \ldots, A_k$ along the diagonal and zeros elsewhere. A matrix formed in this way is sometimes called a *direct sum* of $A_1, A_2, \ldots, A_k$, and the operation is denoted by $\oplus$:

$$A_1 \oplus \cdots \oplus A_k = \mathrm{diag}(A_1, \ldots, A_k). \tag{3.13}$$

Although the direct sum is a binary operation, we are justified in defining it for a list of matrices because the operation is clearly associative.

The A_i may be of different sizes and they may not be square, although in most applications the matrices are square (and some authors define the direct sum only for square matrices).

We will define vector spaces of matrices below and then recall the definition of a direct sum of vector spaces (page 18), which is different from the direct sum defined above in terms of diag(·).

3.1.6.1 Transposes of Partitioned Matrices

The transpose of a partitioned matrix is formed in the obvious way; for example,

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} A_{11}^{\mathrm{T}} & A_{21}^{\mathrm{T}} \\ A_{12}^{\mathrm{T}} & A_{22}^{\mathrm{T}} \\ A_{13}^{\mathrm{T}} & A_{23}^{\mathrm{T}} \end{bmatrix}. \tag{3.14}$$

3.1.7 Matrix Addition

The sum of two matrices of the same shape is the matrix whose elements are the sums of the corresponding elements of the addends. As in the case of vector addition, we overload the usual symbols for the operations on the reals to signify the corresponding operations on matrices when the operations are defined; hence, addition of matrices is also indicated by "+", as with scalar addition and vector addition. We assume throughout that writing a sum of matrices $A + B$ implies that they are of the same shape; that is, that they are *conformable for addition*.

The "+" operator can also mean addition of a scalar to a matrix, as in $A + a$, where A is a matrix and a is a scalar. Although this meaning of "+" is generally not used in mathematical treatments of matrices, in this book we use it to mean the addition of the scalar to each element of the matrix, resulting in a matrix of the same shape. This meaning is consistent with the semantics of modern computer languages such as Fortran and R.

The addition of two $n \times m$ matrices or the addition of a scalar to an $n \times m$ matrix requires nm scalar additions.

The *matrix additive identity* is a matrix with all elements zero. We sometimes denote such a matrix with n rows and m columns as $0_{n\times m}$, or just as 0. We may denote a square additive identity as 0_n.

3.1.7.1 The Transpose of the Sum of Matrices

The transpose of the sum of two matrices is the sum of the transposes:

$$(A + B)^{\mathrm{T}} = A^{\mathrm{T}} + B^{\mathrm{T}}. \tag{3.15}$$

The sum of two symmetric matrices is therefore symmetric.

3.1.7.2 Rank Ordering Matrices

There are several possible ways to form a rank ordering of matrices of the same shape, but no complete ordering is entirely satisfactory. If all of the elements of the matrix A are positive, we write

$$A > 0; \tag{3.16}$$

if all of the elements are nonnegative, we write

$$A \geq 0. \tag{3.17}$$

The terms "positive" and "nonnegative" and these symbols are not to be confused with the terms "positive definite" and "nonnegative definite" and similar symbols for important classes of matrices having different properties (which we will introduce on page 92, and discuss further in Sect. 8.3.)

3.1.7.3 Vector Spaces of Matrices

Having defined scalar multiplication and matrix addition (for conformable matrices), we can define a vector space of $n \times m$ matrices as any set that is closed with respect to those operations. The individual operations of scalar multiplication and matrix addition allow us to define an axpy operation on the matrices, as in equation (2.1) on page 12. Closure of this space implies that it must contain the additive identity, just as we saw on page 13). The matrix additive identity is the 0 matrix.

As with any vector space, we have the concepts of linear independence, generating set or spanning set, basis set, essentially disjoint spaces, and direct sums of matrix vector spaces (as in equation (2.13), which is different from the direct sum of matrices defined in terms of diag(·) as in equation (3.13)).

An important vector space of matrices is $\mathbb{R}^{n\times m}$. For matrices $X, Y \in \mathbb{R}^{n\times m}$ and $a \in \mathbb{R}$, the axpy operation is $aX + Y$.

If $n \geq m$, a set of nm $n \times m$ matrices whose columns consist of all combinations of a set of n n-vectors that span $\mathbb{R}^n$ is a basis set for $\mathbb{R}^{n\times m}$. If $n < m$, we can likewise form a basis set for $\mathbb{R}^{n\times m}$ or for subspaces of $\mathbb{R}^{n\times m}$ in a similar way. If $\{B_1, \ldots, B_k\}$ is a basis set for $\mathbb{R}^{n\times m}$, then any $n \times m$ matrix can be represented as $\sum_{i=1}^{k} c_i B_i$. Subsets of a basis set generate subspaces of $\mathbb{R}^{n\times m}$.

Because the sum of two symmetric matrices is symmetric, and a scalar multiple of a symmetric matrix is likewise symmetric, we have a vector space of the $n \times n$ symmetric matrices. This is clearly a subspace of the vector space $\mathbb{R}^{n\times n}$. All vectors in any basis for this vector space must be symmetric. Using a process similar to our development of a basis for a general vector space of matrices, we see that there are $n(n+1)/2$ matrices in the basis (see Exercise 3.1).

3.1.8 Scalar-Valued Operators on Square Matrices: The Trace

There are several useful mappings from matrices to real numbers; that is, from $\mathbb{R}^{n\times m}$ to $\mathbb{R}$. Some important ones are norms, which are similar to vector norms and which we will consider later. In this section and the next, we define two scalar-valued operators, the trace and the determinant, that apply to square matrices.

3.1.8.1 The Trace: tr(·)

The sum of the diagonal elements of a square matrix is called the *trace* of the matrix. We use the notation "$\mathrm{tr}(A)$" to denote the trace of the matrix A:

$$\mathrm{tr}(A) = \sum_i a_{ii}. \tag{3.18}$$

3.1.8.2 The Trace of the Transpose of Square Matrices

From the definition, we see

$$\mathrm{tr}(A) = \mathrm{tr}(A^{\mathrm{T}}). \tag{3.19}$$

3.1.8.3 The Trace of Scalar Products of Square Matrices

For a scalar c and an $n \times n$ matrix A,

$$\mathrm{tr}(cA) = c\,\mathrm{tr}(A).$$

This follows immediately from the definition because for $\mathrm{tr}(cA)$ each diagonal element is multiplied by c.

3.1.8.4 The Trace of Partitioned Square Matrices

If the square matrix A is partitioned such that the diagonal blocks are square submatrices, that is,

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \tag{3.20}$$

where A_{11} and A_{22} are square, then from the definition, we see that

$$\mathrm{tr}(A) = \mathrm{tr}(A_{11}) + \mathrm{tr}(A_{22}). \tag{3.21}$$

3.1.8.5 The Trace of the Sum of Square Matrices

If A and B are square matrices of the same order, a useful (and obvious) property of the trace is

$$\mathrm{tr}(A + B) = \mathrm{tr}(A) + \mathrm{tr}(B). \tag{3.22}$$

3.1.9 Scalar-Valued Operators on Square Matrices: The Determinant

The determinant, like the trace, is a mapping from $\mathbb{R}^{n\times n}$ to $\mathbb{R}$. Although it may not be obvious from the definition below, the determinant has far-reaching applications in matrix theory.

3.1.9.1 The Determinant: det(·)

For an $n \times n$ (square) matrix A, consider the product $a_{1j_1} \cdots a_{nj_n}$, where $\pi_j = (j_1, \ldots, j_n)$ is one of the $n!$ permutations of the integers from 1 to n. Define a permutation to be *even* or *odd* according to the number of times that a smaller element follows a larger one in the permutation. For example, given the tuple $(1, 2, 3)$, then $(1, 3, 2)$ is an odd permutation, and $(3, 1, 2)$ and $(1, 2, 3)$ are even permutations. Let

$$\sigma(\pi) = \begin{cases} 1 \text{ if } \pi \text{ is an even permutation} \\ -1 \text{ otherwise.} \end{cases} \tag{3.23}$$

Then the *determinant* of A, denoted by $\det(A)$, is defined by

$$\det(A) = \sum_{\text{all permutations } \pi_j} \sigma(\pi_j) a_{1j_1} \cdots a_{nj_n}. \tag{3.24}$$

This simple function has many remarkable relationships to various properties of matrices.

3.1.9.2 Notation and Simple Properties of the Determinant

The determinant is also sometimes written as $|A|$.

I prefer the notation $\det(A)$, because of the possible confusion between $|A|$ and the absolute value of some quantity. The latter notation, however, is recommended by its compactness, and I do use it in expressions such as the PDF of the multivariate normal distribution (see equation (4.73)) that involve nonnegative definite matrices (see page 91 for the definition). The determinant of a matrix may be negative, and sometimes, as in measuring volumes (see page 74 for simple areas and page 215 for special volumes called Jacobians), we need to specify the absolute value of the determinant, so we need something of the form $|\det(A)|$.

The definition of the determinant is not as daunting as it may appear at first glance. Many properties become obvious when we realize that $\sigma(\cdot)$ is always ± 1, and it can be built up by elementary exchanges of adjacent elements. For example, consider $\sigma(3, 2, 1)$. There are two ways we can use three elementary exchanges, each beginning with the natural ordering:

$$(1, 2, 3) \to (2, 1, 3) \to (2, 3, 1) \to (3, 2, 1),$$

or

$$(1,2,3) \to (1,3,2) \to (3,1,2) \to (3,2,1);$$

hence, either way, $\sigma(3,2,1) = (-1)^3 = -1$.

If π_j consists of the interchange of exactly two elements in $(1,\ldots,n)$, say elements p and q with $p < q$, then there are $q-p$ elements before p that are larger than p, and there are $q-p-1$ elements between q and p in the permutation each with exactly one larger element preceding it. The total number is $2q-2p+1$, which is an odd number. Therefore, if π_j consists of the interchange of exactly two elements, then $\sigma(\pi_j) = -1$.

If the integers $1,\ldots,m$ occur sequentially in a given permutation and are followed by $m+1,\ldots,n$ which also occur sequentially in the permutation, they can be considered separately:

$$\sigma(j_1,\ldots,j_n) = \sigma(j_1,\ldots,j_m)\sigma(j_{m+1},\ldots,j_n). \tag{3.25}$$

Furthermore, we see that the product $a_{1j_1}\cdots a_{nj_n}$ has exactly one factor from each unique row-column pair. These observations facilitate the derivation of various properties of the determinant (although the details are sometimes quite tedious).

We see immediately from the definition that the determinant of an upper or lower triangular matrix (or a diagonal matrix) is merely the product of the diagonal elements (because in each term of equation (3.24) there is a 0, except in the term in which the subscripts on each factor are the same).

3.1.9.3 Minors, Cofactors, and Adjugate Matrices

Consider the 2×2 matrix

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}.$$

From the definition of the determinant, we see that

$$\det(A) = a_{11}a_{22} - a_{12}a_{21}. \tag{3.26}$$

Now let A be a 3×3 matrix:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}.$$

In the definition of the determinant, consider all of the terms in which the elements of the first row of A appear. With some manipulation of those terms, we can express the determinant in terms of determinants of submatrices as

$$\begin{aligned}\det(A) = \; & a_{11}(-1)^{1+1}\det\left(\begin{bmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{bmatrix}\right) \\ & + \; a_{12}(-1)^{1+2}\det\left(\begin{bmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{bmatrix}\right) \\ & + a_{13}(-1)^{1+3}\det\left(\begin{bmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix}\right).\end{aligned} \tag{3.27}$$

Notice that this is the same form as in equation (3.26):

$$\det(A) = a_{11}(1)\det(a_{22}) + a_{12}(-1)\det(a_{21}).$$

The manipulation in equation (3.27) of the terms in the determinant could be carried out with other rows of A.

The determinants of the 2×2 submatrices in equation (3.27) are called *minors* or *complementary minors* of the associated element. The definition can be extended to $(n-1) \times (n-1)$ submatrices of an $n \times n$ matrix, for $n \geq 2$. We denote the minor associated with the a_{ij} element as

$$\det\left(A_{-(i)(j)}\right), \tag{3.28}$$

in which $A_{-(i)(j)}$ denotes the submatrix that is formed from A by removing the i^{th} row and the j^{th} column. The sign associated with the minor corresponding to a_{ij} is $(-1)^{i+j}$. The minor together with its appropriate sign is called the *cofactor* of the associated element; that is, the cofactor of a_{ij} is $(-1)^{i+j}\det\left(A_{-(i)(j)}\right)$. We denote the cofactor of a_{ij} as $a_{(ij)}$:

$$a_{(ij)} = (-1)^{i+j}\det\left(A_{-(i)(j)}\right). \tag{3.29}$$

Notice that both minors and cofactors are scalars.

The manipulations leading to equation (3.27), though somewhat tedious, can be carried out for a square matrix of any size larger than 1×1, and minors and cofactors are defined as above. An expression such as in equation (3.27) is called an expansion in minors or an expansion in cofactors.

The extension of the expansion (3.27) to an expression involving a sum of signed products of complementary minors arising from $(n-1) \times (n-1)$ submatrices of an $n \times n$ matrix A is

$$\begin{aligned}\det(A) &= \sum_{j=1}^{n} a_{ij}(-1)^{i+j}\det\left(A_{-(i)(j)}\right) \\ &= \sum_{j=1}^{n} a_{ij}a_{(ij)},\end{aligned} \tag{3.30}$$

or, over the rows,

$$\det(A) = \sum_{i=1}^{n} a_{ij} a_{(ij)}. \tag{3.31}$$

These expressions are called *Laplace expansions*. Each determinant $\det\left(A_{-(i)(j)}\right)$ can likewise be expressed recursively in a similar expansion.

Expressions (3.30) and (3.31) are special cases of a more general Laplace expansion based on an extension of the concept of a complementary minor of an element to that of a complementary minor of a minor. The derivation of the general Laplace expansion is straightforward but rather tedious (see Harville 1997, for example, for the details).

Laplace expansions could be used to compute the determinant, but the main value of these expansions is in proving properties of determinants. For example, from the special Laplace expansion (3.30) or (3.31), we can quickly see that the determinant of a matrix with two rows that are the same is zero. We see this by recursively expanding all of the minors until we have only 2×2 matrices consisting of a duplicated row. The determinant of such a matrix is 0, so the expansion is 0.

The expansion in equation (3.30) has an interesting property: if instead of the elements a_{ij} from the i^{th} row we use elements from a different row, say the k^{th} row, the sum is zero. That is, for $k \neq i$,

$$\begin{aligned} \sum_{j=1}^{n} a_{kj}(-1)^{i+j} \det\left(A_{-(i)(j)}\right) &= \sum_{j=1}^{n} a_{kj} a_{(ij)} \\ &= 0. \end{aligned} \tag{3.32}$$

This is true because such an expansion is exactly the same as an expansion for the determinant of a matrix whose k^{th} row has been replaced by its i^{th} row; that is, a matrix with two identical rows. The determinant of such a matrix is 0, as we saw above.

A certain matrix formed from the cofactors has some interesting properties. We define the matrix here but defer further discussion. The *adjugate* of the $n \times n$ matrix A is defined as

$$\operatorname{adj}(A) = (a_{(ji)}), \tag{3.33}$$

which is an $n \times n$ matrix of the cofactors of the elements of the transposed matrix. (The adjugate is also called the *adjoint* or sometimes "classical adjoint", but as we noted above, the term adjoint may also mean the conjugate transpose. To distinguish it from the conjugate transpose, the adjugate is also sometimes called the "classical adjoint". We will generally avoid using the term "adjoint".) Note the reversal of the subscripts; that is,

$$\text{adj}(A) = (a_{(ij)})^{\mathrm{T}}.$$

The adjugate has an interesting property involving matrix multiplication (which we will define below in Sect. 3.2) and the identity matrix:

$$A\,\text{adj}(A) = \text{adj}(A)A = \det(A)I. \tag{3.34}$$

To see this, consider the $(i,j)^{\text{th}}$ element of $A\,\text{adj}(A)$. By the definition of the multiplication of A and $\text{adj}(A)$, that element is $\sum_k a_{ik}(\text{adj}(A))_{kj}$. Now, noting the reversal of the subscripts in $\text{adj}(A)$ in equation (3.33), and using equations (3.30) and (3.32), we have

$$\sum_k a_{ik}(\text{adj}(A))_{kj} = \begin{cases} \det(A) & \text{if } i = j \\ 0 & \text{if } i \neq j; \end{cases}$$

that is, $A\,\text{adj}(A) = \det(A)I$.

The adjugate has a number of other useful properties, some of which we will encounter later, as in equation (3.172).

3.1.9.4 The Determinant of the Transpose of Square Matrices

One important property we see immediately from a manipulation of the definition of the determinant is

$$\det(A) = \det(A^{\mathrm{T}}). \tag{3.35}$$

3.1.9.5 The Determinant of Scalar Products of Square Matrices

For a scalar c and an $n \times n$ matrix A,

$$\det(cA) = c^n \det(A). \tag{3.36}$$

This follows immediately from the definition because, for $\det(cA)$, each factor in each term of equation (3.24) is multiplied by c.

3.1.9.6 The Determinant of an Upper (or Lower) Triangular Matrix

If A is an $n \times n$ upper (or lower) triangular matrix, then

$$\det(A) = \prod_{i=1}^{n} a_{ii}. \tag{3.37}$$

This follows immediately from the definition. It can be generalized, as in the next section.

3.1.9.7 The Determinant of Certain Partitioned Square Matrices

Determinants of square partitioned matrices that are block diagonal or upper or lower block triangular depend only on the diagonal partitions:

$$\det(A) = \det\left(\begin{bmatrix} A_{11} & 0 \\ 0 & A_{22} \end{bmatrix}\right) = \det\left(\begin{bmatrix} A_{11} & 0 \\ A_{21} & A_{22} \end{bmatrix}\right) = \det\left(\begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{bmatrix}\right)$$
$$= \det(A_{11})\det(A_{22}). \tag{3.38}$$

We can see this by considering the individual terms in the determinant, equation (3.24). Suppose the full matrix is $n \times n$, and A_{11} is $m \times m$. Then A_{22} is $(n-m) \times (n-m)$, A_{21} is $(n-m) \times m$, and A_{12} is $m \times (n-m)$. In equation (3.24), any addend for which $(j_1, \ldots, j_m)$ is not a permutation of the integers $1, \ldots, m$ contains a factor a_{ij} that is in a 0 diagonal block, and hence the addend is 0. The determinant consists only of those addends for which $(j_1, \ldots, j_m)$ is a permutation of the integers $1, \ldots, m$, and hence $(j_{m+1}, \ldots, j_n)$ is a permutation of the integers $m+1, \ldots, n$,

$$\det(A) = \sum\sum \sigma(j_1, \ldots, j_m, j_{m+1}, \ldots, j_n) a_{1j_1} \cdots a_{mj_m} a_{m+1,j_n} \cdots a_{nj_n},$$

where the first sum is taken over all permutations that keep the first m integers together while maintaining a fixed ordering for the integers $m+1$ through n, and the second sum is taken over all permutations of the integers from $m+1$ through n while maintaining a fixed ordering of the integers from 1 to m. Now, using equation (3.25), we therefore have for A of this special form

$$\begin{aligned} \det(A) &= \sum\sum \sigma(j_1, \ldots, j_m, j_{m+1}, \ldots, j_n) a_{1j_1} \cdots a_{mj_m} a_{m+1,j_{m+1}} \cdots a_{nj_n} \\ &= \sum \sigma(j_1, \ldots, j_m) a_{1j_1} \cdots a_{mj_m} \sum \sigma(j_{m+1}, \ldots, j_n) a_{m+1,j_{m+1}} \cdots a_{nj_n} \\ &= \det(A_{11})\det(A_{22}), \end{aligned}$$

which is equation (3.38). We use this result to give an expression for the determinant of more general partitioned matrices in Sect. 3.4.2.

Another useful partitioned matrix of the form of equation (3.20) has $A_{11} = 0$ and $A_{21} = -I$:

$$A = \begin{bmatrix} 0 & A_{12} \\ -I & A_{22} \end{bmatrix}.$$

In this case, using equation (3.30), we get

$$\begin{aligned} \det(A) &= ((-1)^{n+1+1}(-1))^n \det(A_{12}) \\ &= (-1)^{n(n+3)} \det(A_{12}) \\ &= \det(A_{12}). \end{aligned} \tag{3.39}$$

We will consider determinants of a more general partitioning in Sect. 3.4.2, beginning on page 122.

3.1.9.8 The Determinant of the Sum of Square Matrices

Occasionally it is of interest to consider the determinant of the sum of square matrices. We note in general that

$$\det(A+B) \neq \det(A) + \det(B),$$

which we can see easily by an example. (Consider matrices in $\mathbb{R}^{2\times 2}$, for example, and let $A = I$ and $B = \begin{bmatrix} -1 & 0 \\ 0 & 0 \end{bmatrix}$.)

In some cases, however, simplified expressions for the determinant of a sum can be developed. We consider one in the next section.

3.1.9.9 A Diagonal Expansion of the Determinant

A particular sum of matrices whose determinant is of interest is one in which a diagonal matrix D is added to a square matrix A, that is, $\det(A+D)$. (Such a determinant arises in eigenanalysis, for example, as we see in Sect. 3.8.4.)

For evaluating the determinant $\det(A+D)$, we can develop another expansion of the determinant by restricting our choice of minors to determinants of matrices formed by deleting the same rows and columns and then continuing to delete rows and columns recursively from the resulting matrices. The expansion is a polynomial in the elements of D; and for our purposes later, that is the most useful form.

Before considering the details, let us develop some additional notation. The matrix formed by deleting the same row and column of A is denoted $A_{-(i)(i)}$ as above (following equation (3.28)). In the current context, however, it is more convenient to adopt the notation $A_{(i_1,\ldots,i_k)}$ to represent the matrix formed from rows $i_1,\ldots,i_k$ and columns $i_1,\ldots,i_k$ from a given matrix A. That is, the notation $A_{(i_1,\ldots,i_k)}$ indicates the rows and columns *kept* rather than those deleted; and furthermore, in this notation, the indexes of the rows and columns are the same. We denote the determinant of this $k \times k$ matrix in the obvious way, $\det(A_{(i_1,\ldots,i_k)})$. Because the principal diagonal elements of this matrix are principal diagonal elements of A, we call $\det(A_{(i_1,\ldots,i_k)})$ a *principal minor* of A.

Now consider $\det(A+D)$ for the 2×2 case:

$$\det\left(\begin{bmatrix} a_{11}+d_1 & a_{12} \\ a_{21} & a_{22}+d_2 \end{bmatrix}\right).$$

Expanding this, we have

$$\det(A+D) = (a_{11}+d_1)(a_{22}+d_2) - a_{12}a_{21}$$

$$= \det\left(\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}\right) + d_1 d_2 + a_{22} d_1 + a_{11} d_2$$

$$= \det(A_{(1,2)}) + d_1 d_2 + a_{22} d_1 + a_{11} d_2.$$

Of course, $\det(A_{(1,2)}) = \det(A)$, but we are writing it this way to develop the pattern. Now, for the 3×3 case, we have

$$\begin{aligned} \det(A + D) = & \det(A_{(1,2,3)}) \\ & + \det(A_{(2,3)}) d_1 + \det(A_{(1,3)}) d_2 + \det(A_{(1,2)}) d_3 \\ & + a_{33} d_1 d_2 + a_{22} d_1 d_3 + a_{11} d_2 d_3 \\ & + d_1 d_2 d_3. \end{aligned} \tag{3.40}$$

In the applications of interest, the elements of the diagonal matrix D may be a single variable: d, say. In this case, the expression simplifies to

$$\det(A + D) = \det(A_{(1,2,3)}) + \sum_{i \neq j} \det(A_{(i,j)}) d + \sum_i a_{i,i} d^2 + d^3. \tag{3.41}$$

Carefully continuing in this way for an $n \times n$ matrix, either as in equation (3.40) for n variables or as in equation (3.41) for a single variable, we can make use of a Laplace expansion to evaluate the determinant.

Consider the expansion in a single variable because that will prove most useful. The pattern persists; the constant term is $|A|$, the coefficient of the first-degree term is the sum of the $(n-1)$-order principal minors, and, at the other end, the coefficient of the $(n-1)^{\text{th}}$-degree term is the sum of the first-order principal minors (that is, just the diagonal elements), and finally the coefficient of the n^{th}-degree term is 1.

This kind of representation is called a *diagonal expansion* of the determinant because the coefficients are principal minors. It has occasional use for matrices with large patterns of zeros, but its main application is in analysis of eigenvalues, which we consider in Sect. 3.8.4.

3.1.9.10 Computing the Determinant

For an arbitrary matrix, the determinant is rather difficult to compute. The method for computing a determinant is not the one that would arise directly from the definition or even from a Laplace expansion. The more efficient methods involve first factoring the matrix, as we discuss in later sections.

The determinant is not very often directly useful, but although it may not be obvious from its definition, the determinant, along with minors, cofactors, and adjoint matrices, is very useful in discovering and proving properties of matrices. The determinant is used extensively in eigenanalysis (see Sect. 3.8).

3.1.9.11 A Geometrical Perspective of the Determinant

In Sect. 2.2, we discussed a useful geometric interpretation of vectors in a linear space with a Cartesian coordinate system. The elements of a vector correspond to measurements along the respective axes of the coordinate system. When working with several vectors, or with a matrix in which the columns (or rows) are associated with vectors, we may designate a vector x_i as $x_i = (x_{i1}, \ldots, x_{id})$. A set of d linearly independent d-vectors define a parallelotope in d dimensions. For example, in a two-dimensional space, the linearly independent 2-vectors x_1 and x_2 define a parallelogram, as shown in Fig. 3.1.

The area of this parallelogram is the base times the height, bh, where, in this case, b is the length of the vector x_1, and h is the length of x_2 times the sine of the angle θ. Thus, making use of equation (2.54) on page 37 for the cosine of the angle, we have

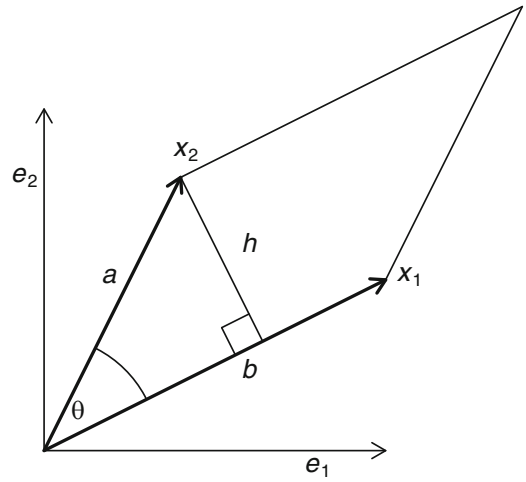


Figure 3.1. Volume (area) of region determined by x_1 and x_2

$$
\begin{aligned}
\text{area} &= bh \\
&= \|x_1\|\|x_2\| \sin(\theta) \\
&= \|x_1\|\|x_2\| \sqrt{1 - \left(\frac{\langle x_1, x_2 \rangle}{\|x_1\|\|x_2\|} \right)^2} \\
&= \sqrt{\|x_1\|^2 \|x_2\|^2 - (\langle x_1, x_2 \rangle)^2} \\
&= \sqrt{(x_{11}^2 + x_{12}^2)(x_{21}^2 + x_{22}^2) - (x_{11}x_{21} - x_{12}x_{22})^2}
\end{aligned}
$$

$$\begin{aligned} &= |x_{11}x_{22} - x_{12}x_{21}| \\ &= |\det(X)|, \end{aligned} \tag{3.42}$$

where $x_1 = (x_{11}, x_{12})$, $x_2 = (x_{21}, x_{22})$, and

$$\begin{aligned} X &= [x_1 \mid x_2] \\ &= \begin{bmatrix} x_{11} & x_{21} \\ x_{12} & x_{22} \end{bmatrix}. \end{aligned}$$

Although we will not go through the details here, this equivalence of a volume of a parallelotope that has a vertex at the origin and the absolute value of the determinant of a square matrix whose columns correspond to the vectors that form the sides of the parallelotope extends to higher dimensions.

In making a change of variables in integrals, as in equation (4.62) on page 215, we use the absolute value of the determinant of the Jacobian as a volume element. Another instance of the interpretation of the determinant as a volume is in the generalized variance, discussed on page 368.

3.2 Multiplication of Matrices and Multiplication of Vectors and Matrices

The elements of a vector or matrix are elements of a field, and most matrix and vector operations are defined in terms of the two operations of the field. Of course, in this book, the field of most interest is the field of real numbers.

3.2.1 Matrix Multiplication (Cayley)

There are various kinds of multiplication of matrices that may be useful. The most common kind of multiplication is *Cayley multiplication*. If the number of columns of the matrix A, with elements a_{ij}, and the number of rows of the matrix B, with elements b_{ij}, are equal, then the *(Cayley) product* of A and B is defined as the matrix C with elements

$$c_{ij} = \sum_k a_{ik} b_{kj}. \tag{3.43}$$

This is the most common type of matrix product, and we refer to it by the unqualified phrase "matrix multiplication".

Cayley matrix multiplication is indicated by juxtaposition, with no intervening symbol for the operation: $C = AB$.

If the matrix A is $n \times m$ and the matrix B is $m \times p$, the product $C = AB$ is $n \times p$:

$$\begin{array}{ccc} C & = & A \quad B \\ \left[\quad\right]_{n\times p} & = & \left[\quad\right]_{n\times m} \left[\quad\right]_{m\times p} \end{array}.$$

Cayley matrix multiplication is a mapping,

$$\mathbb{R}^{n\times m} \times \mathbb{R}^{m\times p} \to \mathbb{R}^{n\times p}.$$

The multiplication of an $n \times m$ matrix and an $m \times p$ matrix requires nmp scalar multiplications and $np(m-1)$ scalar additions. Here, as always in numerical analysis, we must remember that the definition of an operation, such as matrix multiplication, does not necessarily define a good algorithm for evaluating the operation.

It is obvious that while the product AB may be well-defined, the product BA is defined only if $n = p$; that is, if the matrices AB and BA are square. We assume throughout that writing a product of matrices AB implies that the number of columns of the first matrix is the same as the number of rows of the second; that is, they are *conformable for multiplication* in the order given.

It is easy to see from the definition of matrix multiplication (3.43) that in general, even for square matrices, $AB \neq BA$. It is also obvious that if AB exists, then $B^{\mathrm{T}}A^{\mathrm{T}}$ exists and, in fact,

$$B^{\mathrm{T}}A^{\mathrm{T}} = (AB)^{\mathrm{T}}. \tag{3.44}$$

The product of symmetric matrices is not, in general, symmetric. If (but not only if) A and B are symmetric, then $AB = (BA)^{\mathrm{T}}$.

Because matrix multiplication is not commutative, we often use the terms "premultiply" and "postmultiply" and the corresponding nominal forms of these terms. Thus, in the product AB, we may say B is premultiplied by A, or, equivalently, A is postmultiplied by B.

Although matrix multiplication is *not commutative*, it is *associative*; that is, if the matrices are conformable,

$$A(BC) = (AB)C. \tag{3.45}$$

It is also *distributive* over addition; that is,

$$A(B + C) = AB + AC \tag{3.46}$$

and

$$(B + C)A = BA + CA. \tag{3.47}$$

These properties are obvious from the definition of matrix multiplication. (Note that left-sided distribution is not the same as right-sided distribution because the multiplication is not commutative.)

An $n\times n$ matrix consisting of 1s along the diagonal and 0s everywhere else is a *multiplicative identity* for the set of $n\times n$ matrices and Cayley multiplication. Such a matrix is called the *identity matrix of order* n, and is denoted by I_n, or just by I. The columns of the identity matrix are unit vectors.

The identity matrix is a multiplicative identity for any matrix so long as the matrices are conformable for the multiplication. If A is $n \times m$, then

$$I_n A = A I_m = A.$$

Another matrix of interest is a *zero* matrix, which is any matrix consisting of all zeros. We denote a zero matrix as 0, with its shape being implied by the context. Two properties for any matrix A and a zero matrix of the appropriate shape are immediately obvious:

$$0A = 0$$

and

$$0 + A = A.$$

3.2.1.1 Powers of Square Matrices

For a square matrix A, its product with itself is defined, and so we will use the notation A^2 to mean the Cayley product AA, with similar meanings for A^k for a positive integer k. As with the analogous scalar case, A^k for a negative integer may or may not exist, and when it exists, it has a meaning for Cayley multiplication similar to the meaning in ordinary scalar multiplication. We will consider these issues later (in Sect. 3.3.6).

For an $n \times n$ matrix A, if A^k exists for negative integral values of k, we define A^0 by

$$A^0 = I_n. \tag{3.48}$$

For a diagonal matrix $D = \text{diag}\left((d_1, \ldots, d_n)\right)$, we have

$$D^k = \text{diag}\left((d_1^k, \ldots, d_n^k)\right). \tag{3.49}$$

3.2.1.2 Nilpotent Matrices

For an $n \times n$ matrix A, it may be the case for some positive integer k that $A^k = 0$. Such a matrix is said to be nilpotent; more generally, we define a *nilpotent matrix of index k*, for integer $k > 1$, as a square matrix A such that

$$A^k = 0, \qquad \text{but} \qquad A^{k-1} \neq 0. \tag{3.50}$$

We may use the term "nilpotent" without qualification to refer to a matrix that is nilpotent of any index; that is, strictly speaking, a nilpotent matrix is nilpotent of index 2.

A simple example of a matrix that is nilpotent of index 3 is

$$A = \left[\begin{array}{ccc|cc} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{array}\right], \tag{3.51}$$

in which I have indicated four submatrices of interest.

All nilpotent matrices have a certain relationship to matrices of the form of A in equation (3.51). We will identify that form here, but we will not discuss that form further. Notice two submatrices of A:

$$N_1 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \quad \text{and} \quad N_2 = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \tag{3.52}$$

so

$$A = \begin{bmatrix} N_1 & 0 \\ 0 & N_2 \end{bmatrix}.$$

Matrices of the form of N_1 and N_2, consisting of all 0s except for 1s in the supradiagonal, are called *Jordan blocks* and the nilpotent matrix A is said be in *Jordan form*. An important property, which we will merely state without proof, is that the index of a nilpotent matrix in Jordan form is the number of 1s in the largest Jordan block.

A nilpotent matrix is necessarily singular. Nilpotent matrices have many other simple properties, some of which we will list on page 174.

3.2.1.3 Matrix Polynomials

Polynomials in square matrices are similar to the more familiar polynomials in scalars. We may consider

$$p(A) = b_0 I + b_1 A + \cdots b_k A^k.$$

The value of this polynomial is a matrix.

The theory of polynomials in general holds, and in particular, we have the useful factorizations of monomials: for any positive integer k,

$$I - A^k = (I - A)(I + A + \cdots A^{k-1}), \tag{3.53}$$

and for an odd positive integer k,

$$I + A^k = (I + A)(I - A + \cdots A^{k-1}). \tag{3.54}$$

3.2.2 Multiplication of Matrices with Special Patterns

Various properties of matrices may or may not be preserved under matrix multiplication. We have seen already that the product of symmetric matrices is not in general symmetric.

Many of the various patterns of zeroes in matrices discussed on page 58 are preserved under matrix multiplication. Assume A and B are square matrices of the same number of rows.

- If A and B are diagonal, AB is diagonal and the (i,i) element of AB is $a_{ii}b_{ii}$;
- if A and B are block diagonal with conformable blocks, AB is block diagonal;
- if A and B are upper triangular, AB is upper triangular and the (i,i) element of AB is $a_{ii}b_{ii}$;
- if A and B are lower triangular, AB is lower triangular and the (i,i) element of AB is $a_{ii}b_{ii}$;
- if A is upper triangular and B is lower triangular, in general, none of AB, BA, $A^{\mathrm{T}}A$, $B^{\mathrm{T}}B$, AA^{T}, and BB^{T} is triangular.

Each of these statements can be easily proven using the definition of multiplication in equation (3.43). An important special case of diagonal and triangular matrices is one in which all diagonal elements are 1. Such a diagonal matrix is the identity, of course, so it a very special multiplicative property. Triangular matrices with 1s on the diagonal are called "unit triangular" matrices, and they are often used in matrix factorizations, as we see later in this chapter and in Chap. 5.

The products of banded matrices are generally banded with a wider bandwidth. If the bandwidth is too great, obviously the matrix can no longer be called banded.

3.2.2.1 Multiplication of Partitioned Matrices

Multiplication and other operations with partitioned matrices are carried out with their submatrices in the obvious way. Thus, assuming the submatrices are conformable for multiplication,

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{bmatrix}.$$

It is clear that the product of conformable block diagonal matrices is block diagonal.

Sometimes a matrix may be partitioned such that one partition is just a single column or row, that is, a vector or the transpose of a vector. In that case, we may use a notation such as

$$[X\ y]$$

or

$$[X \mid y],$$

where X is a matrix and y is a vector. We develop the notation in the obvious fashion; for example,

$$[X\ y]^{\mathrm{T}}[X\ y] = \begin{bmatrix} X^{\mathrm{T}}X & X^{\mathrm{T}}y \\ y^{\mathrm{T}}X & y^{\mathrm{T}}y \end{bmatrix}. \tag{3.55}$$

3.2.3 Elementary Operations on Matrices

Many common computations involving matrices can be performed as a sequence of three simple types of operations on either the rows or the columns of the matrix:

- the interchange of two rows (columns),
- a scalar multiplication of a given row (column), and
- the replacement of a given row (column) by the sum of that row (columns) and a scalar multiple of another row (column); that is, an axpy operation.

Such an operation on the rows of a matrix can be performed by premultiplication by a matrix in a standard form, and an operation on the columns of a matrix can be performed by postmultiplication by a matrix in a standard form. To repeat:

- premultiplication: operation on rows;
- postmultiplication: operation on columns.

The matrix used to perform the operation is called an *elementary transformation matrix* or *elementary operator matrix*. Such a matrix is the identity matrix transformed by the corresponding operation performed on its unit rows, e_p^{T}, or columns, e_p.

In actual computations, we do not form the elementary transformation matrices explicitly, but their formulation allows us to discuss the operations in a systematic way and better understand the properties of the operations. Products of any of these elementary operator matrices can be used to effect more complicated transformations.

Operations on the rows are more common, and that is what we will discuss here, although operations on columns are completely analogous. These transformations of rows are called *elementary row operations*.

3.2.3.1 Interchange of Rows or Columns: Permutation Matrices

By first interchanging the rows or columns of a matrix, it may be possible to partition the matrix in such a way that the partitions have interesting or desirable properties. Also, in the course of performing computations on a matrix, it is often desirable to interchange the rows or columns of the matrix. (This is an instance of "pivoting", which will be discussed later, especially in Chap. 6.) In matrix computations, we almost never actually move data from one row or column to another; rather, the interchanges are effected by changing the indexes to the data.

Interchanging two rows of a matrix can be accomplished by premultiplying the matrix by a matrix that is the identity with those same two rows interchanged; for example,

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \\ a_{41} & a_{42} & a_{43} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{31} & a_{32} & a_{33} \\ a_{21} & a_{22} & a_{23} \\ a_{41} & a_{42} & a_{43} \end{bmatrix}.$$

The first matrix in the expression above is called an *elementary permutation matrix*. It is the identity matrix with its second and third rows (or columns) interchanged. An elementary permutation matrix, which is the identity with the p^{th} and q^{th} rows interchanged, is denoted by E_{pq}. That is, E_{pq} is the identity, except the p^{th} row is e_q^{T} and the q^{th} row is e_p^{T}. Note that $E_{pq} = E_{qp}$. Thus, for example, if the given matrix is $4 \times m$, to interchange the second and third rows, we use

$$E_{23} = E_{32} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

It is easy to see from the definition that an elementary permutation matrix is symmetric. Note that the notation E_{pq} does not indicate the order of the elementary permutation matrix; that must be specified in the context.

Premultiplying a matrix A by a (conformable) E_{pq} results in an interchange of the p^{th} and q^{th} rows of A as we see above. Any permutation of rows of A can be accomplished by successive premultiplications by elementary permutation matrices. Note that the order of multiplication matters. Although a given permutation can be accomplished by different elementary permutations, the number of elementary permutations that effect a given permutation is always either even or odd; that is, if an odd number of elementary permutations results in a given permutation, any other sequence of elementary permutations to yield the given permutation is also odd in number. Any given permutation can be effected by successive interchanges of adjacent rows.

Postmultiplying a matrix A by a (conformable) E_{pq} results in an interchange of the p^{th} and q^{th} columns of A:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \\ a_{41} & a_{42} & a_{43} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{13} & a_{12} \\ a_{21} & a_{23} & a_{22} \\ a_{31} & a_{33} & a_{32} \\ a_{41} & a_{43} & a_{42} \end{bmatrix}.$$

Note that

$$A = E_{pq}E_{pq}A = AE_{pq}E_{pq}; \tag{3.56}$$

that is, as an operator, an elementary permutation matrix is its own inverse operator: $E_{pq}E_{pq} = I$.

Because all of the elements of a permutation matrix are 0 or 1, the trace of an $n \times n$ elementary permutation matrix is $n - 2$.

The product of elementary permutation matrices is also a *permutation matrix* in the sense that it permutes several rows or columns. For example,

premultiplying A by the matrix $Q = E_{pq}E_{qr}$ will yield a matrix whose p^{th} row is the r^{th} row of the original A, whose q^{th} row is the p^{th} row of A, and whose r^{th} row is the q^{th} row of A. We often use the notation $E_{(\pi)}$ to denote a more general permutation matrix. This expression will usually be used generically, but sometimes we will specify the permutation, π.

A general permutation matrix (that is, a product of elementary permutation matrices) is not necessarily symmetric, but its transpose is also a permutation matrix. It is not necessarily its own inverse, but its permutations can be reversed by a permutation matrix formed by products of permutation matrices in the opposite order; that is,

$$E_{(\pi)}^{\mathrm{T}}E_{(\pi)} = I.$$

As a prelude to other matrix operations, we often permute both rows and columns, so we often have a representation such as

$$B = E_{(\pi_1)}AE_{(\pi_2)}, \tag{3.57}$$

where $E_{(\pi_1)}$ is a permutation matrix to permute the rows and $E_{(\pi_2)}$ is a permutation matrix to permute the columns. We use these kinds of operations to form a full rank partitioning as in equation (3.131) on page 104, to obtain an equivalent canonical form as in equation (3.151) on page 110 and LDU decomposition of a matrix as in equation (5.32) on page 246. These equations are used to determine the number of linearly independent rows and columns and to represent the matrix in a form with a maximal set of linearly independent rows and columns clearly identified.

3.2.3.2 The Vec-Permutation Matrix

A special permutation matrix is the matrix that transforms the vector $\mathrm{vec}(A)$ into $\mathrm{vec}(A^{\mathrm{T}})$. If A is $n \times m$, the matrix K_{nm} that does this is $nm \times nm$. We have

$$\mathrm{vec}(A^{\mathrm{T}}) = K_{nm}\mathrm{vec}(A). \tag{3.58}$$

The matrix K_{nm} is called the *nm vec-permutation matrix*.

3.2.3.3 Scalar Row or Column Multiplication

Often, numerical computations with matrices are more accurate if the rows have roughly equal norms. For this and other reasons, we often transform a matrix by multiplying one of its rows by a scalar. This transformation can also be performed by premultiplication by an elementary transformation matrix. For multiplication of the p^{th} row by the scalar, the elementary transformation matrix, which is denoted by $E_p(a)$, is the identity matrix in which the p^{th} diagonal element has been replaced by a. Thus, for example, if the given matrix is $4 \times m$, to multiply the second row by a, we use

$$E_2(a) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & a & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Postmultiplication of a given matrix by the multiplier matrix $E_p(a)$ results in the multiplication of the p^{th} column by the scalar. For this, $E_p(a)$ is a square matrix of order equal to the number of columns of the given matrix.

Note that the notation $E_p(a)$ does not indicate the number of rows and columns. This must be specified in the context.

Note that, if $a \neq 0$,

$$A = E_p(1/a)E_p(a)A, \tag{3.59}$$

that is, as an operator, the inverse operator is a row multiplication matrix on the same row and with the reciprocal as the multiplier.

3.2.3.4 Axpy Row or Column Transformations

The other elementary operation is an axpy on two rows and a replacement of one of those rows with the result

$$a_p \leftarrow aa_q + a_p.$$

This operation also can be effected by premultiplication by a matrix formed from the identity matrix by inserting the scalar in the (p, q) position. Such a matrix is denoted by $E_{pq}(a)$. Thus, for example, if the given matrix is $4 \times m$, to add a times the third row to the second row, we use

$$E_{23}(a) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & a & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Premultiplication of a matrix A by such a matrix,

$$E_{pq}(a)A, \tag{3.60}$$

yields a matrix whose p^{th} row is a times the q^{th} row plus the original row.

Given the 4×3 matrix $A = (a_{ij})$, we have

$$E_{23}(a)A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} + aa_{31} & a_{22} + aa_{32} & a_{23} + aa_{33} \\ a_{31} & a_{32} & a_{33} \\ a_{41} & a_{42} & a_{43} \end{bmatrix}.$$

Postmultiplication of a matrix A by an axpy operator matrix,

$$AE_{pq}(a),$$

yields a matrix whose q^{th} column is a times the p^{th} column plus the original column. For this, $E_{pq}(a)$ is a square matrix of order equal to the number of columns of the given matrix. Note that the column that is changed corresponds to the *second* subscript in $E_{pq}(a)$.

Note that

$$A = E_{pq}(-a)E_{pq}(a)A; \tag{3.61}$$

that is, as an operator, the inverse operator is the same axpy elementary operator matrix with the negative of the multiplier.

A common use of axpy operator matrices is to form a matrix with zeros in all positions of a given column below a given position in the column. These operations usually follow an operation by a scalar row multiplier matrix that puts a 1 in the position of interest. For example, given an $n \times m$ matrix A with $a_{ij} \neq 0$, to put a 1 in the (i, j) position and 0s in all positions of the j^{th} column below the i^{th} row, we form the product

$$E_{ni}(-a_{nj}) \cdots E_{i+1,i}(-a_{i+1,j})E_i(1/a_{ij})A. \tag{3.62}$$

This process is called *Gaussian elimination*.

The matrix

$$G_{ij} = E_{ni}(-a_{nj}) \cdots E_{i+1,i}(-a_{i+1,j})E_i(1/a_{ij}) \tag{3.63}$$

is called a *Gaussian transformation* matrix. Notice that it is lower triangular, and its inverse, also lower triangular, is

$$G_{ij}^{-1} = E_i(a_{ij})E_{i+1,i}(a_{i+1,j}) \cdots E_{ni}(a_{nj}) \tag{3.64}$$

Gaussian elimination is often performed sequentially down the diagonal elements of a matrix (see its use in the LU factorization on page 244, for example).

To form a matrix with zeros in all positions of a given column except one, we use additional matrices for the rows above the given element:

$$\widetilde{G}_{ij} = E_{ni}(-a_{nj}) \cdots E_{i+1,i}(-a_{i+1,j})E_{i-1,i}(-a_{i-1,j}) \cdots E_{1i}(-a_{1j})E_i(1/a_{ij}). \tag{3.65}$$

This is also called a Gaussian transformation matrix.

We can likewise zero out all elements in the i^{th} row except the one in the $(ij)^{\text{th}}$ position by similar postmultiplications.

If at some point $a_{ii} = 0$, the operations of equation (3.62) cannot be performed. In that case, we may first interchange the i^{th} row with the k^{th} row, where $k > i$ and $a_{ki} \neq 0$. Such an interchange is called *pivoting*. We will discuss pivoting in more detail on page 277 in Chap. 6.

As we mentioned above, in actual computations, we do not form the elementary transformation matrices explicitly, but their formulation allows us to discuss the operations in a systematic way and better understand the properties of the operations.

This is an instance of a principle that we will encounter repeatedly: *the form of a mathematical expression and the way the expression should be evaluated in actual practice may be quite different.*

These elementary transformations are the basic operations in Gaussian elimination, which is discussed in Sects. 5.7 and 6.2.1.

3.2.3.5 Elementary Operator Matrices: Summary of Notation and Properties

Because we have introduced various notation for elementary operator matrices, it may be worthwhile to review the notation. The notation is useful and I will use it from time to time, but unfortunately, there is no general form for the notation. I will generally use an "E" as the root symbol for the matrix, but the specific type is indicated by various other symbols.

Referring back to the listing of the types of operations on page 80, we have the various elementary operator matrices:

- E_{pq}: the interchange of rows p and q (E_{pq} is the same as E_{qp})

$$E_{pq} = E_{qp} = \begin{bmatrix} 1 & 0 & \cdots & 0 & \cdots & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & \cdots & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & \cdots & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 & 0 \\ 0 & 0 & \cdots & 0 & \cdots & 0 & \cdots & 0 & 1 \end{bmatrix} \begin{matrix} \\ \\ \\ p \\ \\ q \\ \\ \\ \end{matrix} \tag{3.66}$$

  It is symmetric,

$$E_{pq}^{\mathrm{T}} = E_{pq}, \tag{3.67}$$

  and it is its own inverse,

$$E_{pq}^{-1} = E_{pq}, \tag{3.68}$$

  that is, it is orthogonal.

  $E_{(\pi)}$: a general permutation of rows, where π denotes a permutation. We have

$$E_{(\pi)} = E_{p_1 q_1} \cdots E_{p_k q_k}, \text{ for some } p_1, \ldots, p_k \text{ and } q_1, \ldots, q_k. \tag{3.69}$$

- $E_p(a)$: multiplication of row p by a.

$$E_p(a) = \begin{bmatrix} 1 & 0 & \cdots & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & a & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & \cdots & 1 & 0 \\ 0 & 0 & \cdots & 0 & \cdots & 0 & 1 \end{bmatrix} p \tag{3.70}$$

Its inverse is

$$E_p^{-1}(a) = E_p(1/a). \tag{3.71}$$

- $E_{pq}(a)$: the replacement of row p by the sum of row p and a times row q. If $q > p$,

$$E_{pq}(a) = \begin{bmatrix} 1 & 0 & \cdots & 0 & \cdots & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & \cdots & a & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & \cdots & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 & 0 \\ 0 & 0 & \cdots & 0 & \cdots & 0 & \cdots & 0 & 1 \end{bmatrix} \begin{matrix} \\ \\ \\ p \\ \\ q \\ \\ \\ \end{matrix} \tag{3.72}$$

Its inverse is

$$E_{pq}^{-1}(a) = E_{pq}(-a). \tag{3.73}$$

Recall that these operations are effected by *premultiplication*. The same kinds of operations on the *columns* are effected by *postmultiplication*.

3.2.3.6 Determinants of Elementary Operator Matrices

The determinant of an elementary permutation matrix E_{pq} has only one term in the sum that defines the determinant (equation (3.24), page 66), and that term is 1 times σ evaluated at the permutation that exchanges p and q. As we have seen (page 67), this is an odd permutation; hence, for an elementary permutation matrix E_{pq},

$$\det(E_{pq}) = -1. \tag{3.74}$$

Because a general permutation matrix $E_{(\pi)}$ can be formed as the product of elementary permutation matrices which together form the permutation π, we have from equation (3.74)

$$\det(E_\pi) = \sigma(\pi), \tag{3.75}$$

where $\sigma(\pi) = 1$ if π is an even permutation and -1 otherwise, as defined in equation (3.23).

Because all terms in $\det(E_{pq}A)$ are exactly the same terms as in $\det(A)$ but with one different permutation in each term, we have

$$\det(E_{pq}A) = -\det(A).$$

More generally, if A and $E_{(\pi)}$ are $n \times n$ matrices, and $E_{(\pi)}$ is any permutation matrix (that is, any product of E_{pq} matrices), then $\det(E_{(\pi)}A)$ is either $\det(A)$ or $-\det(A)$ because all terms in $\det(E_{(\pi)}A)$ are exactly the same as the terms in $\det(A)$ but possibly with different signs because the permutations are different. In fact, the differences in the permutations are exactly the same as the permutation of $1, \ldots, n$ in $E_{(\pi)}$; hence,

$$\begin{aligned}\det(E_{(\pi)}A) &= \det(E_{(\pi)})\det(A)\\ &= \sigma(\pi)\det(A).\end{aligned}$$

(In equation (3.81) below, we will see that this equation holds more generally.)

The determinant of an elementary row multiplication matrix $E_p(a)$ is

$$\det(E_p(a)) = a. \tag{3.76}$$

If A and $E_p(a)$ are $n \times n$ matrices, then

$$\det(E_p(a)A) = a\det(A),$$

as we see from the definition of the determinant, equation (3.24). (Again, this also follows from the general result in equation (3.81) below.)

The determinant of an elementary axpy matrix $E_{pq}(a)$ is 1,

$$\det(E_{pq}(a)) = 1, \tag{3.77}$$

because the term consisting of the product of the diagonals is the only term in the determinant.

Now consider $\det(E_{pq}(a)A)$ for an $n \times n$ matrix A. Expansion in the minors (equation (3.30)) along the p^{th} row yields

$$\begin{aligned}\det(E_{pq}(a)A) &= \sum_{j=1}^{n}(a_{pj} + aa_{qj})(-1)^{p+j}\det(A_{(ij)})\\ &= \sum_{j=1}^{n} a_{pj}(-1)^{p+j}\det(A_{(ij)}) + a\sum_{j=1}^{n} a_{qj}(-1)^{p+j}\det(A_{(ij)}).\end{aligned}$$

From equation (3.32) on page 69, we see that the second term is 0, and since the first term is just the determinant of A, we have

$$\det(E_{pq}(a)A) = \det(A). \tag{3.78}$$

(Again, this also follows from the general result in equation (3.81) below. I have shown the steps in the specific case because I think they help to see the effect of the elementary operator matrix.)

3.2.4 The Trace of a Cayley Product That Is Square

A useful property of the trace for the matrices A and B that are conformable for the multiplications AB and BA is

$$\mathrm{tr}(AB) = \mathrm{tr}(BA). \tag{3.79}$$

This is obvious from the definitions of matrix multiplication and the trace. Note that A and B may not be square (so the trace is not defined for them), but if they are conformable for the multiplications, then both AB and BA are square.

Because of the associativity of matrix multiplication, this relation can be extended as

$$\mathrm{tr}(ABC) = \mathrm{tr}(BCA) = \mathrm{tr}(CAB) \tag{3.80}$$

for matrices A, B, and C that are conformable for the multiplications indicated. Notice that the individual matrices need not be square. This fact is very useful in working with quadratic forms, as in equation (3.90).

3.2.5 The Determinant of a Cayley Product of Square Matrices

An important property of the determinant is

$$\det(AB) = \det(A)\det(B) \tag{3.81}$$

if A and B are square matrices conformable for multiplication. We see this by first forming

$$\det\left(\begin{bmatrix} I & A \\ 0 & I \end{bmatrix}\begin{bmatrix} A & 0 \\ -I & B \end{bmatrix}\right) = \det\left(\begin{bmatrix} 0 & AB \\ -I & B \end{bmatrix}\right) \tag{3.82}$$

and then observing from equation (3.39) that the right-hand side is $\det(AB)$. Now consider the left-hand side. The matrix that is the first factor on the left-hand side is a product of elementary axpy transformation matrices; that is, it is a matrix that when postmultiplied by another matrix merely adds multiples of rows in the lower part of the matrix to rows in the upper part of the matrix. If A and B are $n \times n$ (and so the identities are likewise $n \times n$), the full matrix is the product:

$$\begin{bmatrix} I & A \\ 0 & I \end{bmatrix} = E_{1,n+1}(a_{11}) \cdots E_{1,2n}(a_{1n}) E_{2,n+1}(a_{21}) \cdots E_{2,2n}(a_{2,n}) \cdots E_{n,2n}(a_{nn}).$$

Hence, applying equation (3.78) recursively, we have

$$\det\left(\begin{bmatrix} I & A \\ 0 & I \end{bmatrix}\begin{bmatrix} A & 0 \\ -I & B \end{bmatrix}\right) = \det\left(\begin{bmatrix} A & 0 \\ -I & B \end{bmatrix}\right),$$

and from equation (3.38) we have

$$\det\left(\begin{bmatrix} A & 0 \\ -I & B \end{bmatrix}\right) = \det(A)\det(B),$$

and so finally we have equation (3.81).

From equation (3.81), we see that if A and B are square matrices conformable for multiplication, then

$$\det(AB) = \det(BA). \tag{3.83}$$

(Recall, in general, even in the case of square matrices, $AB \neq BA$.) This equation is to be contrasted with equation (3.79), $\mathrm{tr}(AB) = \mathrm{tr}(BA)$, which does not even require that the matrices be square. A simple counterexample for nonsquare matrices is $\det(xx^{\mathrm{T}}) \neq \det(x^{\mathrm{T}}x)$, where x is a vector with at least two elements. (Here, think of the vector as an $n \times 1$ matrix. This counterexample can be seen in various ways. One way is to use a fact that we will encounter on page 117, and observe that $\det(xx^{\mathrm{T}}) = 0$ for any x with at least two elements.)

3.2.6 Multiplication of Matrices and Vectors

It is often convenient to think of a vector as a matrix with only one element in one of its dimensions. This provides for an immediate extension of the definitions of transpose and matrix multiplication to include vectors as either or both factors. In this scheme, we follow the *convention that a vector corresponds to a column*; that is, if x is a vector and A is a matrix, Ax or $x^{\mathrm{T}}A$ may be well-defined, but neither xA nor Ax^{T} would represent anything, except in the case when all dimensions are 1. In some computer systems for matrix algebra, these conventions are not enforced; in others, they are not. (R, for example sometimes does and sometimes does not; see the discussion beginning on page 572.) The alternative notation $x^{\mathrm{T}}y$ we introduced earlier for the dot product or inner product, $\langle x, y\rangle$, of the vectors x and y is consistent with this paradigm.

Vectors and matrices are fundamentally different kinds of mathematical objects. In general, it is not relevant to say that a vector is a "column" or a "row"; it is merely a one-dimensional (or rank 1) object. We will continue to write vectors as $x = (x_1, \ldots, x_n)$, but this does not imply that the vector is a "row vector". Matrices with just one row or one column are different objects from vectors. We represent a matrix with one row in a form such as $Y = [y_{11} \ldots y_{1n}]$, and we represent a matrix with one column in a form such as $Z = \begin{bmatrix} z_{11} \\ \vdots \\ z_{m1} \end{bmatrix}$ or as $Z = [z_{11} \ldots z_{m1}]^{\mathrm{T}}$.

(Compare the notation in equations (1.1) and (1.2) on page 4.)

3.2.6.1 The Matrix/Vector Product as a Linear Combination

If we represent the vectors formed by the columns of an $n \times m$ matrix A as $a_1, \ldots, a_m$, the matrix/vector product Ax is a linear combination of these columns of A:

$$Ax = \sum_{i=1}^{m} x_i a_i. \tag{3.84}$$

(Here, each x_i is a scalar, and each a_i is a vector.)

Given the equation $Ax = b$, we have $b \in \text{span}(A)$; that is, the n-vector b is in the k-dimensional column space of A, where $k \leq m$.

3.2.6.2 The Matrix as a Mapping on Vector Spaces

In this chapter we have considered matrices to be fundamental objects. Only after defining operations on matrices themselves have we defined an operation by a matrix on a vector. Another way of thinking about matrices is as a class of functions or mappings on vector spaces. In this approach, we give primacy to the vector spaces.

Let $\mathcal{V}_1$ and $\mathcal{V}_2$ be vector spaces of order m and n respectively. Then an $n \times m$ matrix A is a function from $\mathcal{V}_1$ to $\mathcal{V}_2$ defined for $x \in \mathcal{V}_1$ as

$$x \mapsto Ax. \tag{3.85}$$

Matrices are "transformations" of vectors. There is nothing essentially different in this development of concepts about matrices; it does, however, motivate terminology based in geometry that we will use from time to time ("rotations", "projections", and so on; see Sect. 5.3).

A matrix changes the "direction" of a vector. The cosine of the angle between the vector x and the vector Ax is the correlation

$$\text{Cor}(x, Ax) = \frac{(x - \bar{x})^{\mathrm{T}} A (x - \bar{x})}{(x - \bar{x})^{\mathrm{T}} (x - \bar{x})},$$

see page 51. (This expression is the Rayleigh quotient, $R_A(x - \bar{x})$, page 157.)

3.2.7 Outer Products

The *outer product* of the vectors x and y is the matrix

$$xy^{\mathrm{T}}. \tag{3.86}$$

Note that the definition of the outer product does not require the vectors to be of equal length. Note also that while the inner product is commutative, the outer product is not commutative (although it does have the property $xy^{\mathrm{T}} = (yx^{\mathrm{T}})^{\mathrm{T}}$).

While the inner product is a mapping from $\mathbb{R}^n \times \mathbb{R}^n$ to $\mathbb{R}$, the outer product of two vectors is a mapping

$$\mathbb{R}^n \times \mathbb{R}^m \to \mathcal{M} \subseteq \mathbb{R}^{n \times m},$$

where $\mathcal{M}$ is the set of $n \times m$ matrices of rank one. (We will define and discuss matrix rank in Sect. 3.3, beginning on page 99. Also, see Exercise 3.14.)

A very common outer product is of a vector with itself:

$$xx^{\mathrm{T}}.$$

The outer product of a vector with itself is obviously a symmetric matrix.

We should again note some subtleties of differences in the types of objects that result from operations. If A and B are matrices conformable for the operation, the product $A^{\mathrm{T}}B$ is a *matrix* even if both A and B are $n \times 1$ and so the result is 1×1. For the vectors x and y and matrix C, however, $x^{\mathrm{T}}y$ and $x^{\mathrm{T}}Cy$ are *scalars*; hence, the dot product and a quadratic form are *not* the same as the result of a matrix multiplication. The dot product is a scalar, and the result of a matrix multiplication is a matrix. The outer product of vectors is a matrix, even if both vectors have only one element. Nevertheless, as we have mentioned before, *we will treat a one by one matrix or a vector with only one element as a scalar whenever it is convenient to do so.*

3.2.8 Bilinear and Quadratic Forms: Definiteness

Given a matrix A of conformable shape, a variation of the vector dot product, $x^{\mathrm{T}}Ay$, is called a *bilinear form*, and the special bilinear form $x^{\mathrm{T}}Ax$ is called a *quadratic form*. Note

$$x^{\mathrm{T}}A^{\mathrm{T}}x = x^{\mathrm{T}}Ax \qquad \text{and} \qquad x^{\mathrm{T}}A^{\mathrm{T}}y = y^{\mathrm{T}}Ax \neq x^{\mathrm{T}}Ay \quad \text{in general.}$$

Although in the definition of quadratic form we do not require A to be symmetric—because for a given value of x and a given value of the quadratic form $x^{\mathrm{T}}Ax$ there is a unique symmetric matrix A_s such that $x^{\mathrm{T}}A_sx = x^{\mathrm{T}}Ax$—we generally work only with symmetric matrices in dealing with quadratic forms. (The matrix A_s is $\frac{1}{2}(A + A^{\mathrm{T}})$; see Exercise 3.3.) Quadratic forms correspond to sums of squares and hence play an important role in statistical applications.

3.2.8.1 Nonnegative Definite and Positive Definite Matrices

A symmetric matrix A such that for any (conformable and real) vector x the quadratic form $x^{\mathrm{T}}Ax$ is nonnegative, that is,

$$x^{\mathrm{T}}Ax \geq 0, \tag{3.87}$$

is called a *nonnegative definite matrix*. (There is another term, "positive semidefinite matrix" and its acronym PSD, that is often used to mean "nonnegative definite matrix", but the term is not used consistently in the literature. I will generally avoid the term "semidefinite".) We denote the fact that A is nonnegative definite by

$$A \succeq 0.$$

(Note that we consider $0_{n\times n}$ to be nonnegative definite.)

A symmetric matrix A such that for any (conformable) vector $x \neq 0$ the quadratic form

$$x^{\mathrm{T}}Ax > 0 \tag{3.88}$$

is called a *positive definite matrix*. We denote the fact that A is positive definite by

$$A \succ 0.$$

(Recall that $A \geq 0$ and $A > 0$ mean, respectively, that all elements of A are nonnegative and positive.)

Nonnegative and positive definite matrices are very important in applications. We will encounter them from time to time in this chapter, and then we will discuss more of their properties in Sect. 8.3.

In this book we use the terms "nonnegative definite" and "positive definite" only for symmetric matrices. In other literature, these terms may be used more generally; that is, for any (square) matrix that satisfies (3.87) or (3.88).

3.2.8.2 Ordinal Relations among Symmetric Matrices

When A and B are symmetric matrices of the same order, we write $A \succeq B$ to mean $A - B \succeq 0$ and $A \succ B$ to mean $A - B \succ 0$.

The $\succeq$ relationship is a *partial ordering* and the $\succ$ relationship is transitive; that is, if for conformable matrices, $A \succ B$ and $B \succ C$, then $A \succ C$ (See Exercise 8.2 on page 396; also compare ordinal relations among vectors, page 16.)

3.2.8.3 The Trace of Inner and Outer Products

The invariance of the trace to permutations of the factors in a product (equation (3.79)) is particularly useful in working with bilinear and quadratic forms. Let A be an $n \times m$ matrix, x be an n-vector, and y be an m-vector. Because the bilinear form is a scalar (or a 1×1 matrix), and because of the invariance, we have the very useful fact

$$\begin{aligned} x^{\mathrm{T}}Ay &= \mathrm{tr}(x^{\mathrm{T}}Ay) \\ &= \mathrm{tr}(Ayx^{\mathrm{T}}). \end{aligned} \tag{3.89}$$

A common instance is when A is square and $x = y$. We have for the quadratic form the equality

$$x^{\mathrm{T}}Ax = \mathrm{tr}(Axx^{\mathrm{T}}). \tag{3.90}$$

In equation (3.90), if A is the identity I, we have that the inner product of a vector with itself is the trace of the outer product of the vector with itself, that is,

$$x^{\mathrm{T}}x = \mathrm{tr}(xx^{\mathrm{T}}). \tag{3.91}$$

Also, by letting A be the identity in equation (3.90), we have an alternative way of showing that for a given vector x and any scalar a, the norm $\|x - a\|$ is minimized when $a = \bar{x}$:

$$(x-a)^{\mathrm{T}}(x-a) = \mathrm{tr}(x_{\mathrm{c}}x_{\mathrm{c}}^{\mathrm{T}}) + n(a-\bar{x})^2. \tag{3.92}$$

(Here, "$\bar{x}$" denotes the mean of the elements in x, and "x_{c}" is $x - \bar{x}$. Compare this with equation (2.71) on page 48.)

3.2.9 Anisometric Spaces

In Sect. 2.1, we considered various properties of vectors that depend on the inner product, such as orthogonality of two vectors, norms of a vector, angles between two vectors, and distances between two vectors. All of these properties and measures are invariant to the orientation of the vectors; the space is *isometric* with respect to a Cartesian coordinate system. Noting that for real vectors the inner product is the bilinear form $x^{\mathrm{T}}Iy$, we have a heuristic generalization to an anisometric space. Suppose, for example, that the scales of the coordinates differ; say, a given distance along one axis in the natural units of the axis is equivalent (in some sense depending on the application) to twice that distance along another axis, again measured in the natural units of the axis. The properties derived from the inner product, such as a norm and a metric, may correspond to the application better if we use a bilinear form in which the matrix reflects the different effective distances along the coordinate axes. A diagonal matrix whose entries have relative values corresponding to the inverses of the relative scales of the axes may be more useful. Instead of $x^{\mathrm{T}}y$, we may use $x^{\mathrm{T}}Dy$, where D is this diagonal matrix.

Rather than differences in scales being just in the directions of the coordinate axes, more generally we may think of anisometries being measured by general (but perhaps symmetric) matrices. (The covariance and correlation matrices defined on page 367 come to mind.) Any such matrix to be used in this context should be positive definite because we will generalize the dot product, which is necessarily nonnegative, in terms of a quadratic form. A bilinear form $x^{\mathrm{T}}Ay$ may correspond more closely to the properties of the application than the standard inner product.

3.2.9.1 Conjugacy

We define orthogonality of two vectors real vectors x and y with respect to A by

$$x^{\mathrm{T}}Ay = 0. \tag{3.93}$$

In this case, we say x and y are A-*conjugate*.

The L_2 norm of a vector is the square root of the quadratic form of the vector with respect to the identity matrix. A generalization of the L_2 vector norm, called an *elliptic norm* or a *conjugate norm*, is defined for the vector x as the square root of the quadratic form $x^{\mathrm{T}}Ax$ for any symmetric positive definite matrix A. It is sometimes denoted by $\|x\|_A$:

$$\|x\|_A = \sqrt{x^{\mathrm{T}}Ax}. \tag{3.94}$$

It is easy to see that $\|x\|_A$ satisfies the definition of a norm given on page 25. If A is a diagonal matrix with elements $w_i \geq 0$, the elliptic norm is the weighted L_2 norm of equation (2.37).

The elliptic norm yields an *elliptic metric* in the usual way of defining a metric in terms of a norm. The distance between the real vectors x and y with respect to A is $\sqrt{(x-y)^{\mathrm{T}}A(x-y)}$. It is easy to see that this satisfies the definition of a metric given on page 32.

A metric that is widely useful in statistical applications is the Mahalanobis distance, which uses a covariance matrix as the scale for a given space. (The sample covariance matrix is defined in equation (8.67) on page 367.) If S is the covariance matrix, the Mahalanobis distance, with respect to that matrix, between the vectors x and y is

$$\sqrt{(x-y)^{\mathrm{T}}S^{-1}(x-y)}. \tag{3.95}$$

3.2.10 Other Kinds of Matrix Multiplication

The most common kind of product of two matrices is the Cayley product, and when we speak of matrix multiplication without qualification, we mean the Cayley product. Three other types of matrix multiplication that are useful are *Hadamard multiplication*, *Kronecker multiplication*, and *inner product multiplication*.

3.2.10.1 The Hadamard Product

Hadamard multiplication is defined for matrices of the same shape as the multiplication of each element of one matrix by the corresponding element of the other matrix. Hadamard multiplication is often denoted by $\odot$; for two matrices $A_{n\times m}$ and $B_{n\times m}$ we have

$$A \odot B = \begin{bmatrix} a_{11}b_{11} & \dots & a_{1m}b_{1m} \\ \vdots & \dots & \vdots \\ a_{n1}b_{n1} & \dots & a_{nm}b_{nm} \end{bmatrix}.$$

Hadamard multiplication immediately inherits the commutativity, associativity, and distribution over addition of the ordinary multiplication of the underlying field of scalars. Hadamard multiplication is also called array multiplication and element-wise multiplication. Hadamard matrix multiplication is a mapping

$$\mathbb{R}^{n\times m} \times \mathbb{R}^{n\times m} \to \mathbb{R}^{n\times m}.$$

The identity for Hadamard multiplication is the matrix of appropriate shape whose elements are all 1s.

3.2.10.2 The Kronecker Product

Kronecker multiplication, denoted by $\otimes$, is defined for any two matrices $A_{n\times m}$ and $B_{p\times q}$ as

$$A \otimes B = \begin{bmatrix} a_{11}B & \dots & a_{1m}B \\ \vdots & \dots & \vdots \\ a_{n1}B & \dots & a_{nm}B \end{bmatrix}.$$

The Kronecker product of A and B is $np \times mq$; that is, Kronecker matrix multiplication is a mapping

$$\mathbb{R}^{n\times m} \times \mathbb{R}^{p\times q} \to \mathbb{R}^{np\times mq}.$$

The Kronecker product is also called the "right direct product" or just *direct product*. (A left direct product is a Kronecker product with the factors reversed. In some of the earlier literature, "Kronecker product" was used to mean a left direct product.) Note the similarity of the Kronecker product of matrices with the direct product of sets, defined on page 5, in the sense that the result is formed from ordered pairs of elements from the two operands.

Kronecker multiplication is not commutative, but it is associative and it is distributive over addition, as we will see below. (Again, this parallels the direct product of sets.)

The identity for Kronecker multiplication is the 1×1 matrix with the element 1; that is, it is the same as the scalar 1.

We can understand the properties of the Kronecker product by expressing the (i, j) element of $A \otimes B$ in terms of the elements of A and B,

$$(A \otimes B)_{i,j} = A_{\lfloor (i-1)/p \rfloor + 1,\ \lfloor (j-1)/q \rfloor + 1} B_{i - p\lfloor (i-1)/p \rfloor,\ j - q\lfloor (j-1)/q \rfloor}. \tag{3.96}$$

Some additional properties of Kronecker products that are immediate results of the definition are, assuming the matrices are conformable for the indicated operations,

$$\begin{aligned} (aA) \otimes (bB) &= ab(A \otimes B) \\ &= (abA) \otimes B \\ &= A \otimes (abB), \text{ for scalars } a, b, \end{aligned} \tag{3.97}$$

$$(A+B)\otimes(C) = A\otimes C + B\otimes C, \tag{3.98}$$

$$(A\otimes B)\otimes C = A\otimes(B\otimes C), \tag{3.99}$$

$$(A\otimes B)^{\mathrm{T}} = A^{\mathrm{T}}\otimes B^{\mathrm{T}}, \tag{3.100}$$

$$(A\otimes B)(C\otimes D) = AC\otimes BD. \tag{3.101}$$

$$I\otimes A = \mathrm{diag}(A,\ldots,A). \tag{3.102}$$

$$A\otimes I = (a_{ij}I)\,. \tag{3.103}$$

These properties are all easy to see by using equation (3.96) to express the (i,j) element of the matrix on either side of the equation, taking into account the size of the matrices involved. For example, in the first equation, if A is $n\times m$ and B is $p\times q$, the (i,j) element on the left-hand side is

$$aA_{[(i-1)/p]+1,\,[(j-1)/q]+1}bB_{i-p[(i-1)/p],\,j-q[(j-1)/q]}$$

and that on the right-hand side is

$$abA_{[(i-1)/p]+1,\,[(j-1)/q]+1}B_{i-p[(i-1)/p],\,j-q[(j-1)/q]}.$$

They are all this easy! Hence, they are Exercise 3.6.

The determinant of the Kronecker product of two square matrices $A_{n\times n}$ and $B_{m\times m}$ has a simple relationship to the determinants of the individual matrices:

$$\det(A\otimes B) = \det(A)^m\det(B)^n. \tag{3.104}$$

The proof of this, like many facts about determinants, is straightforward but involves tedious manipulation of cofactors. The manipulations in this case can be facilitated by using the vec-permutation matrix. See Harville (1997) for a detailed formal proof.

From equation (3.100) we see that the Kronecker product of symmetric matrices is symmetric.

Another property of the Kronecker product of square matrices is

$$\mathrm{tr}(A\otimes B) = \mathrm{tr}(A)\mathrm{tr}(B). \tag{3.105}$$

This is true because the trace of the product is merely the sum of all possible products of the diagonal elements of the individual matrices.

The Kronecker product and the vec function often find uses in the same application. For example, an $n\times m$ normal random matrix X with parameters

M, Σ, and Ψ can be expressed in terms of an ordinary np-variate normal random variable $Y = \mathrm{vec}(X)$ with parameters $\mathrm{vec}(M)$ and $\Sigma \otimes \Psi$. (We discuss matrix random variables briefly on page 220. For a fuller discussion, the reader is referred to a text on matrix random variables such as Carmeli 1983, or Kollo and von Rosen 2005.)

A useful relationship between the vec function and Kronecker multiplication is

$$\mathrm{vec}(ABC) = (C^{\mathrm{T}} \otimes A)\mathrm{vec}(B) \tag{3.106}$$

for matrices A, B, and C that are conformable for the multiplication indicated. This is easy to show and is left as an exercise.

3.2.10.3 The Inner Product of Matrices

An inner product of two matrices of the same shape is defined as the sum of the dot products of the vectors formed from the columns of one matrix with vectors formed from the corresponding columns of the other matrix; that is, if $a_1, \ldots, a_m$ are the columns of A and $b_1, \ldots, b_m$ are the columns of B, then the *inner product* of A and B, denoted $\langle A, B \rangle$, is

$$\langle A, B \rangle = \sum_{j=1}^{m} \langle a_j, b_j \rangle. \tag{3.107}$$

Similarly as for vectors (page 23), the inner product is sometimes called a "dot product", and the notation $A \cdot B$ is sometimes used to denote the matrix inner product. (I generally try to avoid use of the term dot product for matrices because the term may be used differently by different people. In Matlab, for example, "dot product", implemented in the `dot` function, can refer either to $1 \times m$ matrix consisting of the individual terms in the sum in equation (3.107), or to the $n \times 1$ matrix consisting of the dot products of the vectors formed from the rows of A and B. In the NumPy linear algebra package, the `dot` function implements Cayley multiplication! This is probably because someone working with Python realized the obvious fact that the defining equation of Cayley multiplication, equation (3.43) on page 75, is actually the dot product of the vector formed from the elements in the i^{th} row in the first matrix and the vector formed from the elements in the j^{th} column in the first matrix.)

For real matrices, equation (3.107) can be written as

$$\langle A, B \rangle = \sum_{j=1}^{m} a_j^{\mathrm{T}} b_j. \tag{3.108}$$

As in the case of the product of vectors, the product of matrices defined as in equation (3.108) over the complex field is not an inner product because the first property (on page 24 or as listed below) does not hold.

For conformable matrices A, B, and C, we can easily confirm that this product satisfies the general properties of an inner product listed on page 24:

- If $A \neq 0$, $\langle A, A\rangle > 0$, and $\langle 0, A\rangle = \langle A, 0\rangle = \langle 0, 0\rangle = 0$.
- $\langle A, B\rangle = \langle B, A\rangle$.
- $\langle sA, B\rangle = s\langle A, B\rangle$, for a scalar s.
- $\langle (A + B), C\rangle = \langle A, C\rangle + \langle B, C\rangle$.

As with any inner product (restricted to objects in the field of the reals), its value is a real number. Thus the matrix inner product is a mapping

$$\mathbb{R}^{n\times m} \times \mathbb{R}^{n\times m} \to \mathbb{R}.$$

We see from the definition above that the inner product of real matrices satisfies

$$\langle A, B\rangle = \mathrm{tr}(A^{\mathrm{T}}B), \tag{3.109}$$

which could alternatively be taken as the definition.

Rewriting the definition of $\langle A, B\rangle$ as $\sum_{j=1}^{m}\sum_{i=1}^{n} a_{ij}b_{ij}$, we see that for real matrices

$$\langle A, B\rangle = \langle A^{\mathrm{T}}, B^{\mathrm{T}}\rangle. \tag{3.110}$$

Like any inner product, inner products of matrices obey the Cauchy-Schwarz inequality (see inequality (2.26), page 24),

$$\langle A, B\rangle \leq \langle A, A\rangle^{\frac{1}{2}}\langle B, B\rangle^{\frac{1}{2}}, \tag{3.111}$$

with equality holding only if $A = 0$ or $B = sA$ for some scalar s.

3.2.10.4 Orthonormal Matrices

In Sect. 2.1.8, we defined orthogonality and orthonormality of two or more vectors in terms of inner products. We can likewise define an orthogonal binary relationship between two matrices in terms of inner products of matrices. We say the matrices A and B of the same shape are *orthogonal to each other* if

$$\langle A, B\rangle = 0. \tag{3.112}$$

We also use the term "orthonormal" to refer to matrices that are orthogonal to each other and for which each has an inner product with itself of 1. In Sect. 3.7, we will define orthogonality as a unary property of matrices. The term "orthogonal", when applied to matrices, generally refers to that property rather than the binary property we have defined here. "Orthonormal", on the other hand, refers to the binary property.

3.2.10.5 Orthonormal Basis: Fourier Expansion

On page 64 we identified a vector space of matrices and defined a basis for the space $\mathbb{R}^{n\times m}$. If $\{U_1, \ldots, U_k\}$ is a basis set for $\mathcal{M} \subseteq \mathbb{R}^{n\times m}$ with the property that $\langle U_i, U_j\rangle = 0$ for $i \neq j$ and $\langle U_i, U_i\rangle = 1$, then the set is an orthonormal basis set.

If A is an $n \times m$ matrix, with the Fourier expansion

$$A = \sum_{i=1}^{k} c_i U_i, \tag{3.113}$$

we have, analogous to equation (2.59) on page 41,

$$c_i = \langle A, U_i \rangle. \tag{3.114}$$

The c_i have the same properties (such as the Parseval identity, equation (2.60), for example) as the Fourier coefficients in any orthonormal expansion. Best approximations within $\mathcal{M}$ can also be expressed as truncations of the sum in equation (3.113) as in equation (2.63). The objective of course is to reduce the truncation error, and the optimality of the Fourier expansion in this regard discussed on page 42 holds in the matrix case as well. (The norms in Parseval's identity and in measuring the goodness of an approximation are matrix norms in this case. We discuss matrix norms in Sect. 3.9 beginning on page 164.)

3.3 Matrix Rank and the Inverse of a Matrix

The linear dependence or independence of the vectors forming the rows or columns of a matrix is an important characteristic of the matrix.

The maximum number of linearly independent vectors (those forming either the rows or the columns) is called the *rank* of the matrix. We use the notation

$$\text{rank}(A)$$

to denote the rank of the matrix A. (We have used the term "rank" before to denote dimensionality of an array. "Rank" as we have just defined it applies only to a matrix or to a set of vectors, and this is by far the more common meaning of the word. The meaning is clear from the context, however.)

Because multiplication by a nonzero scalar does not change the linear independence of vectors, for the scalar a with $a \neq 0$, we have

$$\text{rank}(aA) = \text{rank}(A). \tag{3.115}$$

From results developed in Sect. 2.1, we see that for the $n \times m$ matrix A,

$$\text{rank}(A) \leq \min(n, m). \tag{3.116}$$

The rank of the zero matrix is 0, and the rank of any nonzero matrix is positive.

3.3.1 Row Rank and Column Rank

We have defined matrix rank in terms of numbers of linearly independent rows *or* columns. This is because the number of linearly independent rows is the same as the number of linearly independent columns. Although we may use the terms "row rank" or "column rank", the single word "rank" is sufficient because they are the same. To see this, assume we have an $n \times m$ matrix A and that there are exactly p linearly independent rows and exactly q linearly independent columns. We can permute the rows and columns of the matrix so that the first p rows are linearly independent rows and the first q columns are linearly independent and the remaining rows or columns are linearly dependent on the first ones. (Recall that applying the same permutation to all of the elements of each vector in a set of vectors does not change the linear dependencies over the set.) After these permutations, we have a matrix B with submatrices W, X, Y, and Z,

$$B = \begin{bmatrix} W_{p\times q} & X_{p\times m-q} \\ Y_{n-p\times q} & Z_{n-p\times m-q} \end{bmatrix}, \tag{3.117}$$

where the rows of $R = [W|X]$ correspond to p linearly independent m-vectors and the columns of $C = \begin{bmatrix} W \\ Y \end{bmatrix}$ correspond to q linearly independent n-vectors. Without loss of generality, we can assume $p \leq q$. Now, if $p < q$, it must be the case that the columns of W are linearly dependent because there are q of them, but they have only p elements. Therefore, there is some q-vector $a \neq 0$ such that $Wa = 0$. Now, since the rows of R are the full set of linearly independent rows, any row in $[Y|Z]$ can be expressed as a linear combination of the rows of R, and any row in Y can be expressed as a linear combination of the rows of W. This means, for some $n-p \times p$ matrix T, that $Y = TW$. In this case, however, $Ca = 0$. But this contradicts the assumption that the columns of C are linearly independent; therefore it cannot be the case that $p < q$. We conclude therefore that $p = q$; that is, that the maximum number of linearly independent rows is the same as the maximum number of linearly independent columns.

Because the row rank, the column rank, and the rank of A are all the same, we have

$$\text{rank}(A) = \dim(\mathcal{V}(A)), \tag{3.118}$$

$$\text{rank}(A^{\text{T}}) = \text{rank}(A), \tag{3.119}$$

$$\dim(\mathcal{V}(A^{\text{T}})) = \dim(\mathcal{V}(A)). \tag{3.120}$$

(Note, of course, that in general $\mathcal{V}(A^{\text{T}}) \neq \mathcal{V}(A)$; the orders of the vector spaces are possibly different.)

3.3.2 Full Rank Matrices

If the rank of a matrix is the same as its smaller dimension, we say the matrix is of *full rank*. In the case of a nonsquare matrix, we may say the matrix is of full row rank or full column rank just to emphasize which is the smaller number.

If a matrix is not of full rank, we say it is *rank deficient* and define the *rank deficiency* as the difference between its smaller dimension and its rank.

A full rank matrix that is square is called *nonsingular*, and one that is not nonsingular is called *singular*.

A square matrix that is either row or column diagonally dominant is nonsingular. The proof of this is Exercise 3.9. (It's easy!)

A positive definite matrix is nonsingular. The proof of this is Exercise 3.10.

Later in this section, we will identify additional properties of square full rank matrices. (For example, they have inverses and their determinants are nonzero.)

3.3.3 Rank of Elementary Operator Matrices and Matrix Products Involving Them

Because within any set of rows of an elementary operator matrix (see Sect. 3.2.3), for some given column, only one of those rows contains a nonzero element, the elementary operator matrices are all obviously of full rank (with the proviso that $a \neq 0$ in $E_p(a)$).

Furthermore, the rank of the product of any given matrix with an elementary operator matrix is the same as the rank of the given matrix. To see this, consider each type of elementary operator matrix in turn. For a given matrix A, the set of rows of $E_{pq}A$ is the same as the set of rows of A; hence, the rank of $E_{pq}A$ is the same as the rank of A. Likewise, the set of columns of AE_{pq} is the same as the set of columns of A; hence, again, the rank of AE_{pq} is the same as the rank of A.

The set of rows of $E_p(a)A$ for $a \neq 0$ is the same as the set of rows of A, except for one, which is a nonzero scalar multiple of the corresponding row of A; therefore, the rank of $E_p(a)A$ is the same as the rank of A. Likewise, the set of columns of $AE_p(a)$ is the same as the set of columns of A, except for one, which is a nonzero scalar multiple of the corresponding row of A; therefore, again, the rank of $AE_p(a)$ is the same as the rank of A.

Finally, the set of rows of $E_{pq}(a)A$ for $a \neq 0$ is the same as the set of rows of A, except for one, which is a nonzero scalar multiple of some row of A added to the corresponding row of A; therefore, the rank of $E_{pq}(a)A$ is the same as the rank of A. Likewise, we conclude that the rank of $AE_{pq}(a)$ is the same as the rank of A.

We therefore have that if P and Q are the products of elementary operator matrices,

$$\operatorname{rank}(PAQ) = \operatorname{rank}(A). \tag{3.121}$$

On page 113, we will extend this result to products by any full rank matrices.

3.3.4 The Rank of Partitioned Matrices, Products of Matrices, and Sums of Matrices

The partitioning in equation (3.117) leads us to consider partitioned matrices in more detail.

3.3.4.1 Rank of Partitioned Matrices and Submatrices

Let the matrix A be partitioned as

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

where any pair of submatrices in a column or row may be null (that is, where for example, it may be the case that $A = [A_{11}|A_{12}]$). Then the number of linearly independent rows of A must be at least as great as the number of linearly independent rows of $[A_{11}|A_{12}]$ and the number of linearly independent rows of $[A_{21}|A_{22}]$. By the properties of subvectors in Sect. 2.1.1, the number of linearly independent rows of $[A_{11}|A_{12}]$ must be at least as great as the number of linearly independent rows of A_{11} or A_{21}. We could go through a similar argument relating to the number of linearly independent columns and arrive at the inequality

$$\operatorname{rank}(A_{ij}) \leq \operatorname{rank}(A). \tag{3.122}$$

Furthermore, we see that

$$\operatorname{rank}(A) \leq \operatorname{rank}([A_{11}|A_{12}]) + \operatorname{rank}([A_{21}|A_{22}]) \tag{3.123}$$

because $\operatorname{rank}(A)$ is the number of linearly independent columns of A, which is less than or equal to the number of linearly independent rows of $[A_{11}|A_{12}]$ plus the number of linearly independent rows of $[A_{12}|A_{22}]$. Likewise, we have

$$\operatorname{rank}(A) \leq \operatorname{rank}\left(\begin{bmatrix} A_{11} \\ A_{21} \end{bmatrix}\right) + \operatorname{rank}\left(\begin{bmatrix} A_{12} \\ A_{22} \end{bmatrix}\right). \tag{3.124}$$

In a similar manner, by merely counting the number of independent rows, we see that, if

$$\mathcal{V}\left([A_{11}|A_{12}]^{\mathrm{T}}\right) \perp \mathcal{V}\left([A_{21}|A_{22}]^{\mathrm{T}}\right),$$

then

$$\operatorname{rank}(A) = \operatorname{rank}([A_{11}|A_{12}]) + \operatorname{rank}([A_{21}|A_{22}]); \tag{3.125}$$

and, if

$$\mathcal{V}\left(\begin{bmatrix} A_{11} \\ A_{21} \end{bmatrix}\right) \perp \mathcal{V}\left(\begin{bmatrix} A_{12} \\ A_{22} \end{bmatrix}\right),$$

then

$$\operatorname{rank}(A) = \operatorname{rank}\left(\begin{bmatrix} A_{11} \\ A_{21} \end{bmatrix}\right) + \operatorname{rank}\left(\begin{bmatrix} A_{12} \\ A_{22} \end{bmatrix}\right). \tag{3.126}$$

3.3.4.2 An Upper Bound on the Rank of Products of Matrices

Because the columns of the product AB are linear combinations of the columns of A, it is clear that

$$\mathcal{V}(AB) \subseteq \mathcal{V}(A). \tag{3.127}$$

The rank of the product of two matrices is less than or equal to the lesser of the ranks of the two:

$$\text{rank}(AB) \leq \min(\text{rank}(A), \text{rank}(B)). \tag{3.128}$$

This follows from equation (3.127). We can also show this by separately considering two cases for the $n \times k$ matrix A and the $k \times m$ matrix B. In one case, we assume k is at least as large as n and $n \leq m$, and in the other case we assume $k < n \leq m$. In both cases, we represent the rows of AB as k linear combinations of the rows of B.

From inequality (3.128), we see that the rank of a nonzero outer product matrix (that is, a matrix formed as the outer product of two nonzero vectors) is 1.

The bound in inequality (3.128) is sharp, as we can see by exhibiting matrices A and B such that $\text{rank}(AB) = \min(\text{rank}(A), \text{rank}(B))$, as you are asked to do in Exercise 3.12a.

Inequality (3.128) provides a useful upper bound on $\text{rank}(AB)$. In Sect. 3.3.11, we will develop a lower bound on $\text{rank}(AB)$.

3.3.4.3 An Upper and a Lower Bound on the Rank of Sums of Matrices

The rank of the sum of two matrices is less than or equal to the sum of their ranks; that is,

$$\text{rank}(A + B) \leq \text{rank}(A) + \text{rank}(B). \tag{3.129}$$

We can see this by observing that

$$A + B = [A|B] \begin{bmatrix} I \\ I \end{bmatrix},$$

and so $\text{rank}(A + B) \leq \text{rank}([A|B])$ by equation (3.128), which in turn is $\leq \text{rank}(A) + \text{rank}(B)$ by equation (3.124).

The bound in inequality (3.129) is sharp, as we can see by exhibiting matrices A and B such that $\text{rank}(A + B) = \text{rank}(A) + \text{rank}(B)$, as you are asked to do in Exercise 3.12c.

Using inequality (3.129) and the fact that $\text{rank}(-B) = \text{rank}(B)$, we write $\text{rank}(A-B) \leq \text{rank}(A)+\text{rank}(B)$, and so, replacing A in (3.129) by $A+B$, we have $\text{rank}(A) \leq \text{rank}(A+B)+\text{rank}(B)$, or $\text{rank}(A+B) \geq \text{rank}(A)-\text{rank}(B)$. By a similar procedure, we get $\text{rank}(A + B) \geq \text{rank}(B) - \text{rank}(A)$, or

$$\text{rank}(A+B) \geq |\text{rank}(A) - \text{rank}(B)|. \tag{3.130}$$

The bound in inequality (3.130) is sharp, as we can see by exhibiting matrices A and B such that $\text{rank}(A+B) = |\text{rank}(A) - \text{rank}(B)|$, as you are asked to do in Exercise 3.12d.

3.3.5 Full Rank Partitioning

As we saw above, the matrix W in the partitioned B in equation (3.117) is square; in fact, it is $r \times r$, where r is the rank of B:

$$B = \begin{bmatrix} W_{r\times r} & X_{r\times m-r} \\ Y_{n-r\times r} & Z_{n-r\times m-r} \end{bmatrix}. \tag{3.131}$$

This is called a *full rank partitioning* of B.

The matrix B in equation (3.131) has a very special property: the full set of linearly independent rows are the first r rows, and the full set of linearly independent columns are the first r columns.

Any rank r matrix can be put in the form of equation (3.131) by using permutation matrices as in equation (3.57), assuming that $r \geq 1$. That is, if A is a nonzero matrix, there is a matrix of the form of B above that has the same rank. For some permutation matrices $E_{(\pi_1)}$ and $E_{(\pi_2)}$,

$$B = E_{(\pi_1)} A E_{(\pi_2)}. \tag{3.132}$$

The inverses of these permutations coupled with the full rank partitioning of B form a full rank partitioning of the original matrix A.

For a square matrix of rank r, this kind of partitioning implies that there is a full rank $r \times r$ principal submatrix, and the principal submatrix formed by including any of the remaining diagonal elements is singular. The principal minor formed from the full rank principal submatrix is nonzero, but if the order of the matrix is greater than r, a principal minor formed from a submatrix larger than $r \times r$ is zero.

The partitioning in equation (3.131) is of general interest, and we will use this type of partitioning often. We express an equivalent partitioning of a transformed matrix in equation (3.151) below.

The same methods as above can be used to form a full rank square submatrix of any order less than or equal to the rank. That is, if the $n \times m$ matrix A is of rank r and $q \leq r$, we can form

$$E_{(\pi_r)} A E_{(\pi_c)} = \begin{bmatrix} S_{q\times q} & T_{q\times m-q} \\ U_{n-q\times r} & V_{n-q\times m-q} \end{bmatrix}, \tag{3.133}$$

where S is of rank q.

It is obvious that the rank of a matrix can never exceed its smaller dimension (see the discussion of linear independence on page 12). Whether or not

a matrix has more rows than columns, the rank of the matrix is the same as the dimension of the column space of the matrix. (As we have just seen, the dimension of the column space is necessarily the same as the dimension of the row space, but the order of the column space is different from the order of the row space unless the matrix is square.)

3.3.6 Full Rank Matrices and Matrix Inverses

We have already seen that full rank matrices have some important properties. In this section, we consider full rank matrices and matrices that are their Cayley multiplicative inverses.

3.3.6.1 Solutions of Linear Equations

Important applications of vectors and matrices involve systems of linear equations:

$$\begin{aligned} a_{11}x_1 + \cdots + a_{1m}x_m &\stackrel{?}{=} b_1 \\ \vdots \qquad\qquad \vdots \qquad &\vdots \\ a_{n1}x_1 + \cdots + a_{nm}x_m &\stackrel{?}{=} b_n \end{aligned} \tag{3.134}$$

or

$$Ax \stackrel{?}{=} b. \tag{3.135}$$

In this system, A is called the coefficient matrix. An x that satisfies this system of equations is called a *solution* to the system. For given A and b, a solution may or may not exist. From equation (3.84), a solution exists if and only if the n-vector b is in the k-dimensional column space of A, where $k \leq m$. A system for which a solution exists is said to be *consistent*; otherwise, it is *inconsistent*.

We note that if $Ax = b$, for any conformable y,

$$y^{\mathrm{T}}Ax = 0 \Longleftrightarrow y^{\mathrm{T}}b = 0. \tag{3.136}$$

3.3.6.2 Consistent Systems

A linear system $A_{n\times m}x = b$ is consistent if and only if

$$\operatorname{rank}([A \mid b]) = \operatorname{rank}(A). \tag{3.137}$$

We can see this following the argument above that $b \in \mathcal{V}(A)$; that is, the space spanned by the columns of A is the same as that spanned by the columns of A and the vector b. Therefore b must be a linear combination of the columns of A, and furthermore, the linear combination is a solution to the system $Ax = b$. (Note, of course, that it is not necessary that it be a unique linear combination.)

Equation (3.137) implies the equivalence of the conditions

$$[A \mid b]y = 0 \text{ for some } y \neq 0 \quad \Leftrightarrow \quad Ax = 0 \text{ for some } x \neq 0. \tag{3.138}$$

A special case that yields equation (3.137) for any b is

$$\operatorname{rank}(A_{n\times m}) = n, \tag{3.139}$$

and so if A is of full row rank, the system is consistent regardless of the value of b. In this case, of course, the number of rows of A must be no greater than the number of columns (by inequality (3.116)). A square system in which A is nonsingular is clearly consistent. (The condition of consistency is also called "compatibility" of the system; that is, the linear system $Ax = b$ is said to be *compatible* if it is consistent.)

A generalization of the linear system $Ax = b$ is $AX = B$, where B is an $n \times k$ matrix. This is the same as k systems $Ax_1 = b_1, \ldots, Ax_k = b_k$, where the x_i and the b_i are the columns of the respective matrices. Consistency of $AX = B$, as above, is the condition for a solution in X to exist, and in that case the system is also said to be compatible.

It is clear that the system $AX = B$ is consistent if each of the $Ax_i = b_i$ systems is consistent. Furthermore, if the system is consistent, then every linear relationship among the rows of A exists among the rows of B; that is, for any c such that $c^{\mathrm{T}}A = 0$, then $c^{\mathrm{T}}B = 0$. To see this, let c be such that $c^{\mathrm{T}}A = 0$. We then have $c^{\mathrm{T}}AX = c^{\mathrm{T}}B = 0$, and so the same linear relationship that exists among the rows of A exists among the rows of B.

As above for $Ax = b$, we also see that the system $AX = B$ is consistent if and only if any of the following conditions hold:

$$\mathcal{V}(B) \subseteq \mathcal{V}(A) \tag{3.140}$$

$$\mathcal{V}([A \mid B]) = \mathcal{V}(A) \tag{3.141}$$

$$\operatorname{rank}([A \mid B]) = \operatorname{rank}(A). \tag{3.142}$$

These relations imply that if $AX = B$ is consistent, then for any conformable vector c,

$$c^{\mathrm{T}}A = 0 \iff c^{\mathrm{T}}B = 0. \tag{3.143}$$

It is clear that this condition also implies that $AX = B$ is consistent (because right-hand implication of the condition implies the relationship (3.140)).

We discuss methods for solving linear systems in Sect. 3.5 and in Chap. 6. In the next section, we consider a special case of $n \times n$ (square) A when equation (3.139) is satisfied (that is, when A is nonsingular).

3.3.6.3 Matrix Inverses

Let A be an $n \times n$ nonsingular matrix, and consider the linear systems

$$Ax_i = e_i,$$

where e_i is the i^{th} unit vector. For each e_i, this is a consistent system by equation (3.137).

We can represent all n such systems as

$$A\left[x_1|\cdots|x_n\right] = \left[e_1|\cdots|e_n\right]$$

or

$$AX = I_n,$$

and this full system must have a solution; that is, there must be an X such that $AX = I_n$. Because $AX = I$, we call X a "right inverse" of A. The matrix X must be $n \times n$ and nonsingular (because I is); hence, it also has a right inverse, say Y, and $XY = I$. From $AX = I$, we have $AXY = Y$, so $A = Y$, and so finally $XA = I$; that is, the right inverse of A is also the "left inverse". We will therefore just call it the *inverse* of A and denote it as A^{-1}. This is the Cayley multiplicative inverse. Hence, for an $n \times n$ nonsingular matrix A, we have a matrix A^{-1} such that

$$A^{-1}A = AA^{-1} = I_n. \tag{3.144}$$

The inverse of the nonsingular square matrix A is unique. (This follows from the argument above about a "right inverse" and a "left inverse".)

We have already encountered the idea of a matrix inverse in our discussions of elementary transformation matrices. The matrix that performs the inverse of the elementary operation is the inverse matrix.

From the definitions of the inverse and the transpose, we see that

$$(A^{-1})^{\mathrm{T}} = (A^{\mathrm{T}})^{-1}, \tag{3.145}$$

and because in applications we often encounter the inverse of a transpose of a matrix, we adopt the notation

$$A^{-\mathrm{T}}$$

to denote the inverse of the transpose.

In the linear system (3.135), if $n = m$ and A is nonsingular, the solution is

$$x = A^{-1}b. \tag{3.146}$$

For scalars, the combined operations of inversion and multiplication are equivalent to the single operation of division. From the analogy with scalar operations, we sometimes denote AB^{-1} by A/B. Because matrix multiplication is not commutative, we often use the notation "\" to indicate the combined operations of inversion and multiplication on the left; that is, $B\backslash A$ is the same as $B^{-1}A$. The solution given in equation (3.146) is also sometimes represented as $A\backslash b$.

We discuss the solution of systems of equations in Chap. 6, but here we will point out that when we write an expression that involves computations to

evaluate it, such as $A^{-1}b$ or $A\backslash b$, the form of the expression does not specify how to do the computations. This is an instance of a principle that we will encounter repeatedly: *the form of a mathematical expression and the way the expression should be evaluated in actual practice may be quite different.*

3.3.6.4 Nonsquare Full Rank Matrices: Right and Left Inverses

Suppose A is $n \times m$ and $\text{rank}(A) = n$; that is, $n \leq m$ and A is of full row rank. Then $\text{rank}([A \,|\, e_i]) = \text{rank}(A)$, where e_i is the i^{th} unit vector of length n; hence the system

$$Ax_i = e_i$$

is consistent for each e_i, and, as before, we can represent all n such systems as

$$A\left[x_1|\cdots|x_n\right] = \left[e_1|\cdots|e_n\right]$$

or

$$AX = I_n.$$

As above, there must be an X such that $AX = I_n$, and we call X a *right inverse* of A. The matrix X must be $m \times n$ and it must be of rank n (because I is). This matrix is not necessarily the inverse of A, however, because A and X may not be square. We denote the right inverse of A as

$$A^{-\text{R}}.$$

Furthermore, we could only have solved the system AX if A was of full row rank because $n \leq m$ and $n = \text{rank}(I) = \text{rank}(AX) \leq \text{rank}(A)$. To summarize, A has a right inverse if and only if A is of full row rank.

Now, suppose A is $n \times m$ and $\text{rank}(A) = m$; that is, $m \leq n$ and A is of full column rank. Writing $YA = I_m$ and reversing the roles of the coefficient matrix and the solution matrix in the argument above, we have that Y exists and is a *left inverse* of A. We denote the left inverse of A as

$$A^{-\text{L}}.$$

Also, using a similar argument as above, we see that the matrix A has a left inverse if and only if A is of full column rank.

We also note that if AA^{T} is of full rank, the right inverse of A is

$$A^{-\text{R}} = A^{\text{T}}(AA^{\text{T}})^{-1}. \tag{3.147}$$

Likewise, if $A^{\text{T}}A$ is of full rank, the left inverse of A is

$$A^{-\text{L}} = (A^{\text{T}}A)^{-1}A^{\text{T}}. \tag{3.148}$$

3.3.7 Full Rank Factorization

For a given matrix A, it is often of interest to find matrices $A_1, \ldots, A_k$ such that $A_1, \ldots, A_k$ have some useful properties and $A = A_1 \cdots A_k$. This is called a *factorization* or *decomposition* of A. (We will usually use these two words interchangeably; that is, by "decomposition", we will usually mean "multiplicative decomposition". Occasionally we will be interested in an additive decomposition of a matrix, as in Cochran's theorem, discussed on page 401 and later in Sect. 9.2.3.)

In most cases, the number of factors in $A = A_1 \cdots A_k$ is either 2 or 3. In this chapter, we will discuss some factorizations as they arise naturally in the development, and then in Chap. 5 we will discuss factorizations in more detail.

The partitioning of an $n \times m$ matrix as in equation (3.131) on page 104 leads to an interesting factorization of a matrix. Recall that we had an $n \times m$ matrix B partitioned as

$$B = \begin{bmatrix} W_{r\times r} & X_{r\times m-r} \\ Y_{n-r\times r} & Z_{n-r\times m-r} \end{bmatrix},$$

where r is the rank of B, W is of full rank, the rows of $R = [W|X]$ span the full row space of B, and the columns of $C = \begin{bmatrix} W \\ Y \end{bmatrix}$ span the full column space of B.

Therefore, for some T, we have $[Y|Z] = TR$, and for some S, we have $\begin{bmatrix} X \\ Z \end{bmatrix} = CS$. From this, we have $Y = TW$, $Z = TX$, $X = WS$, and $Z = YS$, so $Z = TWS$. Since W is nonsingular, we have $T = YW^{-1}$ and $S = W^{-1}X$, so $Z = YW^{-1}X$.

We can therefore write the partitions as

$$\begin{aligned} B &= \begin{bmatrix} W & X \\ Y & YW^{-1}X \end{bmatrix} \\ &= \begin{bmatrix} I \\ YW^{-1} \end{bmatrix} W \begin{bmatrix} I \mid W^{-1}X \end{bmatrix}. \end{aligned} \tag{3.149}$$

From this, we can form two equivalent factorizations of B:

$$B = \begin{bmatrix} W \\ Y \end{bmatrix} \begin{bmatrix} I \mid W^{-1}X \end{bmatrix} = \begin{bmatrix} I \\ YW^{-1} \end{bmatrix} \begin{bmatrix} W \mid X \end{bmatrix}.$$

The matrix B has a very special property: the full set of linearly independent rows are the first r rows, and the full set of linearly independent columns are the first r columns. We have seen, however, that any matrix A of rank r can be put in this form, and $A = E_{(\pi_2)}BE_{(\pi_1)}$ for an $n \times n$ permutation matrix $E_{(\pi_2)}$ and an $m \times m$ permutation matrix $E_{(\pi_1)}$.

We therefore have, for the $n \times m$ matrix A with rank r, two equivalent factorizations,

$$A = \begin{bmatrix} Q_1 W \\ Q_2 Y \end{bmatrix} \left[P_1 \mid W^{-1} X P_2 \right]$$
$$= \begin{bmatrix} Q_1 \\ Q_2 Y W^{-1} \end{bmatrix} \left[W P_1 \mid X P_2 \right],$$

both of which are in the general form

$$A_{n\times m} = L_{n\times r}\, R_{r\times m}, \tag{3.150}$$

where L is of full column rank and R is of full row rank. This is called a *full rank factorization* of the matrix A. We will use a full rank factorization in proving various properties of matrices. We will consider other factorizations later in this chapter and in Chap. 5 that have more practical uses in computations.

3.3.8 Equivalent Matrices

Matrices of the same order that have the same rank are said to be *equivalent matrices*.

3.3.8.1 Equivalent Canonical Forms

For any $n \times m$ matrix A with $\text{rank}(A) = r > 0$, by combining the permutations that yield equation (3.131) with other operations, we have, for some matrices P and Q that are products of various elementary operator matrices,

$$PAQ = \begin{bmatrix} I_r & 0 \\ 0 & 0 \end{bmatrix}. \tag{3.151}$$

This is called an *equivalent canonical form* of A, and it exists for any matrix A that has at least one nonzero element (which is the same as requiring $\text{rank}(A) > 0$).

We can see by construction that an equivalent canonical form exists for any $n \times m$ matrix A that has a nonzero element. First, assume $a_{ij} \neq 0$. By two successive permutations, we move a_{ij} to the $(1,1)$ position; specifically, $(E_{i1} A E_{1j})_{11} = a_{ij}$. We then divide the first row by a_{ij}; that is, we form $E_1(1/a_{ij}) E_{i1} A E_{1j}$. We then proceed with a sequence of $n-1$ premultiplications by axpy matrices to zero out the first column of the matrix, as in expression (3.62), followed by a sequence of $(m-1)$ postmultiplications by axpy matrices to zero out the first row. We then have a matrix of the form

$$\begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & & & \\ \vdots & & X & \\ 0 & & & \end{bmatrix}. \tag{3.152}$$

If $X = 0$, we are finished; otherwise, we perform the same kinds of operations on the $(n-1) \times (m-1)$ matrix X and continue until we have the form of equation (3.151).

The matrices P and Q in equation (3.151) are not unique. The order in which they are built from elementary operator matrices can be very important in preserving the accuracy of the computations.

Although the matrices P and Q in equation (3.151) are not unique, the equivalent canonical form itself (the right-hand side) is obviously unique because the only thing that determines it, aside from the shape, is the r in I_r, and that is just the rank of the matrix. There are two other, more general, equivalent forms that are often of interest. These equivalent forms, "row echelon form" and "Hermite form", are not unique. A matrix R is said to be in *row echelon form*, or just *echelon form*, if

- $r_{ij} = 0$ for $i > j$, and
- if k is such that $r_{ik} \neq 0$ and $r_{il} = 0$ for $l < k$, then $r_{i+1,j} = 0$ for $j \leq k$.

A matrix in echelon form is upper triangular. An upper triangular matrix H is said to be in *Hermite form* if

- $h_{ii} = 0$ or 1,
- if $h_{ii} = 0$, then $h_{ij} = 0$ for all j, and
- if $h_{ii} = 1$, then $h_{ki} = 0$ for all $k \neq i$.

If H is in Hermite form, then $H^2 = H$, as is easily verified. (A matrix H such that $H^2 = H$ is said to be *idempotent*. We discuss idempotent matrices beginning on page 352.) Another, more specific, equivalent form, called the *Jordan form*, is a special row echelon form based on eigenvalues, which we show on page 151.

Any of these equivalent forms is useful in determining the rank of a matrix. Each form may have special uses in proving properties of matrices. We will often make use of the equivalent canonical form in other sections of this chapter.

3.3.8.2 Products with a Nonsingular Matrix

It is easy to see that if A is a square full rank matrix (that is, A is nonsingular), and if B and C are conformable matrices for the multiplications AB and CA, respectively, then

$$\text{rank}(AB) = \text{rank}(B) \tag{3.153}$$

and

$$\text{rank}(CA) = \text{rank}(C). \tag{3.154}$$

This is true because, for a given conformable matrix B, by the inequality (3.128), we have $\text{rank}(AB) \leq \text{rank}(B)$. Forming $B = A^{-1}AB$, and again applying the inequality, we have $\text{rank}(B) \leq \text{rank}(AB)$; hence, $\text{rank}(AB) =$

rank(B). Likewise, for a square full rank matrix A, we have rank(CA) = rank(C). (Here, we should recall that all matrices are real.)

On page 113, we give a more general result for products with general full rank matrices.

3.3.8.3 A Factorization Based on an Equivalent Canonical Form

Elementary operator matrices and products of them are of full rank and thus have inverses. When we introduced the matrix operations that led to the definitions of the elementary operator matrices in Sect. 3.2.3, we mentioned the inverse operations, which would then define the inverses of the matrices.

The matrices P and Q in the equivalent canonical form of the matrix A, PAQ in equation (3.151), have inverses. From an equivalent canonical form of a matrix A with rank r, we therefore have the equivalent canonical factorization of A:

$$A = P^{-1} \begin{bmatrix} I_r & 0 \\ 0 & 0 \end{bmatrix} Q^{-1}. \tag{3.155}$$

A factorization based on an equivalent canonical form is also a full rank factorization and could be written in the same form as equation (3.150).

3.3.8.4 Equivalent Forms of Symmetric Matrices

If A is symmetric, the equivalent form in equation (3.151) can be written as $PAP^{\mathrm{T}} = \mathrm{diag}(I_r, 0)$ and the equivalent canonical factorization of A in equation (3.155) can be written as

$$A = P^{-1} \begin{bmatrix} I_r & 0 \\ 0 & 0 \end{bmatrix} P^{-\mathrm{T}}. \tag{3.156}$$

These facts follow from the same process that yielded equation (3.151) for a general matrix.

Also a full rank factorization for a symmetric matrix, as in equation (3.150), can be given as

$$A = LL^{\mathrm{T}}. \tag{3.157}$$

3.3.9 Multiplication by Full Rank Matrices

We have seen that a matrix has an inverse if it is square and of full rank. Conversely, it has an inverse only if it is square and of full rank. We see that a matrix that has an inverse must be square because $A^{-1}A = AA^{-1}$, and we see that it must be full rank by the inequality (3.128). In this section, we consider other properties of full rank matrices. In some cases, we require the matrices to be square, but in other cases, these properties hold whether or not they are square.

Using matrix inverses allows us to establish important properties of products of matrices in which at least one factor is a full rank matrix.

3.3.9.1 Products with a General Full Rank Matrix

If C is a full column rank matrix and if B is a matrix conformable for the multiplication CB, then

$$\text{rank}(CB) = \text{rank}(B). \tag{3.158}$$

To see this, consider a full rank $n \times m$ matrix C with $\text{rank}(C) = m$ (that is, $m \leq n$) and let B be conformable for the multiplication CB. Because C is of full column rank, it has a left inverse (see page 108); call it $C^{-\text{L}}$, and so $C^{-\text{L}}C = I_m$. From inequality (3.128), we have $\text{rank}(CB) \leq \text{rank}(B)$, and applying the inequality again, we have $\text{rank}(B) = \text{rank}(C^{-\text{L}}CB) \leq \text{rank}(CB)$; hence $\text{rank}(CB) = \text{rank}(B)$.

If R is a full row rank matrix and if B is a matrix conformable for the multiplication BR, then

$$\text{rank}(BR) = \text{rank}(B). \tag{3.159}$$

To see this, consider a full rank $n \times m$ matrix R with $\text{rank}(R) = n$ (that is, $n \leq m$) and let B be conformable for the multiplication BR. Because R is of full row rank, it has a right inverse; call it $R^{-\text{R}}$, and so $RR^{-\text{R}} = I_n$. From inequality (3.128), we have $\text{rank}(BR) \leq \text{rank}(B)$, and applying the inequality again, we have $\text{rank}(B) = \text{rank}(BRR^{-\text{L}}) \leq \text{rank}(BR)$; hence $\text{rank}(BR) = \text{rank}(B)$.

To state this more simply:

- Premultiplication of a given matrix by a full column rank matrix yields a product with the same rank as the given matrix, and postmultiplication of a given matrix by a full row rank matrix yields a product with the same rank as the given matrix.

From this we see that, given any matrix B, if A is a square matrix of full rank that is compatible for the multiplication $AB = D$, then B and D are equivalent matrices. (And, of course, a similar statement for postmultiplication by a full-rank matrix holds.)

Furthermore, if the matrix B is square and A is a square matrix of the same order that is full rank, then

$$\text{rank}(AB) = \text{rank}(BA) = \text{rank}(B). \tag{3.160}$$

3.3.9.2 Preservation of Positive Definiteness

A certain type of product of a full rank matrix and a positive definite matrix preserves not only the rank, but also the positive definiteness: if A is $n \times n$ and positive definite, and C is $n \times m$ and of rank m (hence, $m \leq n$), then $C^{\text{T}}AC$ is positive definite. (Recall from inequality (3.88) that a matrix A is positive definite if it is symmetric and for any $x \neq 0$, $x^{\text{T}}Ax > 0$.)

To see this, assume matrices A and C as described. Let x be any m-vector such that $x \neq 0$, and let $y = Cx$. Because C is of full column rank, $y \neq 0$. We have

$$\begin{aligned} x^{\mathrm{T}}(C^{\mathrm{T}}AC)x &= (Cx)^{\mathrm{T}}A(Cx) \\ &= y^{\mathrm{T}}Ay \\ &> 0. \end{aligned} \tag{3.161}$$

Therefore, since $C^{\mathrm{T}}AC$ is symmetric,

- if A is positive definite and C is of full column rank, then $C^{\mathrm{T}}AC$ is positive definite.

Furthermore, we have the converse:

- if $C^{\mathrm{T}}AC$ is positive definite, then C is of full column rank,

for otherwise there exists an $x \neq 0$ such that $Cx = 0$, and so $x^{\mathrm{T}}(C^{\mathrm{T}}AC)x = 0$.

3.3.9.3 The General Linear Group

Consider the set of all square $n \times n$ full rank matrices together with the usual (Cayley) multiplication. As we have seen, this set is closed under multiplication. (The product of two square matrices of full rank is of full rank, and of course the product is also square.) Furthermore, the (multiplicative) identity is a member of this set, and each matrix in the set has a (multiplicative) inverse in the set; therefore, the set together with the usual multiplication is a mathematical structure called a *group*. (See any text on modern algebra.) This group is called the *general linear group* and is denoted by $\mathcal{G}L(n)$. The *order* of the group is n, the order of the square matrices in the group. General group-theoretic properties can be used in the derivation of properties of these full-rank matrices. Note that this group is not commutative.

We note that all matrices in the general linear group of order n are equivalent.

As we mentioned earlier (before we had considered inverses in general), if A is an $n \times n$ matrix and if A^{-1} exists, we define A^0 to be I_n (otherwise, A^0 does not exist).

The $n \times n$ elementary operator matrices are members of the general linear group $\mathcal{G}L(n)$.

The elements in the general linear group are matrices and, hence, can be viewed as transformations or operators on n-vectors. Another set of linear operators on n-vectors are the doubletons (A, v), where A is an $n \times n$ full-rank matrix and v is an n-vector. As an operator on $x \in \mathbb{R}^n$, (A, v) is the transformation $Ax + v$, which preserves affine spaces. Two such operators, (A, v) and (B, w), are combined by composition: $(A, v)((B, w)(x)) = ABx + Aw + v$. The set of such doubletons together with composition forms a group, called the *affine group*. It is denoted by $\mathcal{A}L(n)$. A subset of the elements of

the affine group with the same first element, together with the axpy operator, constitute a *quotient space*.

3.3.10 Gramian Matrices: Products of the Form $A^{\mathrm{T}}A$

Given a real matrix A, an important matrix product is $A^{\mathrm{T}}A$. (This is called a *Gramian matrix*, or just a *Gram matrix*. We will discuss this kind of matrix in more detail beginning on page 359. I should note here that this is not a definition of "Gramian" or "Gram"; these terms have more general meanings, but they do include any matrix expressible as $A^{\mathrm{T}}A$.)

We first note that AA^{T} is a Gramian matrix, and has the same properties as $A^{\mathrm{T}}A$ with any dependencies on A being replaced with dependencies on A^{T}.

3.3.10.1 General Properties of Gramian Matrices

Gramian matrices have several interesting properties. First of all, we note that for any A, because

$$(A^{\mathrm{T}}A)_{ij} = a_{*i}^{\mathrm{T}}a_{*j} = a_{*j}^{\mathrm{T}}a_{*i} = (A^{\mathrm{T}}A)_{ji} \quad \text{(recall notation, page 600)},$$

$A^{\mathrm{T}}A$ is symmetric, and hence has all of the useful properties of symmetric matrices. (These properties are shown in various places in this book, but are summarized conveniently in Sect. 8.2 beginning on page 340.) Furthermore, $A^{\mathrm{T}}A$ is nonnegative definite, as we see by observing that for any y, $y^{\mathrm{T}}(A^{\mathrm{T}}A)y = (Ay)^{\mathrm{T}}(Ay) \geq 0$.

Another interesting property of a Gramian matrix is that, for any matrices C and D (that are conformable for the operations indicated),

$$CA^{\mathrm{T}}A = DA^{\mathrm{T}}A \iff CA^{\mathrm{T}} = DA^{\mathrm{T}}. \tag{3.162}$$

The implication from right to left is obvious, and we can see the left to right implication by writing

$$(CA^{\mathrm{T}}A - DA^{\mathrm{T}}A)(C^{\mathrm{T}} - D^{\mathrm{T}}) = (CA^{\mathrm{T}} - DA^{\mathrm{T}})(CA^{\mathrm{T}} - DA^{\mathrm{T}})^{\mathrm{T}},$$

and then observing that if the left-hand side is null, then so is the right-hand side, and if the right-hand side is null, then $CA^{\mathrm{T}} - DA^{\mathrm{T}} = 0$ because $A^{\mathrm{T}}A = 0 \implies A = 0$, as above.

Similarly, we have

$$A^{\mathrm{T}}AC = A^{\mathrm{T}}AD \iff AC = AD. \tag{3.163}$$

3.3.10.2 Rank of $A^{\mathrm{T}}A$

Consider the linear system $A^{\mathrm{T}}AX = A^{\mathrm{T}}B$. Suppose that c is such that $c^{\mathrm{T}}A^{\mathrm{T}}A = 0$. Then by (3.162), $c^{\mathrm{T}}A^{\mathrm{T}} = 0$, which by (3.143) on page 106, implies

that $A^{\mathrm{T}}AX = A^{\mathrm{T}}B$ is consistent. Letting $B = I$, we have that $A^{\mathrm{T}}AX = A^{\mathrm{T}}$ is consistent.

Now if $A^{\mathrm{T}}AX = A^{\mathrm{T}}$, for any conformable matrix K,

$$\mathcal{V}(K^{\mathrm{T}}A^{\mathrm{T}}) = \mathcal{V}(K^{\mathrm{T}}A^{\mathrm{T}}AX).$$

By (3.127) on page 103, $\mathcal{V}(K^{\mathrm{T}}A^{\mathrm{T}}AX) \subseteq \mathcal{V}(K^{\mathrm{T}}A^{\mathrm{T}}A)$ and $\mathcal{V}(K^{\mathrm{T}}A^{\mathrm{T}}A) \subseteq \mathcal{V}(K^{\mathrm{T}}A^{\mathrm{T}})$; hence $\mathcal{V}(K^{\mathrm{T}}A^{\mathrm{T}}A) = \mathcal{V}(K^{\mathrm{T}}A^{\mathrm{T}})$. By similar arguments applied to the transposes we have $\mathcal{V}(A^{\mathrm{T}}AK) = \mathcal{V}(AK)$.

With $K = I$, this yields

$$\operatorname{rank}(A^{\mathrm{T}}A) = \operatorname{rank}(A). \tag{3.164}$$

In a similar manner, we have $\operatorname{rank}(AA^{\mathrm{T}}) = \operatorname{rank}(A)$, and hence,

$$\operatorname{rank}(A^{\mathrm{T}}A) = \operatorname{rank}(AA^{\mathrm{T}}). \tag{3.165}$$

It is clear from the statements above that $(A^{\mathrm{T}}A)$ is of full rank if and only if A is of full column rank.

We also see that $A^{\mathrm{T}}A$ is positive definite, that is, for any $y \neq 0$ $y^{\mathrm{T}}A^{\mathrm{T}}Ay > 0$, if and only if A is of full column rank. This follows from (3.167), and if A is of full column rank, $Ay = 0 \Rightarrow y = 0$.

3.3.10.3 Zero Matrices and Equations Involving Gramians

First of all, for any $n \times m$ matrix A, we have the fact that $A^{\mathrm{T}}A = 0$ if and only if $A = 0$. We see this by noting that if $A = 0$, then $\operatorname{tr}(A^{\mathrm{T}}A) = 0$. Conversely, if $\operatorname{tr}(A^{\mathrm{T}}A) = 0$, then $a_{ij}^2 = 0$ for all i, j, and so $a_{ij} = 0$, that is, $A = 0$. Summarizing, we have

$$\operatorname{tr}(A^{\mathrm{T}}A) = 0 \;\Leftrightarrow\; A = 0 \tag{3.166}$$

and

$$A^{\mathrm{T}}A = 0 \;\Leftrightarrow\; A = 0. \tag{3.167}$$

Now consider the equation $A^{\mathrm{T}}A = 0$. We have for any conformable B and C

$$A^{\mathrm{T}}A(B - C) = 0.$$

Multiplying by $B^{\mathrm{T}} - C^{\mathrm{T}}$ and factoring $(B^{\mathrm{T}} - C^{\mathrm{T}})A^{\mathrm{T}}A(B - C)$, we have

$$(AB - AC)^{\mathrm{T}}(AB - AC) = 0;$$

hence, from (3.167), we have $AB - AC = 0$. Furthermore, if $AB - AC = 0$, then clearly $A^{\mathrm{T}}A(B - C) = 0$. We therefore conclude that

$$A^{\mathrm{T}}AB = A^{\mathrm{T}}AC \;\Leftrightarrow\; AB = AC. \tag{3.168}$$

By the same argument, we have

$$BA^{\mathrm{T}}A = CA^{\mathrm{T}}A \;\Leftrightarrow\; BA^{\mathrm{T}} = CA^{\mathrm{T}}.$$

From equation (3.164), we have another useful fact for Gramian matrices. The system

$$A^{\mathrm{T}}Ax = A^{\mathrm{T}}b \tag{3.169}$$

is consistent for any A and b.

3.3.11 A Lower Bound on the Rank of a Matrix Product

Equation (3.128) gives an upper bound on the rank of the product of two matrices; the rank cannot be greater than the rank of either of the factors. Now, using equation (3.155), we develop a lower bound on the rank of the product of two matrices if one of them is square.

If A is $n \times n$ (that is, square) and B is a matrix with n rows, then

$$\operatorname{rank}(AB) \geq \operatorname{rank}(A) + \operatorname{rank}(B) - n. \tag{3.170}$$

We see this by first letting $r = \operatorname{rank}(A)$, letting P and Q be matrices that form an equivalent canonical form of A (see equation (3.155)), and then forming

$$C = P^{-1}\begin{bmatrix} 0 & 0 \\ 0 & I_{n-r} \end{bmatrix} Q^{-1},$$

so that $A + C = P^{-1}Q^{-1}$. Because P^{-1} and Q^{-1} are of full rank, $\operatorname{rank}(C) = \operatorname{rank}(I_{n-r}) = n - \operatorname{rank}(A)$. We now develop an upper bound on $\operatorname{rank}(B)$,

$$\begin{aligned} \operatorname{rank}(B) &= \operatorname{rank}(P^{-1}Q^{-1}B) \\ &= \operatorname{rank}(AB + CB) \\ &\leq \operatorname{rank}(AB) + \operatorname{rank}(CB), \text{ by equation (3.129)} \\ &\leq \operatorname{rank}(AB) + \operatorname{rank}(C), \text{ by equation (3.128)} \\ &= \operatorname{rank}(AB) + n - \operatorname{rank}(A), \end{aligned}$$

yielding (3.170), a lower bound on $\operatorname{rank}(AB)$.

The inequality (3.170) is called *Sylvester's law of nullity*. It provides a lower bound on $\operatorname{rank}(AB)$ to go with the upper bound of inequality (3.128), $\min(\operatorname{rank}(A), \operatorname{rank}(B))$. The bound in inequality (3.170) is also sharp, as we can see by exhibiting matrices A and B such that $\operatorname{rank}(AB) = \operatorname{rank}(A) + \operatorname{rank}(B) - n$, as you are asked to do in Exercise 3.12b.

3.3.12 Determinants of Inverses

From the relationship $\det(AB) = \det(A)\det(B)$ for square matrices mentioned earlier, it is easy to see that for nonsingular square A,

$$\det(A^{-1}) = (\det(A))^{-1}, \tag{3.171}$$

and so

- $\det(A) = 0$ if and only if A is singular.

(From the definition of the determinant in equation (3.24), we see that the determinant of any finite-dimensional matrix with finite elements is finite. We implicitly assume that the elements are finite.)

For an $n \times n$ matrix with $n \geq 2$ whose determinant is nonzero, from equation (3.34) we have

$$A^{-1} = \frac{1}{\det(A)}\text{adj}(A). \tag{3.172}$$

If $\det(A) = 1$, this obviously implies

$$A^{-1} = \text{adj}(A).$$

See Exercise 3.15 on page 179 for an interesting consequence of this.

3.3.13 Inverses of Products and Sums of Nonsingular Matrices

In linear regression analysis and other applications, we sometimes need inverses of various sums or products of matrices. In regression analysis, this may be because we wish to update regression estimates based on additional data or because we wish to delete some observations.

There is no simple relationship between the inverses of factors in a Hadamard product and the product matrix, but there are simple relationships between the inverses of factors in Cayley and Kronecker products and the product matrices.

3.3.13.1 Inverses of Cayley Products of Matrices

The inverse of the Cayley product of two nonsingular matrices of the same size is particularly easy to form. If A and B are square full rank matrices of the same size,

$$(AB)^{-1} = B^{-1}A^{-1}. \tag{3.173}$$

We can see this by multiplying $B^{-1}A^{-1}$ and (AB). This, of course, generalizes to

$$(A_1 \cdots A_n)^{-1} = A_n^{-1} \cdots A_1^{-1}$$

if $A_1, \cdots, A_n$ are all full rank and conformable.

3.3.13.2 Inverses of Kronecker Products of Matrices

If A and B are square full rank matrices, then

$$(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}. \tag{3.174}$$

We can see this by multiplying $A^{-1} \otimes B^{-1}$ and $A \otimes B$ using equation (3.101) on page 96.

3.3.13.3 Inverses of Sums of Matrices and Their Inverses

The inverse of the sum of two nonsingular matrices is somewhat more complicated. The first question of course is whether the sum is nonsingular. We can develop several useful relationships of inverses of sums and the sums and products of the individual matrices.

The simplest case to get started is $I + A$. Let A and $I + A$ be nonsingular. Then it is easy to derive $(I + A)^{-1}$ by use of $I = AA^{-1}$ and equation (3.173). We get

$$(I + A)^{-1} = A^{-1}(I + A^{-1})^{-1}. \tag{3.175}$$

If A and B are full rank matrices of the same size and such sums as $I + A$, $A + B$, and so on, are full rank, the following relationships are easy to show (and are easily proven in the order given, using equations (3.173) and (3.175); see Exercise 3.16):

$$A(I + A)^{-1} = (I + A^{-1})^{-1}, \tag{3.176}$$

$$(A + B)^{-1} = A^{-1} - A^{-1}(A^{-1} + B^{-1})^{-1}A^{-1}, \tag{3.177}$$

$$(A + BB^{\mathrm{T}})^{-1}B = A^{-1}B(I + B^{\mathrm{T}}A^{-1}B)^{-1}, \tag{3.178}$$

$$(A^{-1} + B^{-1})^{-1} = A(A + B)^{-1}B, \tag{3.179}$$

$$A - A(A + B)^{-1}A = B - B(A + B)^{-1}B, \tag{3.180}$$

$$A^{-1} + B^{-1} = A^{-1}(A + B)B^{-1}, \tag{3.181}$$

$$(I + AB)^{-1} = I - A(I + BA)^{-1}B, \tag{3.182}$$

$$(I + AB)^{-1}A = A(I + BA)^{-1}. \tag{3.183}$$

When A and/or B are not of full rank, the inverses may not exist, but in that case these equations may or may not hold for a generalized inverse, which we will discuss in Sect. 3.6.

Another simple general result, this time involving some non-square matrices, is that if A is a full-rank $n \times n$ matrix, B is a full-rank $m \times m$ matrix, C is any $n \times m$ matrix, and D is any $m \times n$ matrix such that $A + CBD$ is full rank, then

$$(A + CBD)^{-1} = A^{-1} - A^{-1}C(B^{-1} + DA^{-1}C)^{-1}DA^{-1}. \tag{3.184}$$

This can be derived from equation (3.176), which is a special case of it. We can verify this by multiplication (Exercise 3.17).

From this it also follows that if A is a full-rank $n \times n$ matrix and b and c are n-vectors such that $(A + bc^{\mathrm{T}})$ is full rank, then

$$(A + bc^{\mathrm{T}})^{-1} = A^{-1} - \frac{A^{-1}bc^{\mathrm{T}}A^{-1}}{1 + c^{\mathrm{T}}A^{-1}b}. \tag{3.185}$$

This fact has application in adding an observation to a least squares linear regression problem (page 418).

3.3.13.4 An Expansion of a Matrix Inverse

There is also an analogue to the expansion of the inverse of $(1-a)$ for a scalar a:

$$(1-a)^{-1} = 1 + a + a^2 + a^3 + \cdots, \quad \text{if } |a| < 1.$$

This expansion for the scalar a comes from a factorization of the binomial $1 - a^k$ and the fact that $a^k \to 0$ if $|a| < 1$.

To extend this to $(I + A)^{-1}$ for a matrix A, we need a similar condition on A^k as k increases without bound. In Sect. 3.9 on page 164, we will discuss conditions that ensure the convergence of A^k for a square matrix A. We will define a norm $\|A\|$ on A and show that if $\|A\| < 1$, then $A^k \to 0$. Then, analogous to the scalar series, using equation (3.53) on page 78 for a square matrix A, we have

$$(I - A)^{-1} = I + A + A^2 + A^3 + \cdots, \quad \text{if } \|A\| < 1. \tag{3.186}$$

We include this equation here because of its relation to equations (3.176) through (3.182). We will discuss it further on page 171, after we have introduced and discussed $\|A\|$ and other conditions that ensure convergence. This expression and the condition that determines it are very important in the analysis of time series and other stochastic processes.

Also, looking ahead, we have another expression similar to equations (3.176) through (3.182) and (3.186) for a special type of matrix. If $A^2 = A$, for any $a \neq -1$,

$$(I + aA)^{-1} = I - \frac{a}{a+1}A$$

(see page 354).

3.3.14 Inverses of Matrices with Special Forms

Matrices with various special patterns may have inverses with similar patterns.

- The inverse of a nonsingular symmetric matrix is symmetric.
- The inverse of a diagonal matrix with nonzero entries is a diagonal matrix consisting of the reciprocals of those elements.

- The inverse of a block diagonal matrix with nonsingular submatrices along the diagonal is a block diagonal matrix consisting of the inverses of the submatrices.
- The inverse of a nonsingular triangular matrix is a triangular matrix with the same pattern; furthermore, the diagonal elements in the inverse are the reciprocals of the diagonal elements in the original matrix.

Each of these statements can be easily proven by multiplication (using the fact that the inverse is unique). See also Exercise 3.19 (and the hint).

The inverses of other matrices with special patterns, such as banded matrices, may not have those patterns.

In Chap. 8, we discuss inverses of various other special matrices that arise in applications in statistics.

3.3.15 Determining the Rank of a Matrix

Although the equivalent canonical form (3.151) immediately gives the rank of a matrix, in practice the numerical determination of the rank of a matrix is not an easy task. The problem is that rank is a mapping $\mathbb{R}^{n\times m} \to \mathbb{Z}_+$, where $\mathbb{Z}_+$ represents the positive integers. Such a function is often difficult to compute because the domain is dense and the range is sparse. Small changes in the domain may result in large discontinuous changes in the function value. (In Hadamard's sense, the problem is *ill-posed*.) The common way that the rank of a matrix is evaluated is by use of the QR decomposition; see page 252.

It is not even always clear whether a matrix is nonsingular. Because of rounding on the computer, a matrix that is mathematically nonsingular may appear to be singular. We sometimes use the phrase "nearly singular" or "algorithmically singular" to describe such a matrix. In Sects. 6.1 and 11.4, we consider this kind of problem in more detail.

3.4 More on Partitioned Square Matrices: The Schur Complement

A square matrix A that can be partitioned as

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \tag{3.187}$$

where A_{11} is nonsingular, has interesting properties that depend on the matrix

$$Z = A_{22} - A_{21}A_{11}^{-1}A_{12}, \tag{3.188}$$

which is called the *Schur complement* of A_{11} in A.

We first observe from equation (3.149) that if equation (3.187) represents a full rank partitioning (that is, if the rank of A_{11} is the same as the rank of A), then

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{21}A_{11}^{-1}A_{12} \end{bmatrix}, \tag{3.189}$$

and $Z = 0$.

There are other useful properties of the Schur complement, which we mention below. There are also some interesting properties of certain important random matrices partitioned in this way. For example, suppose A_{22} is $k \times k$ and A is an $m \times m$ Wishart matrix with parameters n and Σ partitioned like A in equation (3.187). (This of course means A is symmetrical, and so $A_{12} = A_{21}^{\mathrm{T}}$.) Then Z has a Wishart distribution with parameters $n - m + k$ and $\Sigma_{22} - \Sigma_{21}\Sigma_{11}^{-1}\Sigma_{12}$, and is independent of A_{21} and A_{11}. (See Exercise 4.12 on page 224 for the probability density function for a Wishart distribution.)

3.4.1 Inverses of Partitioned Matrices

Suppose A is nonsingular and can be partitioned as above with both A_{11} and A_{22} nonsingular. It is easy to see (Exercise 3.20, page 180) that the inverse of A is given by

$$A^{-1} = \begin{bmatrix} A_{11}^{-1} + A_{11}^{-1}A_{12}Z^{-1}A_{21}A_{11}^{-1} & -A_{11}^{-1}A_{12}Z^{-1} \\ -Z^{-1}A_{21}A_{11}^{-1} & Z^{-1} \end{bmatrix}, \tag{3.190}$$

where Z is the Schur complement of A_{11}.

If

$$A = [X\ y]^{\mathrm{T}}[X\ y]$$

and is partitioned as in equation (3.55) on page 79 and X is of full column rank, then the Schur complement of $X^{\mathrm{T}}X$ in $[X\ y]^{\mathrm{T}}[X\ y]$ is

$$y^{\mathrm{T}}y - y^{\mathrm{T}}X(X^{\mathrm{T}}X)^{-1}X^{\mathrm{T}}y. \tag{3.191}$$

This particular partitioning is useful in linear regression analysis (see, for example, page 363), where this Schur complement is the residual sum of squares and the more general Wishart distribution mentioned above reduces to a chi-squared distribution. (Although the expression is useful, this is an instance of a principle that we will encounter repeatedly: *the form of a mathematical expression and the way the expression should be evaluated in actual practice may be quite different.*)

3.4.2 Determinants of Partitioned Matrices

If the square matrix A is partitioned as

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

and A_{11} is square and nonsingular, then

$$\det(A) = \det(A_{11}) \det\left(A_{22} - A_{21}A_{11}^{-1}A_{12}\right); \tag{3.192}$$

that is, the determinant is the product of the determinant of the principal submatrix and the determinant of its Schur complement.

This result is obtained by using equation (3.38) on page 71 and the factorization

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & 0 \\ A_{21} & A_{22} - A_{21}A_{11}^{-1}A_{12} \end{bmatrix} \begin{bmatrix} I & A_{11}^{-1}A_{12} \\ 0 & I \end{bmatrix}. \tag{3.193}$$

The factorization in equation (3.193) is often useful in other contexts as well.

3.5 Linear Systems of Equations

Some of the most important applications of matrices are in representing and solving systems of n linear equations in m unknowns,

$$Ax = b,$$

where A is an $n \times m$ matrix, x is an m-vector, and b is an n-vector. As we observed in equation (3.84), the product Ax in the linear system is a linear combination of the columns of A; that is, if a_j is the j^{th} column of A, $Ax = \sum_{j=1}^{m} x_j a_j$.

If $b = 0$, the system is said to be *homogeneous*. In this case, unless $x = 0$, the columns of A must be linearly dependent.

3.5.1 Solutions of Linear Systems

When in the linear system $Ax = b$, A is square and nonsingular, the solution is obviously $x = A^{-1}b$. We will not discuss this simple but common case further here. Rather, we will discuss it in detail in Chap. 6 after we have discussed matrix factorizations later in this chapter and in Chap. 5.

When A is not square or is singular, the system may not have a solution or may have more than one solution. A consistent system (see equation (3.137)) has a solution. For consistent systems that are singular or not square, the *generalized inverse* is an important concept. We introduce it in this section but defer its discussion to Sect. 3.6.

3.5.1.1 Underdetermined Systems

A consistent system in which $\text{rank}(A) < m$ is said to be *underdetermined*. An underdetermined system may have fewer equations than variables, or the coefficient matrix may just not be of full rank. For such a system there is more than one solution. In fact, there are infinitely many solutions because if

the vectors x_1 and x_2 are solutions, the vector $wx_1 + (1-w)x_2$ is likewise a solution for any scalar w.

Underdetermined systems arise in analysis of variance in statistics, and it is useful to have a compact method of representing the solution to the system. It is also desirable to identify a unique solution that has some kind of optimal properties. Below, we will discuss types of solutions and the number of linearly independent solutions and then describe a unique solution of a particular type.

3.5.1.2 Overdetermined Systems

Often in mathematical modeling applications, the number of equations in the system $Ax = b$ is not equal to the number of variables; that is the coefficient matrix A is $n \times m$ and $n \neq m$. If $n > m$ and $\text{rank}([A \mid b]) > \text{rank}(A)$, the system is said to be *overdetermined*. There is no x that satisfies such a system, but approximate solutions are useful. We discuss approximate solutions of such systems in Sect. 6.6 on page 289 and in Sect. 9.3.2 on page 408.

3.5.1.3 Generalized Inverses

A matrix G such that $AGA = A$ is called a *generalized inverse* and is denoted by A^-:

$$AA^-A = A. \tag{3.194}$$

Note that if A is $n \times m$, then A^- is $m \times n$. If A is nonsingular (square and of full rank), then obviously $A^- = A^{-1}$.

Without additional restrictions on A, the generalized inverse is not unique. Various types of generalized inverses can be defined by adding restrictions to the definition of the inverse. In Sect. 3.6, we will discuss various types of generalized inverses and show that A^- exists for any $n \times m$ matrix A. Here we will consider some properties of any generalized inverse.

From equation (3.194), we see that

$$A^{\mathrm{T}}(A^-)^{\mathrm{T}}A^{\mathrm{T}} = A^{\mathrm{T}};$$

thus, if A^- is a generalized inverse of A, then $(A^-)^{\mathrm{T}}$ is a generalized inverse of A^{T}.

The $m \times m$ square matrices A^-A and $(I - A^-A)$ are often of interest. By using the definition (3.194), we see that

$$(A^-A)(A^-A) = A^-A. \tag{3.195}$$

(Such a matrix is said to be *idempotent*. We discuss idempotent matrices beginning on page 352.) From equation (3.128) together with the fact that $AA^-A = A$, we see that

$$\text{rank}(A^-A) = \text{rank}(A). \tag{3.196}$$

By multiplication as above, we see that

$$A(I - A^{-}A) = 0, \tag{3.197}$$

that

$$(I - A^{-}A)(A^{-}A) = 0, \tag{3.198}$$

and that $(I - A^{-}A)$ is also idempotent:

$$(I - A^{-}A)(I - A^{-}A) = (I - A^{-}A). \tag{3.199}$$

The fact that $(A^{-}A)(A^{-}A) = A^{-}A$ yields the useful fact that

$$\text{rank}(I - A^{-}A) = m - \text{rank}(A). \tag{3.200}$$

This follows from equations (3.198), (3.170), and (3.196), which yield

$$0 \geq \text{rank}(I - A^{-}A) + \text{rank}(A) - m,$$

and from equation (3.129), which gives

$$m = \text{rank}(I) \leq \text{rank}(I - A^{-}A) + \text{rank}(A).$$

The two inequalities result in the equality of equation (3.200).

3.5.1.4 Multiple Solutions in Consistent Systems

Suppose the system $Ax = b$ is consistent and A^{-} is a generalized inverse of A; that is, it is any matrix such that $AA^{-}A = A$. Then

$$x = A^{-}b \tag{3.201}$$

is a solution to the system because if $AA^{-}A = A$, then $AA^{-}Ax = Ax$ and since $Ax = b$,

$$AA^{-}b = b; \tag{3.202}$$

that is, $A^{-}b$ is a solution.

Furthermore, if $x = Gb$ is any solution, then $AGA = A$; that is, G is a generalized inverse of A. This can be seen by the following argument. Let a_j be the j^{th} column of A. The m systems of n equations, $Ax = a_j$, $j = 1, \ldots, m$, all have solutions. (Each solution is a vector with 0s in all positions except the j^{th} position, which is a 1.) Now, if Gb is a solution to the original system, then Ga_j is a solution to the system $Ax = a_j$. So $AGa_j = a_j$ for all j; hence $AGA = A$.

If $Ax = b$ is consistent, not only is $A^{-}b$ a solution but also, for any z,

$$A^{-}b + (I - A^{-}A)z \tag{3.203}$$

is a solution because $A(A^{-}b + (I - A^{-}A)z) = AA^{-}b + (A - AA^{-}A)z = b$. Furthermore, any solution to $Ax = b$ can be represented as $A^{-}b + (I - A^{-}A)z$ for some z. This is because if y is any solution (that is, if $Ay = b$), we have

$$y = A^{-}b - A^{-}Ay + y = A^{-}b - (A^{-}A - I)y = A^{-}b + (I - A^{-}A)z.$$

The number of linearly independent solutions arising from $(I - A^{-}A)z$ is just the rank of $(I - A^{-}A)$, which from equation (3.200) is $m - \text{rank}(A)$.

3.5.2 Null Space: The Orthogonal Complement

The solutions of a consistent system $Ax = b$, which we characterized in equation (3.203) as $A^-b + (I - A^-A)z$ for any z, are formed as a given solution to $Ax = b$ plus all solutions to $Az = 0$.

For an $n \times m$ matrix A, the set of vectors generated by all solutions, z, of the homogeneous system

$$Az = 0 \tag{3.204}$$

is called the *null space* of A. We denote the null space of A by

$$\mathcal{N}(A).$$

The null space is either the single 0 vector (in which case we say the null space is empty or null) or it is a vector space. (It is actually a vector space in either case, but recall our ambiguity about the null vector space, page 13.)

We see that $\mathcal{N}(A)$ is a vector space (if it is not empty) because the zero vector is in $\mathcal{N}(A)$, and if x and y are in $\mathcal{N}(A)$ and a is any scalar, $ax + y$ is also a solution of $Az = 0$, and hence in $\mathcal{N}(A)$. We call the dimension of $\mathcal{N}(A)$ the *nullity* of A. The nullity of A is

$$\begin{aligned} \dim(\mathcal{N}(A)) &= \operatorname{rank}(I - A^-A) \\ &= m - \operatorname{rank}(A) \end{aligned} \tag{3.205}$$

from equation (3.200).

If $Ax = b$ is consistent, any solution can be represented as $A^-b + z$, for some z in the null space of A, because if y is some solution, $Ay = b = AA^-b$ from equation (3.202), and so $A(y - A^-b) = 0$; that is, $z = y - A^-b$ is in the null space of A. If A is nonsingular, then there is no such z, and the solution is unique. The number of linearly independent solutions to $Az = 0$, is the same as the nullity of A.

The order of $\mathcal{N}(A)$ is m. (Recall that the order of $\mathcal{V}(A)$ is n. The order of $\mathcal{V}(A^{\mathrm{T}})$ is m.)

If A is square, we have

$$\mathcal{N}(A) \subseteq \mathcal{N}(A^2) \subseteq \mathcal{N}(A^3) \subseteq \cdots \tag{3.206}$$

and

$$\mathcal{V}(A) \supseteq \mathcal{V}(A^2) \supseteq \mathcal{V}(A^3) \supseteq \cdots . \tag{3.207}$$

(We see this easily from the inequality (3.128) on page 103.)

If a is in $\mathcal{V}(A^{\mathrm{T}})$ and b is in $\mathcal{N}(A)$, we have $b^{\mathrm{T}}a = b^{\mathrm{T}}A^{\mathrm{T}}x = 0$. In other words, the null space of A is orthogonal to the row space of A; that is, $\mathcal{N}(A) \perp \mathcal{V}(A^{\mathrm{T}})$. This is because $A^{\mathrm{T}}x = a$ for some x, and $Ab = 0$ or $b^{\mathrm{T}}A^{\mathrm{T}} = 0$. For any matrix B whose columns are in $\mathcal{N}(A)$, $AB = 0$, and $B^{\mathrm{T}}A^{\mathrm{T}} = 0$.

Because $\dim(\mathcal{N}(A)) + \dim(\mathcal{V}(A^{\mathrm{T}})) = m$ and $\mathcal{N}(A) \perp \mathcal{V}(A^{\mathrm{T}})$, by equation (2.44) we have

$$\mathcal{N}(A) \oplus \mathcal{V}(A^{\mathrm{T}}) = \mathbb{R}^m; \tag{3.208}$$

that is, the null space of A is the *orthogonal complement* of $\mathcal{V}(A^{\mathrm{T}})$. All vectors in the null space of the matrix A^{T} are orthogonal to all vectors in the column space of A.

3.6 Generalized Inverses

On page 124, we defined a generalized inverse of a matrix A as a matrix A^- such that $AA^-A = A$, and we observed several interesting properties of generalized inverses. We will now consider some additional properties, after quickly summarizing some we observed previously.

3.6.1 Immediate Properties of Generalized Inverses

Let A be an $n \times m$ matrix, and let A^- be a generalized inverse of A. The properties of a generalized inverse A^- derived in equations (3.195) through (3.203) include:

- $(A^-)^{\mathrm{T}}$ is a generalized inverse of A^{T}.
- $\mathrm{rank}(A^-A) = \mathrm{rank}(A)$.
- A^-A is idempotent.
- $I - A^-A$ is idempotent.
- $\mathrm{rank}(I - A^-A) = m - \mathrm{rank}(A)$.

We note that if A is square (that is, $n = m$) and nonsingular, then $A^- = A^{-1}$, and so all of these properties apply to ordinary inverses.

In this section, we will first consider some special types of generalized inverses. Two of these special types of generalized inverses are unique. We will then discuss some more properties of "general" generalized inverses, which are analogous to properties of inverses. (We will call general generalized inverses "g_1 inverses".)

3.6.2 Special Generalized Inverses: The Moore-Penrose Inverse

A generalized inverse is not unique in general. As we have seen on page 125, a generalized inverse determines a set of linearly independent solutions to a linear system $Ax = b$. We may impose other conditions on the generalized inverse to arrive at a unique matrix that yields a solution that has some desirable properties. If we impose three more conditions, we have a unique matrix, denoted by A^+, that yields a solution A^+b that has the minimum length of any solution to $Ax = b$. We define this matrix and discuss some of its properties below, and in Sect. 6.6 we discuss properties of the solution A^+b.

3.6.2.1 Definitions and Terminology

To the general requirement $AA^-A = A$, we successively add three requirements that define special generalized inverses, sometimes called respectively g_2 or g_{12}, g_3 or g_{123}, and g_4 or g_{1234} inverses. The "general" generalized inverse is sometimes called a g_1 inverse. The g_4 inverse is called the Moore-Penrose inverse. As we will see below, it is unique. The terminology distinguishing the various types of generalized inverses is not used consistently in the literature. I will indicate some alternative terms in the definition below.

For a matrix A, a *Moore-Penrose inverse*, denoted by A^+, is a matrix that has the following four properties.

1. $AA^+A = A$. Any matrix that satisfies this condition is called a generalized inverse, and as we have seen above is denoted by A^-. For many applications, this is the only condition necessary. Such a matrix is also called a g_1 *inverse*, an *inner pseudoinverse*, or a *conditional inverse*.
2. $A^+AA^+ = A^+$. A matrix A^+ that satisfies this condition is called an *outer pseudoinverse*. A g_1 inverse that also satisfies this condition is called a g_2 *inverse* or *reflexive generalized inverse*, and is denoted by A^*.
3. A^+A is symmetric.
4. AA^+ is symmetric.

The Moore-Penrose inverse is also called the *pseudoinverse*, the *p-inverse*, and the *normalized generalized inverse*. (My current preferred term is "Moore-Penrose inverse", but out of habit, I often use the term "pseudoinverse" for this special generalized inverse. I generally avoid using any of the other alternative terms introduced above. I use the term "generalized inverse" to mean the "general generalized inverse", the g_1.) The name Moore-Penrose derives from the preliminary work of Moore (1920) and the more thorough later work of Penrose (1955), who laid out the conditions above and proved existence and uniqueness.

3.6.2.2 Existence

We can see by construction that the Moore-Penrose inverse exists for any matrix A. First, if $A = 0$, note that $A^+ = 0$. If $A \neq 0$, it has a full rank factorization, $A = LR$, as in equation (3.150), so $L^{\mathrm{T}}AR^{\mathrm{T}} = L^{\mathrm{T}}LRR^{\mathrm{T}}$. Because the $n \times r$ matrix L is of full column rank and the $r \times m$ matrix R is of full row rank, $L^{\mathrm{T}}L$ and RR^{T} are both of full rank, and hence $L^{\mathrm{T}}LRR^{\mathrm{T}}$ is of full rank. Furthermore, $L^{\mathrm{T}}AR^{\mathrm{T}} = L^{\mathrm{T}}LRR^{\mathrm{T}}$, so it is of full rank, and $(L^{\mathrm{T}}AR^{\mathrm{T}})^{-1}$ exists. Now, form $R^{\mathrm{T}}(L^{\mathrm{T}}AR^{\mathrm{T}})^{-1}L^{\mathrm{T}}$. By checking properties 1 through 4 above, we see that

$$A^+ = R^{\mathrm{T}}(L^{\mathrm{T}}AR^{\mathrm{T}})^{-1}L^{\mathrm{T}} \tag{3.209}$$

is a Moore-Penrose inverse of A. This expression for the Moore-Penrose inverse based on a full rank decomposition of A is not as useful as another expression we will consider later, based on QR decomposition (equation (5.45) on page 251).

3.6.2.3 Uniqueness

We can see that the Moore-Penrose inverse is unique by considering any matrix G that satisfies the properties 1 through 4 for $A \neq 0$. (The Moore-Penrose inverse of $A = 0$ (that is, $A^+ = 0$) is clearly unique, as there could be no other matrix satisfying property 2.) By applying the properties and using A^+ given above, we have the following sequence of equations:

$$\begin{gathered} G = \\ GAG = (GA)^{\mathrm{T}}G = A^{\mathrm{T}}G^{\mathrm{T}}G = (AA^+A)^{\mathrm{T}}G^{\mathrm{T}}G = (A^+A)^{\mathrm{T}}A^{\mathrm{T}}G^{\mathrm{T}}G = \\ A^+AA^{\mathrm{T}}G^{\mathrm{T}}G = A^+A(GA)^{\mathrm{T}}G = A^+AGAG = A^+AG = A^+AA^+AG = \\ A^+(AA^+)^{\mathrm{T}}(AG)^{\mathrm{T}} = A^+(A^+)^{\mathrm{T}}A^{\mathrm{T}}G^{\mathrm{T}}A^{\mathrm{T}} = A^+(A^+)^{\mathrm{T}}(AGA)^{\mathrm{T}} = \\ A^+(A^+)^{\mathrm{T}}A^{\mathrm{T}} = A^+(AA^+)^{\mathrm{T}} = A^+AA^+ \\ = A^+. \end{gathered}$$

3.6.2.4 Other Properties

Similarly to the property for inverses expressed in equation (3.145), we have

$$(A^+)^{\mathrm{T}} = (A^{\mathrm{T}})^+. \tag{3.210}$$

This is easily seen from the defining properties of the Moore-Penrose inverse.

If A is nonsingular, then obviously $A^+ = A^{-1}$, just as for any generalized inverse.

Because A^+ is a generalized inverse, all of the properties for a generalized inverse A^- discussed above hold; in particular, A^+b is a solution to the linear system $Ax = b$ (see equation (3.201)). In Sect. 6.6, we will show that this unique solution has a kind of optimality.

Moore-Penrose inverses also have a few additional interesting properties not shared by generalized inverses; for example

$$(I - A^+A)A^+ = 0. \tag{3.211}$$

3.6.2.5 Drazin Inverses

A *Drazin inverse* of a square matrix A is a matrix, which we will denote as A^{D}, such that

1. $A^{\mathrm{D}}AA^{\mathrm{D}} = A^{\mathrm{D}}$; that is, it is an outer pseudoinverse,
2. $AA^{\mathrm{D}} = A^{\mathrm{D}}A$, and
3. $A^{k+1}A^{\mathrm{D}} = A^k$ for any positive integer k.

Notice that these conditions together imply that a Drazin inverse is a g_1 inverse (that is, $AA^{\mathrm{D}}A = A$; see the conditions for the Moore-Penrose inverse on page 128). Because of this, a Drazin inverse satisfies most of the properties

listed for any generalized inverse on page 127; for example, $A^{\mathrm{D}}A$ is idempotent. The first property listed there is also satisfied; that is, $(A^{\mathrm{D}})^{\mathrm{T}}$ is the Drazin inverse of A^{T}. (This does not follow just because the Drazin inverse is a generalized inverse, however.)

Other important properties of Drazin inverses include

- $A^{\mathrm{D}} = A^{-1}$ if A is nonsingular.
- $(A^{\mathrm{D}})^{\mathrm{D}} = A$.
- For any square matrix, the Drazin inverse is unique.

These are left as exercises.

There is an interesting relationship between Drazin inverses and Moore-Penrose inverses. If A is any square matrix, for any positive integer k, its Drazin inverse is the matrix

$$A^{\mathrm{D}} = A^k(A^{2k+1})^+A^k. \tag{3.212}$$

Drazin inverses arise in the solutions of linear systems of differential equations. See Campbell and Meyer (1991) for further discussions of properties and applications of Drazin inverses and of their relationship to Moore-Penrose inverses.

3.6.3 Generalized Inverses of Products and Sums of Matrices

We often need to perform various operations on a matrix that is expressed as sums or products of various other matrices. Some operations are rather simple, for example, the transpose of the sum of two matrices is the sum of the transposes (equation (3.15)), and the transpose of the product is the product of the transposes in reverse order (equation (3.44)). Once we know the relationships for a single sum and a single product, we can extend those relationships to various sums and products of more than just two matrices.

In Sect. 3.3.13, beginning on page 118, we gave a number of relationships between inverses of sums and/or products and sums and/or products of sums. The two basic relationships were equations (3.173) and (3.175):

$$(AB)^{-1} = B^{-1}A^{-1}$$

and

$$(I + A)^{-1} = A^{-1}(I + A^{-1})^{-1}.$$

These same relations hold with the inverse replaced by generalized inverses.

We can relax the conditions on nonsingularity of A, B, $I + A$ and so on, but because of the nonuniqueness of generalized inverses, in some cases we must interpret the equations as "holding for some generalized inverse".

With the relaxation on the nonsingularity of constituent matrices, equations (3.176) through (3.182) do not necessarily hold for generalized inverses of general matrices, but some do. For example,

$$A(I+A)^- = (I+A^-)^-.$$

(Again, the true relationships are easily proven if taken in the order given on page 119, and in Exercise 3.18 you are asked to determine which are true for generalized inverses of general matrices and to prove that those are.)

3.6.4 Generalized Inverses of Partitioned Matrices

If A is partitioned as

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \tag{3.213}$$

then, similar to equation (3.190), a generalized inverse of A is given by

$$A^- = \begin{bmatrix} A_{11}^- + A_{11}^- A_{12} Z^- A_{21} A_{11}^- & -A_{11}^- A_{12} Z^- \\ -Z^- A_{21} A_{11}^- & Z^- \end{bmatrix}, \tag{3.214}$$

where $Z = A_{22} - A_{21}A_{11}^- A_{12}$ (see Exercise 3.23, page 180).

If the inverses on the right-hand side of equation (3.214) are Moore-Penrose inverses, then the result is the Moore-Penrose inverse of A.

If the partitioning in (3.213) happens to be such that A_{11} is of full rank and of the same rank as A, a generalized inverse of A is given by

$$A^- = \begin{bmatrix} A_{11}^{-1} & 0 \\ 0 & 0 \end{bmatrix}, \tag{3.215}$$

where 0 represents matrices of the appropriate shapes. The generalized inverse given in equation (3.215) is the same as the Moore-Penrose inverse given in equation (3.209), but it is not necessarily the same generalized inverse as in equation (3.214). The fact that it is a generalized inverse is easy to establish by using the definition of generalized inverse and equation (3.189).

3.7 Orthogonality

In Sect. 2.1.8, we defined orthogonality and orthonormality of two or more vectors in terms of dot products. On page 98, in equation (3.112), we also defined the orthogonal binary relationship between two matrices. Now we define the orthogonal unary property of a matrix. This is the more important property and is what is commonly meant when we speak of orthogonality of matrices. We use the orthonormality property of vectors, which is a binary relationship, to define orthogonality of a single matrix.

3.7.1 Orthogonal Matrices: Definition and Simple Properties

A matrix whose rows or columns constitute a set of orthonormal vectors is said to be an *orthogonal* matrix. If Q is an $n \times m$ orthogonal matrix, then $QQ^{\mathrm{T}} = I_n$ if $n \leq m$, and $Q^{\mathrm{T}}Q = I_m$ if $n \geq m$. If Q is a square orthogonal matrix, then $QQ^{\mathrm{T}} = Q^{\mathrm{T}}Q = I$.

The determinant of a square orthogonal matrix is ± 1 (because the determinant of the product is the product of the determinants and the determinant of I is 1).

When $n \geq m$, the matrix inner product of an $n \times m$ orthogonal matrix Q with itself is its number of columns:

$$\langle Q, Q \rangle = m. \tag{3.216}$$

This is because $Q^{\mathrm{T}}Q = I_m$. If $n \leq m$, the matrix inner product of Q with itself is its number of rows.

Recalling the definition of the orthogonal binary relationship from page 98, we note that if Q is an orthogonal matrix, then Q is not orthogonal to itself in that sense.

A permutation matrix (see page 81) is orthogonal. We can see this by building the permutation matrix as a product of elementary permutation matrices, and it is easy to see that they are all orthogonal.

One further property we see by simple multiplication is that if A and B are orthogonal, then $A \otimes B$ is orthogonal.

The definition of orthogonality is sometimes made more restrictive to require that the matrix be square.

An Aside: Unitary Matrices

For square matrices whose elements are complex numbers, a matrix is said to be *unitary* if the matrix times its conjugate transpose is the identity; that is, if $UU^{\mathrm{H}} = U^{\mathrm{H}}U = I$. Transformations using unitary matrices are analogous in many ways to transformations using orthogonal matrices, but there are important differences.

An orthogonal matrix with real elements is also a unitary matrix.

The definition of orthogonality of vectors is the same for complex vectors as it is for real vectors; in both cases, it is that the inner product is 0. Because of our emphasis on real vectors and matrices, we often think of orthogonality of vectors in terms of $x^{\mathrm{T}}y$, but this only applies to real vectors. In general, x and y are orthogonal if $x^{\mathrm{H}}y = 0$, which is the inner product. The corresponding binary relationship of orthogonality for matrices, as defined in equation (3.112) on page 98, likewise depends on an inner product, which is given in equation (3.107). The relationship in equation (3.108) may not be correct if the elements are not real.

For matrices, orthogonality is both a type of binary relationship and a unary property. The unary property of orthogonality is defined in terms of a transpose. A matrix that is orthogonal is also unitary only if it is real.

3.7.2 Orthogonal and Orthonormal Columns

The definition given above for orthogonal matrices is sometimes relaxed to require only that the columns or rows be orthogonal (rather than orthonormal). If orthonormality is not required, the determinant is not necessarily ± 1. If Q is a matrix that is "orthogonal" in this weaker sense of the definition, and Q has more rows than columns, then

$$Q^{\mathrm{T}}Q = \begin{bmatrix} \mathtt{X} & \mathtt{0} & \cdots & \mathtt{0} \\ \mathtt{0} & \mathtt{X} & \cdots & \mathtt{0} \\ & & \ddots & \\ \mathtt{0} & \mathtt{0} & \cdots & \mathtt{X} \end{bmatrix}.$$

Unless stated otherwise, I use the term "orthogonal matrix" to refer to a matrix whose columns are orthonormal; that is, for which $Q^{\mathrm{T}}Q = I$.

3.7.3 The Orthogonal Group

The set of $n \times m$ orthogonal matrices for which $n \geq m$ is called an (n, m) Stiefel manifold, and an (n, n) Stiefel manifold together with Cayley multiplication is a group, sometimes called the *orthogonal group* and denoted as $\mathcal{O}(n)$. The orthogonal group $\mathcal{O}(n)$ is a subgroup of the general linear group $\mathcal{G}L(n)$, defined on page 114. The orthogonal group is useful in multivariate analysis because of the invariance of the so-called Haar measure over this group (see Sect. 4.5.1).

Because the Euclidean norm of any column of an $n \times m$ orthogonal matrix with $n \geq m$ is 1, no element in the matrix can be greater than 1 in absolute value. We therefore have an analogue of the Bolzano-Weierstrass theorem for sequences of orthogonal matrices. The standard Bolzano-Weierstrass theorem for real numbers states that if a sequence a_i is bounded, then there exists a subsequence a_{i_j} that converges. (See any text on real analysis.) From this, we conclude that if $Q_1, Q_2, \ldots$ is a sequence of $n \times n$ orthogonal matrices, then there exists a subsequence $Q_{i_1}, Q_{i_2}, \ldots$, such that

$$\lim_{j \to \infty} Q_{i_j} = Q, \tag{3.217}$$

where Q is some fixed matrix. The limiting matrix Q must also be orthogonal because $Q_{i_j}^{\mathrm{T}} Q_{i_j} = I$, and so, taking limits, we have $Q^{\mathrm{T}}Q = I$. The set of $n \times n$ orthogonal matrices is therefore compact.

3.7.4 Conjugacy

Instead of defining orthogonality of vectors in terms of dot products as in Sect. 2.1.8, we could define it more generally in terms of a bilinear form as in Sect. 3.2.9. If the bilinear form $x^{\mathrm{T}}Ay = 0$, we say x and y are orthogonal with respect to the matrix A. We also often use a different term and say that the vectors are *conjugate* with respect to A, as in equation (3.93). The usual definition of orthogonality in terms of a dot product is equivalent to the definition in terms of a bilinear form in the identity matrix.

Likewise, but less often, orthogonality of matrices is generalized to conjugacy of two matrices with respect to a third matrix: $Q^{\mathrm{T}}AQ = I$.

3.8 Eigenanalysis: Canonical Factorizations

Throughout this section on eigenanalysis, we will generally implicitly assume that the matrices we discuss are square, unless we state otherwise.

Multiplication of a given vector by a square matrix may result in a scalar multiple of the vector. Stating this more formally, and giving names to such a special vector and scalar, if A is an $n \times n$ (square) matrix, v is a vector not equal to 0, and c is a scalar such that

$$Av = cv, \tag{3.218}$$

we say v is an *eigenvector* of the matrix A, and c is an *eigenvalue* of the matrix A. We refer to the pair c and v as an associated eigenvector and eigenvalue or as an *eigenpair*.

We immediately note that if v is an eigenvector of A, then for any scalar, b, because $A(bv) = c(bv)$, bv is also an eigenvector of A. (We will exclude the case $b = 0$, so that we do not consider the 0 vector to be an eigenvector of A.) That is, any vector in the double cone generated by an eigenvector, except the 0 vector, is an eigenvector (see discussion of cones, beginning on page 43).

While we restrict an eigenvector to be nonzero (or else we would have 0 as an eigenvector associated with any number being an eigenvalue), an eigenvalue can be 0; in that case, of course, the matrix must be singular. (Some authors restrict the definition of an eigenvalue to real values that satisfy (3.218), and there is an important class of matrices for which it is known that all eigenvalues are real. In this book, we do not want to restrict ourselves to that class; hence, we do not require c or v in equation (3.218) to be real.)

We use the term "eigenanalysis" or "eigenproblem" to refer to the general theory, applications, or computations related to either eigenvectors or eigenvalues.

There are various other terms used for eigenvalues and eigenvectors. An eigenvalue is also called a *characteristic value* (that is why I use a "c" to represent an eigenvalue), a *latent root* (that is why I also might use a "λ" to

represent an eigenvalue), or a *proper value*, and similar synonyms exist for an eigenvector. An eigenvalue is also sometimes called a *singular value*, but the latter term has a different meaning that we will use in this book (see page 161; the absolute value of an eigenvalue is a singular value, and singular values are also defined for nonsquare matrices).

Although generally throughout this chapter we have assumed that vectors and matrices are real, in eigenanalysis, even if A is real, it may be the case that c and v are complex. Therefore, in this section, we must be careful about the nature of the eigenpairs, even though we will continue to assume the basic matrices are real.

3.8.1 Eigenvalues and Eigenvectors Are Remarkable

Before proceeding to consider properties of eigenvalues and eigenvectors, we should note how remarkable the relationship $Av = cv$ is.

The effect of a matrix multiplication of an eigenvector is the same as a scalar multiplication of the eigenvector.

The eigenvector is an *invariant* of the transformation in the sense that its direction does not change. This would seem to indicate that the eigenvalue and eigenvector depend on some kind of deep properties of the matrix, and indeed this is the case, as we will see.

Of course, the first question is, for a given matrix, do such special vectors and scalars exist?

> The answer is yes.

The next question is, for a given matrix, what is the formula for the eigenvalues (or what is a finite sequence of steps to compute the eigenvalues)?

> The answer is a formula does not exist and there is no finite sequence of steps, in general, for determining the eigenvalues (if the matrix is bigger than 4×4).

Before considering these and other more complicated issues, we will state some simple properties of any scalar and vector that satisfy $Av = cv$ and introduce some additional terminology.

3.8.2 Left Eigenvectors

In the following, when we speak of an eigenvector or eigenpair without qualification, we will mean the objects defined by equation (3.218). There is another type of eigenvector for A, however, a *left eigenvector*, defined as a nonzero w in

$$w^{\mathrm{T}} A = c w^{\mathrm{T}}. \tag{3.219}$$

For emphasis, we sometimes refer to the eigenvector of equation (3.218), $Av = cv$, as a *right eigenvector*.

We see from the definition of a left eigenvector, that if a matrix is symmetric, each left eigenvector is an eigenvector (a *right* eigenvector).

If v is an eigenvector of A and w is a left eigenvector of A with a different associated eigenvalue, then v and w are orthogonal; that is, if $Av = c_1v$, $w^{\mathrm{T}}A = c_2w^{\mathrm{T}}$, and $c_1 \neq c_2$, then $w^{\mathrm{T}}v = 0$. We see this by multiplying both sides of $w^{\mathrm{T}}A = c_2w^{\mathrm{T}}$ by v to get $w^{\mathrm{T}}Av = c_2w^{\mathrm{T}}v$ and multiplying both sides of $Av = c_1v$ by w^{T} to get $w^{\mathrm{T}}Av = c_1w^{\mathrm{T}}v$. Hence, we have $c_1w^{\mathrm{T}}v = c_2w^{\mathrm{T}}v$, and because $c_1 \neq c_2$, we have $w^{\mathrm{T}}v = 0$.

3.8.3 Basic Properties of Eigenvalues and Eigenvectors

If c is an eigenvalue and v is a corresponding eigenvector for a real matrix A, we see immediately from the definition of eigenvector and eigenvalue in equation (3.218) the following properties. (In Exercise 3.25, you are asked to supply the simple proofs for these properties, or you can see the proofs in a text such as Harville 1997, for example.)

Assume that $Av = cv$ and that all elements of A are real.

1. bv is an eigenvector of A, where b is any nonzero scalar.
   It is often desirable to scale an eigenvector v so that $v^{\mathrm{T}}v = 1$. Such an eigenvector is also called a "unit eigenvector", but I prefer the term "normalized eigenvector" because of the use of the phrase "unit vector" to refer to the special vectors e_i.
   For a given eigenvector, there is always a particular eigenvalue associated with it, but for a given eigenvalue there is a space of associated eigenvectors. (The space is a vector space if we consider the zero vector to be a member.) It is therefore not appropriate to speak of "the" eigenvector associated with a given eigenvalue—although we do use this term occasionally. (We could interpret it as referring to the normalized eigenvector.) There is, however, another sense in which an eigenvalue does not determine a unique eigenvector, as we discuss below.
2. bc is an eigenvalue of bA, where b is any nonzero scalar.
3. $1/c$ and v are an eigenpair of A^{-1} (if A is nonsingular).
4. $1/c$ and v are an eigenpair of A^{+} if A (and hence A^{+}) is square and c is nonzero.
5. If A is diagonal or triangular with elements a_{ii}, the eigenvalues are a_{ii}, and for diagonal A the corresponding eigenvectors are e_i (the unit vectors).
6. c^2 and v are an eigenpair of A^2. More generally, c^k and v are an eigenpair of A^k for $k = 1, 2, \ldots$.
7. $c - d$ and v are an eigenpair of $A - dI$. This obvious fact is useful in computing eigenvalues (see Sect. 7.1.5).
8. If A and B are conformable for the multiplications AB and BA, the nonzero eigenvalues of AB are the same as the nonzero eigenvalues of BA. (Note that A and B are not necessarily square.) All of eigenvalues are the same if A and B are square. (Note, however, that if A and B are

square and d is an eigenvalue of B, d is not necessarily an eigenvalue of AB.)

9. If A and B are square and of the same order and if B^{-1} exists, then the eigenvalues of BAB^{-1} are the same as the eigenvalues of A. (This is called a similarity transformation; see page 146.)

List continued on page 140.

3.8.3.1 Eigenvalues of Elementary Operator Matrices

For a matrix with a very simple pattern, such as a disagonal matrix, whose determinant is just the product of the elements, we can determine the eigenvalues by inspection. For example, it is clear immediately that all eigenvalues of the identity matrix are 1s. (Although they are all the same, we still say there are n of them, if n is the order of the identity. Multiplicity of eigenvalues is an important property, which we will discuss beginning on page 143.)

Because of their simple patterns, we can also easily determine the eigenvalues of elementary operator matrices, possibly by considering one or two adjugates that arise from submatrices that are identity matrices.

The eigenvalues of the 2×2 permutation

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

are just the two square roots of 1; that is, 1 and -1. From this, using partitions of an elementary permutation matrix E_{pq} of order n to form adjugates that are identities, we see that the eigenvalues of an elementary permutation matrix E_{pq} are $n-1$ 1s and one -1.

With a little more effort we can determine the eigenvalues of general permutation matrices. Following the preceding approach, we immediately see that the eigenvalues of the matrix

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

are the three cube roots of 1, two of which contain imaginary components. In Chap. 8, on page 388, we describe the full set of eigenvalues for a permutation matrix in which all rows are moved.

By inspection of the determinant, we see that the eigenvalues of an elementary row-multiplication matrix $E_p(a)$ of order n are $n-1$ 1s and one a.

Again by inspection of the determinant, we see that the eigenvalues of an elementary axpy matrix $E_{pq}(a)$ of order n are n 1s, the same as the identity itself.

3.8.4 The Characteristic Polynomial

From the equation $(A - cI)v = 0$ that defines eigenvalues and eigenvectors, we see that in order for v to be nonnull, $(A - cI)$ must be singular, and hence

$$\det(A - cI) = \det(cI - A) = 0. \tag{3.220}$$

Equation (3.220) is sometimes taken as the definition of an eigenvalue c. It is definitely a fundamental relation, and, as we will see, allows us to identify a number of useful properties.

For the $n \times n$ matrix A, the determinant in equation (3.220) is a polynomial of degree n in c, $p_A(c)$, called the *characteristic polynomial*, and when it is equated to 0, it is called the *characteristic equation*:

$$p_A(c) = s_0 + s_1 c + \cdots + s_n c^n = 0. \tag{3.221}$$

From the expansion of the determinant $\det(cI - A)$, as in equation (3.41) on page 73, we see that $s_0 = (-1)^n \det(A)$ and $s_n = 1$, and, in general, $s_k = (-1)^{n-k}$ times the sums of all principal minors of A of order $n - k$. (Note that the signs of the s_i are different depending on whether we use $\det(cI - A)$ or $\det(A - cI)$.)

An eigenvalue of A is a root of the characteristic polynomial. The existence of n roots of the polynomial (by the Fundamental Theorem of Algebra) allows the characteristic polynomial to be written in factored form as

$$p_A(c) = (-1)^n (c - c_1) \cdots (c - c_n), \tag{3.222}$$

and establishes the existence of n eigenvalues. Some may be complex, some may be zero, and some may be equal to others. We call the set of all eigenvalues the *spectrum* of the matrix. The "number of eigenvalues" must be distinguished from the cardinality of the spectrum, which is the number of unique values.

A real matrix may have complex eigenvalues (and, hence, eigenvectors), just as a polynomial with real coefficients can have complex roots. Clearly, the eigenvalues of a real matrix must occur in conjugate pairs just as in the case of roots of polynomials with real coefficients. (As mentioned above, some authors restrict the definition of an eigenvalue to real values that satisfy (3.218). We will see below that the eigenvalues of a real symmetric matrix are always real, and this is a case that we will emphasize, but in this book we do not restrict the definition.)

The characteristic polynomial has many interesting properties. One, stated in the *Cayley-Hamilton theorem*, is that the matrix itself is a root of the matrix polynomial formed by the characteristic polynomial; that is,

$$p_A(A) = s_0 I + s_1 A + \cdots + s_n A^n = 0_n. \tag{3.223}$$

We see this by using equation (3.34) to write the matrix in equation (3.220) as

$$(A - cI)\text{adj}(A - cI) = p_A(c)I. \tag{3.224}$$

Hence $\text{adj}(A - cI)$ is a polynomial in c of degree less than or equal to $n - 1$, so we can write it as

$$\text{adj}(A - cI) = B_0 + B_1 c + \cdots + B_{n-1}c^{n-1},$$

where the B_i are $n \times n$ matrices. Now, equating the coefficients of c on the two sides of equation (3.224), we have

$$\begin{aligned} AB_0 &= s_0 I \\ AB_1 - B_0 &= s_1 I \\ &\vdots \\ AB_{n-1} - B_{n-2} &= s_{n-1} I \\ B_{n-1} &= s_n I. \end{aligned}$$

Now, multiply the second equation by A, the third equation by A^2, and the i^{th} equation by A^{i-1}, and add all equations. We get the desired result: $p_A(A) = 0$. See also Exercise 3.26.

Another interesting fact is that any given n^{th}-degree polynomial, p, is the characteristic polynomial of an $n \times n$ matrix, A, of particularly simple form. Consider the polynomial

$$p(c) = s_0 + s_1 c + \cdots + s_{n-1}c^{n-1} + c^n$$

and the matrix

$$A = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ & & & \ddots & \\ 0 & 0 & 0 & \cdots & 1 \\ -s_0 & -s_1 & -s_2 & \cdots & -s_{n-1} \end{bmatrix}. \tag{3.225}$$

(Note that this matrix is the same as the Jordan block (see page 78), except that the last row of 0s is replaced with the row of coefficients of the characteristic equation.) The matrix A is called the *companion matrix* of the polynomial p, and it is easy to see (by a tedious expansion) that the characteristic polynomial of A is p. This, of course, shows that a characteristic polynomial does not uniquely determine a matrix, although the converse is true (within signs).

3.8.4.1 Additional Properties of Eigenvalues and Eigenvectors

Using the characteristic polynomial yields the following properties. This is a continuation of the list we began on page 136. We assume A is a real matrix with eigenpair (c, v).

10. c is an eigenvalue of A^{T} (because $\det(A^{\mathrm{T}}-cI) = \det(A-cI)$ for any c). The eigenvectors of A^{T}, which are left eigenvectors of A, are not necessarily the same as the eigenvectors of A, however.
11. There is a left eigenvector such that c is the associated eigenvalue.
12. $(\bar{c}, \bar{v})$ is an eigenpair of A, where $\bar{c}$ and $\bar{v}$ are the complex conjugates and A, as usual, consists of real elements. (If c and v are real, this is a tautology.)
13. $c\bar{c}$ is an eigenvalue of the Gramian matrix $A^{\mathrm{T}}A$.
14. The nonzero eigenvalues of the Gramian matrix $A^{\mathrm{T}}A$ are the same as the nonzero eigenvalues of the Gramian matrix AA^{T}. (This is actually property 8 on page 136.)
15. c is real if A is symmetric or if A is triangular (the elements of A are assumed to be real, of course).

In Exercise 3.27, you are asked to supply the simple proofs for these properties, or you can see the proofs in a text such as Harville (1997), for example.

A further comment on property 12 may be worthwhile. Throughout this book, we assume we begin with real numbers. There are some times, however, when standard operations in the real domain carry us outside the reals. The simplest situations, which of course are related, are roots of polynomial equations with real coefficients and eigenpairs of matrices with real elements. In both of these situations, because sums must be real, the complex values occur in conjugate pairs.

There are many additional interesting properties of eigenvalues and eigenvectors that we will encounter in later sections, but there is one more that I want to list here with these very basic and important properties:

16. $|c| \leq \|A\|$, where $\|\cdot\|$ is any consistent matrix norm.

(We will discuss matrix norms in Sect. 3.9 beginning on page 164, and this particular bound is given in equation (3.310) in that section. In my definition of matrix norm, all norms are required to be consistent.)

3.8.4.2 Eigenvalues and the Trace and Determinant

If the eigenvalues of the matrix A are $c_1, \ldots, c_n$, because they are the roots of the characteristic polynomial, we can readily form that polynomial as

$$\begin{aligned} p_A(c) &= (c-c_1)\cdots(c-c_n) \\ &= (-1)^n \prod c_i + \cdots + (-1)^{n-1} \sum c_i c^{n-1} + c^n. \end{aligned} \tag{3.226}$$

Because this is the same polynomial as obtained by the expansion of the determinant in equation (3.221), the coefficients must be equal. In particular, by simply equating the corresponding coefficients of the constant terms and $(n-1)^{\text{th}}$-degree terms, we have the two very important facts:

$$\det(A) = \prod c_i \tag{3.227}$$

and

$$\mathrm{tr}(A) = \sum c_i. \tag{3.228}$$

It might be worth recalling that we have assumed that A is real, and therefore $\det(A)$ and $\mathrm{tr}(A)$ are real, but the eigenvalues c_i may not be real. Nonreal eigenvalues, however, occur in conjugate pairs (property 12 above); hence $\prod c_i$ and $\sum c_i$ are real even though the individual elements may not be.

3.8.5 The Spectrum

Although, for an $n \times n$ matrix, from the characteristic polynomial we have n roots, and hence n eigenvalues, some of these roots may be the same. It may also be the case that more than one eigenvector corresponds to a given eigenvalue. As we mentioned above, the set of all the distinct eigenvalues of a matrix is called the *spectrum* of the matrix.

3.8.5.1 Notation

Sometimes it is convenient to refer to the distinct eigenvalues and sometimes we wish to refer to all eigenvalues, as in referring to the number of roots of the characteristic polynomial. To refer to the distinct eigenvalues in a way that allows us to be consistent in the subscripts, we will call the distinct eigenvalues $\lambda_1, \ldots, \lambda_k$. The set of these constitutes the spectrum.

We denote the spectrum of the matrix A by $\sigma(A)$:

$$\sigma(A) = \{\lambda_1, \ldots, \lambda_k\}. \tag{3.229}$$

We see immediately that $\sigma(A^{\mathrm{T}}) = \sigma(A)$ (property 10 above).

In terms of the spectrum, equation (3.222) becomes

$$p_A(c) = (-1)^n (c - \lambda_1)^{m_1} \cdots (c - \lambda_k)^{m_k}, \tag{3.230}$$

for $m_i \geq 1$.

We label the c_i and v_i so that

$$|c_1| \geq \cdots \geq |c_n|. \tag{3.231}$$

We likewise label the λ_i so that

$$|\lambda_1| > \cdots > |\lambda_k|. \tag{3.232}$$

With this notation, we have

$$|\lambda_1| = |c_1|$$

and

$$|\lambda_k| = |c_n|,$$

but we cannot say anything about the other λs and cs.

3.8.5.2 The Spectral Radius

For the matrix A with these eigenvalues, $|c_1|$ is called the *spectral radius* and is denoted by $\rho(A)$:

$$\rho(A) = \max |c_i| = |c_1| = |\lambda_1|. \tag{3.233}$$

We immediately note that $\rho(A^{\mathrm{T}}) = \rho(A)$.

The set of complex numbers

$$\{z : |z| = \rho(A)\} \tag{3.234}$$

is called the *spectral circle* of A.

An eigenvalue equal to $\pm \max |c_i|$ (that is, equal to $\pm c_1$) is called a *dominant eigenvalue*. We are more often interested in the absolute value (or modulus) of a dominant eigenvalue rather than the eigenvalue itself; that is, $\rho(A)$ (or $|c_1|$) is more often of interest than c_1.

Interestingly, we have for all i

$$|c_i| \leq \max_j \sum_k |a_{kj}| \tag{3.235}$$

and

$$|c_i| \leq \max_k \sum_j |a_{kj}|. \tag{3.236}$$

The inequalities of course also hold for $\rho(A)$ on the left-hand side. Rather than proving this here, we show this fact in a more general setting relating to matrix norms in inequality (3.310) on page 171. (The two bounds above relate to the L_1 and L_∞ matrix norms, respectively, as we will see.)

The spectral radius gives a very simple indication of the region in the complex plane in which the entire spectrum lies. Consider, for example, the matrix

$$A = \begin{bmatrix} 9 & -6 & 1 \\ -2 & 9 & -5 \\ 10 & -2 & 4 \end{bmatrix}. \tag{3.237}$$

(See Exercise 3.24 for comments on the origins of this matrix.)

From equation (3.235), we see that all eigenvalues are less than or equal to 16 in modulus. In fact, the eigenvalues are $\sigma(A) = \{16, 3 + 4\mathrm{i}, 3 - 4\mathrm{i}\}$, and $\rho(A) = 16$.

On page 145, we will discuss other regions of the complex plane in which all eigenvalues necessarily lie.

A matrix may have all eigenvalues equal to 0 but yet the matrix itself may not be 0. (The matrix must be singular, however.) A nilpotent matrix (see page 77), as well as any upper triangular matrix with all 0s on the diagonal are examples.

Because, as we saw on page 136, if c is an eigenvalue of A, then bc is an eigenvalue of bA where b is any nonzero scalar, we can scale a matrix with a nonzero eigenvalue so that its spectral radius is 1. The scaled matrix is simply $S = A/|c_1|$, and $\rho(S) = 1$.

The spectral radius, is one of the most important properties of a matrix. As we will see in Sect. 3.9.1, it is the the L_p norm for a symmetric matrix. From equations (3.235) and (3.236), we have seen in any event that it is bounded from above by the L_1 and L_∞ matrix norms (which we will define formally in Sect. 3.9.1), and, in fact, in equation (3.310) we will see that the spectral radius is bounded from above by any matrix norm. We will discuss the spectral radius further in Sects. 3.9.5 and 3.9.6. In Sect. 3.9.6 we will see that the spectral radius determines the convergence of a matrix power series (and this fact is related to the behavior of autoregressive processes).

3.8.5.3 Linear Independence of Eigenvectors Associated with Distinct Eigenvalues

Suppose that $\{\lambda_1, \ldots, \lambda_k\}$ is a set of distinct eigenvalues of the matrix A and $\{x_1, \ldots, x_k\}$ is a set of eigenvectors such that (λ_i, x_i) is an eigenpair. Then $x_1, \ldots, x_k$ are linearly independent; that is, eigenvectors associated with distinct eigenvalues are linearly independent.

We can see that this must be the case by assuming that the eigenvectors are not linearly independent. In that case, let $\{y_1, \ldots, y_j\} \subset \{x_1, \ldots, x_k\}$, for some $j < k$, be a maximal linearly independent subset. Let the corresponding eigenvalues be $\{\mu_1, \ldots, \mu_j\} \subset \{\lambda_1, \ldots, \lambda_k\}$. Then, for some eigenvector y_{j+1}, we have

$$y_{j+1} = \sum_{i=1}^{j} t_i y_i$$

for some t_i. Now, multiplying both sides of the equation by $A - \mu_{j+1} I$, where μ_{j+1} is the eigenvalue corresponding to y_{j+1}, we have

$$0 = \sum_{i=1}^{j} t_i (\mu_i - \mu_{j+1}) y_i.$$

If the eigenvalues are distinct (that is, for each $i \leq j$), we have $\mu_i \neq \mu_{j+1}$, then the assumption that the eigenvalues are not linearly independent is contradicted because otherwise we would have a linear combination with nonzero coefficients equal to zero.

3.8.5.4 The Eigenspace and Geometric Multiplicity

Rewriting the definition (3.218) for the i^{th} eigenvalue and associated eigenvector of the $n \times n$ matrix A as

$$(A - c_i I)v_i = 0, \tag{3.238}$$

we see that the eigenvector v_i is in $\mathcal{N}(A - c_i I)$, the null space of $(A - c_i I)$. For such a nonnull vector to exist, of course, $(A - c_i I)$ must be singular; that is, $\text{rank}(A - c_i I)$ must be less than n. This null space is called the *eigenspace* of the eigenvalue c_i.

It is possible that a given eigenvalue may have more than one associated eigenvector that are linearly independent of each other. For example, we easily see that the identity matrix has only one distinct eigenvalue, namely 1, but any vector is an eigenvector, and so the number of linearly independent eigenvectors is equal to the number of rows or columns of the identity. If u and v are eigenvectors corresponding to the same eigenvalue λ, then any linear combination of u and v is an eigenvector corresponding to λ; that is, if $Au = \lambda u$ and $Av = \lambda v$, for any scalars a and b,

$$A(au + bv) = \lambda(au + bv).$$

The dimension of the eigenspace corresponding to the eigenvalue c_i is called the *geometric multiplicity* of c_i; that is, the geometric multiplicity of c_i is the nullity of $A - c_i I$. If g_i is the geometric multiplicity of c_i, an eigenvalue of the $n \times n$ matrix A, then we can see from equation (3.205) that $\text{rank}(A - c_i I) + g_i = n$.

The multiplicity of 0 as an eigenvalue is just the nullity of A. If A is of full rank, the multiplicity of 0 will be 0, but, in this case, we do not consider 0 to be an eigenvalue. If A is singular, however, we consider 0 to be an eigenvalue, and the multiplicity of the 0 eigenvalue is the rank deficiency of A.

Multiple linearly independent eigenvectors corresponding to the same eigenvalue can be chosen to be orthogonal to each other using, for example, the Gram-Schmidt transformations, as in equation (2.56) on page 38. These orthogonal eigenvectors span the same eigenspace. They are not unique, of course, as any sequence of Gram-Schmidt transformations could be applied.

3.8.5.5 Algebraic Multiplicity

A single value that occurs as a root of the characteristic equation m times is said to have *algebraic multiplicity* m. Although we sometimes refer to this as just the multiplicity, algebraic multiplicity should be distinguished from geometric multiplicity, defined above. These are not the same, as we will see in an example later (page 150). The algebraic multiplicity of a given eigenvalue is at least as great as its geometric multiplicity (exercise).

An eigenvalue whose algebraic multiplicity and geometric multiplicity are the same is called a *semisimple* eigenvalue. An eigenvalue with algebraic multiplicity 1 is called a *simple* eigenvalue (hence, a simple eigenvalue semisimple eigenvalue).

Because the determinant that defines the eigenvalues of an $n \times n$ matrix is an n^{th}-degree polynomial, we see that the sum of the multiplicities of distinct eigenvalues is n.

3.8.5.6 Gershgorin Disks

In addition to the spectral circle, there is a another specification of regions in the complex plane that regions in the complex plane that contain the spectrum of an $n \times n$ matrix A. This is the union of the n *Gershgorin disks*, where for $i = 1, \ldots, n$, the i^{th} of which is the disk

$$|z - a_{ii}| \leq r_i \quad \text{where } r_i = \sum_{1 \leq j \leq n;\ j \neq i} |a_{ij}|. \tag{3.239}$$

("Gershgorin" is often spelled as "Gerschgorin" or "Gersgorin" or even "Geršgorin"; he was Russian.)

To see that this is the case, let (c, v) be an arbitrary eigenpair of A with v normalized by the L_∞ norm (that is, $\max(v) = 1$). Let k be such that $|v_k| = 1$. Then

$$cv_k = (Av)_k = \sum_{j=1}^{n} a_{kj} v_j;$$

hence

$$(c - a_{kk})v_k = \sum_{1 \leq j \leq n;\ j \neq k} a_{kj} v_j.$$

Now introduce the modulus, and we get the desired inequality:

$$\begin{aligned} |c - a_{kk}| &= |c - a_{kk}||v_k| \\ &= \left| \sum_{1 \leq j \leq n;\ j \neq k} a_{kj} v_j \right| \\ &\leq \sum_{1 \leq j \leq n;\ j \neq k} |a_{kj}||v_j| \\ &\leq \sum_{1 \leq j \leq n;\ j \neq k} |a_{kj}| \\ &= r_k. \end{aligned}$$

We conclude that every eigenvalue lies in some similar disk; that is, the spectrum lies in the union of such disks.

Since $\sigma(A^{\mathrm{T}}) = \sigma(A)$, using the same argument as above, we can define another collection of n Gershgorin disks based on column sums:

$$|z - a_{jj}| \leq s_j \quad \text{where } s_j = \sum_{1 \leq i \leq n;\ i \neq j} |a_{ij}|. \tag{3.240}$$

All eigenvalues of A lie within the union of these disks.

Combining the two restrictions, we see that all eigenvalues of A lie within the intersection of these two unions of Gershgorin disks.

3.8.6 Similarity Transformations

Two $n \times n$ matrices, A and B, are said to be *similar* if there exists a nonsingular matrix P such that

$$B = P^{-1}AP. \tag{3.241}$$

The transformation in equation (3.241) is called a *similarity transformation*. (Compare similar matrices with *equivalent matrices* on page 110. The matrices A and B in equation (3.241) are also equivalent, as we see using equations (3.153) and (3.154).)

It is clear from the definition that the similarity relationship is both commutative and transitive.

If A and B are similar, as in equation (3.241), then for any scalar c

$$\begin{aligned} \det(A - cI) &= \det(P^{-1})\det(A - cI)\det(P) \\ &= \det(P^{-1}AP - cP^{-1}IP) \\ &= \det(B - cI), \end{aligned}$$

and, hence, A and B have the same eigenvalues. (This simple fact was stated as property 9 on page 137.)

3.8.6.1 Orthogonally and Unitarily Similar Transformations

An important type of similarity transformation is based on an orthogonal matrix in equation (3.241). If Q is orthogonal and

$$B = Q^{\mathrm{T}}AQ, \tag{3.242}$$

A and B are said to be *orthogonally similar*.

If B in equation (3.242) $B = Q^{\mathrm{T}}AQ$ is a diagonal matrix, A is said to be *orthogonally diagonalizable*, and QBQ^{T} is called the *orthogonally diagonal factorization* or *orthogonally similar factorization* of A.

The concepts of orthogonally similar and orthogonal diagonalization are very important, but for matrices with complex entries or for real matrices with complex eigenvalues, generalizations of the concepts based on unitary matrices are more useful. If U is unitary and

$$B = U^{\mathrm{H}}AU, \tag{3.243}$$

A and B are said to be *unitarily similar*. Since an orthogonal matrix is unitary, two matrices that are orthogonally similar are also unitarily similar.

If B in equation (3.243) $B = U^{\mathrm{H}}AU$ is a diagonal matrix, A is said to be *unitarily diagonalizable*, and QBQ^{T} is called the *unitarily diagonal factorization* or *unitarily similar factorization* of A. A matrix that is orthogonally diagonalizable is also unitarily diagonalizable.

We will discuss characteristics of orthogonally diagonalizable matrices in Sects. 3.8.8 through 3.8.10 below. The significant fact that we will see there is that a matrix is orthogonally diagonalizable if and only if it is symmetric.

We will discuss characteristics of unitarily diagonalizable matrices in Sect. 8.2.3 on page 345. The significant fact that we will see there is that a matrix is unitarily diagonalizable if and only if it is normal (which includes symmetric matrices).

3.8.6.2 Uses of Similarity Transformations

Similarity transformations are very useful in establishing properties of matrices, such as convergence properties of sequences (see, for example, Sect. 3.9.6). Similarity transformations are also used in algorithms for computing eigenvalues (see, for example, Sect. 7.3). In an orthogonally similar factorization, the elements of the diagonal matrix are the eigenvalues. Although the diagonals in the upper triangular matrix of the Schur factorization are the eigenvalues, that particular factorization is rarely used in computations.

Although similar matrices have the same eigenvalues, they do not necessarily have the same eigenvectors. If A and B are similar, for some nonzero vector v and some scalar c, $Av = cv$ implies that there exists a nonzero vector u such that $Bu = cu$, but it does not imply that $u = v$ (see Exercise 3.29b).

3.8.7 Schur Factorization

If B in equation (3.242) is an upper triangular matrix, QBQ^{T} is called the *Schur factorization* of A:

$$A = QBQ^{\mathrm{T}}. \tag{3.244}$$

This is also called the "Schur form" of A.

For any square matrix, the Schur factorization exists. Although the matrices in the factorization are not unique, the diagonal elements of the upper triangular matrix B are the eigenvalues of A.

There are various forms of the Schur factorization. Because in general the eigenvalues and eigenvectors may contain imaginary components, the orthogonal matrices in equation (3.244) may contain imaginary components, and furthermore, the Schur factorization is particularly useful in studying factorizations involving unitary matrices, we will describe the Schur factorization that use unitary matrices.

The general form of the Schur factorization for a square matrix A is

$$A = UTU^{\mathrm{H}}, \tag{3.245}$$

where U is a unitary matrix and T is an upper triangular matrix whose diagonal entries are the eigenvalues of A.

The Schur factorization exists for any square matrix, which we can see by induction. It clearly exists in the degenerate case of a 1×1 matrix. To see that

it exists for any $n \times n$ matrix A, let (c, v) be an arbitrary eigenpair of A with v normalized, and form a unitary matrix U_1 with v as its first column. Let U_2 be the matrix consisting of the remaining columns; that is, U_1 is partitioned as $[v \,|\, U_2]$.

$$\begin{aligned} U_1^{\mathrm{H}} A U_1 &= \begin{bmatrix} v^{\mathrm{H}} A v & v^{\mathrm{H}} A U_2 \\ U_2^{\mathrm{H}} A v & U_2^{\mathrm{H}} A U_2 \end{bmatrix} \\ &= \begin{bmatrix} c & v^{\mathrm{H}} A U_2 \\ 0 & U_2^{\mathrm{H}} A U_2 \end{bmatrix} \\ &= T, \end{aligned}$$

where $U_2^{\mathrm{H}} A U_2$ is an $(n-1) \times (n-1)$ matrix. Now the eigenvalues of $U^{\mathrm{H}} A U$ are the same as those of A; hence, if $n = 2$, then $U_2^{\mathrm{H}} A U_2$ is a scalar and must equal the other eigenvalue, and so the statement is proven for a 2×2 matrix.

We now use induction on n to establish the general case. Assume that the factorization exists for any $(n-1) \times (n-1)$ matrix, and let A be any $n \times n$ matrix. We let (c, v) be an arbitrary eigenpair of A (with v normalized), follow the same procedure as in the preceding paragraph, and get

$$U_1^{\mathrm{H}} A U_1 = \begin{bmatrix} c & v^{\mathrm{H}} A U_2 \\ 0 & U_2^{\mathrm{H}} A U_2 \end{bmatrix}.$$

Now, since $U_2^{\mathrm{H}} A U_2$ is an $(n-1) \times (n-1)$ matrix, by the induction hypothesis there exists an $(n-1) \times (n-1)$ Hermitian matrix V such that $V^{\mathrm{H}} (U_2^{\mathrm{H}} A U_2) V = T_1$, where T_1 is upper triangular. Now let

$$U = U_1 \begin{bmatrix} 1 & 0 \\ 0 & V \end{bmatrix}.$$

By multiplication, we see that $U^{\mathrm{H}} U = I$ (that is, Q is Hermitian). Now form

$$U^{\mathrm{H}} A U = \begin{bmatrix} c & v^{\mathrm{H}} A U_2 V \\ 0 & V^{\mathrm{H}} U_2^{\mathrm{H}} A U_2 V \end{bmatrix} = \begin{bmatrix} c & v^{\mathrm{H}} A U_2 V \\ 0 & T_1 \end{bmatrix} = T.$$

We see that T is upper triangular because T_1 is, and so by induction the Schur factorization exists for any $n \times n$ matrix.

The steps in the induction did not necessarily involve unique choices except for the eigenvalues on the diagonal of T.

Note that the Schur factorization is also based on unitarily similar transformations, but the term "unitarily similar factorization" is generally used only to refer to the diagonal factorization.

3.8.8 Similar Canonical Factorization: Diagonalizable Matrices

If V is a matrix whose columns correspond to the eigenvectors of A, and C is a diagonal matrix whose entries are the eigenvalues corresponding to the columns of V, using the definition (equation (3.218)) we can write

$$AV = VC. \tag{3.246}$$

Now, if V is nonsingular, we have

$$A = VCV^{-1}. \tag{3.247}$$

Expression (3.247) represents a *diagonal factorization* of the matrix A. We see that a matrix A with eigenvalues $c_1, \ldots, c_n$ that can be factorized this way is similar to the matrix $\mathrm{diag}((c_1, \ldots, c_n))$, and this representation is sometimes called the *similar canonical form* of A or the *similar canonical factorization* of A.

Not all matrices can be factored as in equation (3.247). It obviously depends on V being nonsingular; that is, that the eigenvectors are linearly independent. If a matrix can be factored as in (3.247), it is called a *diagonalizable matrix*, a *simple matrix*, or a *regular matrix* (the terms are synonymous, and we will generally use the term "diagonalizable"); a matrix that cannot be factored in that way is called a *deficient matrix* or a *defective matrix* (the terms are synonymous).

Any matrix all of whose eigenvalues are unique (that is, distinct) is diagonalizable (because, as we saw on page 143, in that case the eigenvectors are linearly independent), but uniqueness of the eigenvalues is not a necessary condition.

A necessary and sufficient condition for a matrix to be diagonalizable can be stated in terms of the unique eigenvalues and their multiplicities: suppose for the $n \times n$ matrix A that the distinct eigenvalues $\lambda_1, \ldots, \lambda_k$ have algebraic multiplicities $m_1, \ldots, m_k$. If, for $l = 1, \ldots, k$,

$$\mathrm{rank}(A - \lambda_l I) = n - m_l \tag{3.248}$$

(that is, if all eigenvalues are semisimple), then A is diagonalizable, and this condition is also necessary for A to be diagonalizable. This fact is called the "diagonalizability theorem". Recall that A being diagonalizable is equivalent to V in $AV = VC$ (equation (3.246)) being nonsingular.

To see that the condition is sufficient, assume, for each i, $\mathrm{rank}(A - c_i I) = n - m_i$, and so the equation $(A - c_i I)x = 0$ has exactly $n - (n - m_i)$ linearly independent solutions, which are by definition eigenvectors of A associated with c_i. (Note the somewhat complicated notation. Each c_i is the same as some λ_l, and for each λ_l, we have $\lambda_l = c_{l_1} = c_{l_{m_l}}$ for $1 \leq l_1 < \cdots < l_{m_l} \leq n$.) Let $w_1, \ldots, w_{m_i}$ be a set of linearly independent eigenvectors associated with c_i, and let u be an eigenvector associated with c_j and $c_j \neq c_i$. (The vectors $w_1, \ldots, w_{m_i}$ and u are columns of V.) We have already seen on page 143 that u must be linearly independent of the other eigenvectors, but we can also use a slightly different argument here. Now if u is not linearly independent of $w_1, \ldots, w_{m_i}$, we write $u = \sum b_k w_k$, and so $Au = A\sum b_k w_k = c_i \sum b_k w_k = c_i u$, contradicting the assumption that u is not an eigenvector associated with c_i. Therefore, the eigenvectors associated with different eigenvalues are linearly independent, and so V is nonsingular.

Now, to see that the condition is necessary, assume V is nonsingular; that is, V^{-1} exists. Because C is a diagonal matrix of all n eigenvalues, the matrix $(C - c_i I)$ has exactly m_i zeros on the diagonal, and hence, $\text{rank}(C - c_i I) = n - m_i$. Because $V(C - c_i I)V^{-1} = (A - c_i I)$, and multiplication by a full rank matrix does not change the rank (see page 113), we have $\text{rank}(A - c_i I) = n - m_i$.

3.8.8.1 Symmetric Matrices

A symmetric matrix is a diagonalizable matrix. We see this by first letting A be any $n \times n$ symmetric matrix with eigenvalue c of multiplicity m. We need to show that $\text{rank}(A - cI) = n - m$. Let $B = A - cI$, which is symmetric because A and I are. First, we note that c is real, and therefore B is real. Let $r = \text{rank}(B)$. From equation (3.164), we have

$$\text{rank}\left(B^2\right) = \text{rank}\left(B^{\mathrm{T}}B\right) = \text{rank}(B) = r.$$

In the full rank partitioning of B, there is at least one $r \times r$ principal submatrix of full rank. The r-order principal minor in B^2 corresponding to any full rank $r \times r$ principal submatrix of B is therefore positive. Furthermore, any j-order principal minor in B^2 for $j > r$ is zero. Now, rewriting the characteristic polynomial in equation (3.221) slightly by attaching the sign to the variable w, we have

$$p_{B^2}(w) = t_{n-r}(-w)^{n-r} + \cdots + t_{n-1}(-w)^{n-1} + (-w)^n = 0,$$

where t_{n-j} is the sum of all j-order principal minors. Because $t_{n-r} \neq 0$, $w = 0$ is a root of multiplicity $n - r$. It is likewise an eigenvalue of B with multiplicity $n - r$. Because $A = B + cI$, $0 + c$ is an eigenvalue of A with multiplicity $n - r$; hence, $m = n - r$. Therefore $n - m = r = \text{rank}(A - cI)$.

As we will see below in Sect. 3.8.10, a symmetric matrix A is not only diagonalizable in the form (3.247), $A = VCV^{-1}$, the matrix V can be chosen as an orthogonal matrix, so we have $A = UCU^{\mathrm{T}}$. We will say that the symmetric matrix is *orthogonally diagonalizable*.

3.8.8.2 A Defective Matrix

Although most matrices encountered in statistics applications are diagonalizable, it may be of interest to consider an example of a matrix that is not diagonalizable. Searle (1982) gives an example of a small matrix:

$$A = \begin{bmatrix} 0 & 1 & 2 \\ 2 & 3 & 0 \\ 0 & 4 & 5 \end{bmatrix}.$$

The three strategically placed 0s make this matrix easy to work with, and the determinant of $(cI - A)$ yields the characteristic polynomial equation

$$c^3 - 8c^2 + 13c - 6 = 0.$$

This can be factored as $(c-6)(c-1)^2$, hence, we have eigenvalues $c_1 = 6$ with algebraic multiplicity $m_1 = 1$, and $c_2 = 1$ with algebraic multiplicity $m_2 = 2$. Now, consider $A - c_2 I$:

$$A - I = \begin{bmatrix} -1 & 1 & 2 \\ 2 & 2 & 0 \\ 0 & 4 & 4 \end{bmatrix}.$$

This is clearly of rank 2; hence the rank of the null space of $A - c_2 I$ (that is, the geometric multiplicity of c_2) is $3 - 2 = 1$. The matrix A is not diagonalizable.

3.8.8.3 The Jordan Decomposition

Although not all matrices can be diagonalized in the form of equation (3.247), $V^{-1}AV = C = \text{diag}(c_i)$, any square matrix A can be expressed in the form

$$X^{-1}AX = \text{diag}(J_{j_i}), \tag{3.249}$$

where the J_{j_i} are *Jordan blocks* associated with a single eigenvalue λ_j, of the form

$$J_{j_i}(\lambda_j) = \begin{bmatrix} \lambda_j & 1 & 0 & \cdots & 0 & 0 \\ 0 & \lambda_j & 1 & \cdots & 0 & 0 \\ 0 & 0 & \lambda_j & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & \lambda_j & 1 \\ 0 & 0 & \cdots & 0 & 0 & \lambda_j \end{bmatrix},$$

or, in the degenerate case, $J_{j_i}(\lambda_j) = [\lambda_j]$, where λ_j is a specific distinct eigenvalue (that is, $\lambda_j \neq \lambda_k$ if $j \neq k$). (Compare this with the Jordan form of a nilpotent matrix following equation (3.51) on page 77, in which the diagonal elements are 0s.) If each Jordan block J_{j_i} is 1×1, the Jordan decomposition is a diagonal decomposition.

There are some interesting facts about the Jordan decomposition. If there are g_j Jordan blocks associated with the eigenvalue λ_j, then λ_j has geometric multiplicity g_j. The algebraic multiplicity of λ_j is the total number of diagonal elements in all the Jordan blocks associated with λ_j; hence, if each Jordan block J_{j_i} is 1×1 then all eigenvalues are semisimple. While these two facts appear rather profound, they are of little interest for our purposes, and we will not give proofs. (Proofs can be found in Horn and Johnson (1991).) The problem of computing a Jordan decomposition is ill-conditioned because slight perturbations in the elements of A can obviously result in completely different sets of Jordan blocks.

3.8.9 Properties of Diagonalizable Matrices

If the matrix A has the similar canonical factorization VCV^{-1} of equation (3.247), some important properties are immediately apparent. First of all, this factorization implies that the eigenvectors of a diagonalizable matrix are linearly independent.

Other properties are easy to derive or to show because of this factorization. For example, the general equations (3.227) and (3.228) concerning the product and the sum of eigenvalues follow easily from

$$\det(A) = \det(VCV^{-1}) = \det(V)\det(C)\det(V^{-1}) = \det(C)$$

and

$$\mathrm{tr}(A) = \mathrm{tr}(VCV^{-1}) = \mathrm{tr}(V^{-1}VC) = \mathrm{tr}(C).$$

One important fact is that the number of nonzero eigenvalues of a diagonalizable matrix A is equal to the rank of A. This must be the case because the rank of the diagonal matrix C is its number of nonzero elements and the rank of A must be the same as the rank of C. Another way of saying this is that the sum of the multiplicities of the unique nonzero eigenvalues is equal to the rank of the matrix; that is, $\sum_{i=1}^{k} m_i = \mathrm{rank}(A)$, for the matrix A with k distinct eigenvalues with multiplicities m_i.

3.8.9.1 Matrix Functions

We can use the diagonal factorization (3.247) of the matrix $A = VCV^{-1}$ to define a function of the matrix that corresponds to a scalar-valued function of a scalar, $f(x)$,

$$f(A) = V\mathrm{diag}((f(c_1), \ldots, f(c_n)))V^{-1}, \tag{3.250}$$

if $f(\cdot)$ is defined for each eigenvalue c_i. (Notice the relationship of this definition to the Cayley-Hamilton theorem, page 138, and to Exercise 3.26.)

Another useful feature of the diagonal factorization of the matrix A in equation (3.247) is that it allows us to study functions of powers of A because $A^k = VC^kV^{-1}$. In particular, we may assess the convergence of a function of a power of A,

$$\lim_{k\to\infty} g(k, A).$$

Functions defined by elementwise operations have limited applications. Functions of real numbers that have power series expansions may be defined for matrices in terms of power series expansions in A, which are effectively power series in the diagonal elements of C. For example, using the power series expansion of $\mathrm{e}^x = \sum_{k=0}^{\infty} \frac{x^k}{k!}$, we can define the *matrix exponential* for the square matrix A as the matrix

$$e^A = \sum_{k=0}^{\infty} \frac{A^k}{k!}, \tag{3.251}$$

where $A^0/0!$ is defined as I. (Recall that we did not define A^0 if A is singular.) If A is represented as VCV^{-1}, this expansion becomes

$$\begin{aligned} e^A &= V \sum_{k=0}^{\infty} \frac{C^k}{k!} V^{-1} \\ &= V \operatorname{diag}\left((e^{c_1}, \ldots, e^{c_n})\right) V^{-1}. \end{aligned}$$

This is called the *matrix exponential* for the square matrix A.

The expression $\exp(A)$ is generally interpreted as $\exp(A) = (\exp(a_{ij}))$, while the expression e^A is interpreted as in equation (3.251), but often each expression is used in the opposite way. As mentioned above, the standard `exp` function in software systems, when evaluated for a matrix A, yields $(\exp(a_{ij}))$. Both R and Matlab have a function `expm` for the matrix exponential. (In R, `expm` is in the `Matrix` package.)

3.8.10 Eigenanalysis of Symmetric Matrices

The eigenvalues and eigenvectors of symmetric matrices have some interesting properties. First of all, as we have already observed, for a real symmetric matrix, the eigenvalues are all real. We have also seen that symmetric matrices are diagonalizable; therefore all of the properties of diagonalizable matrices carry over to symmetric matrices.

3.8.10.1 Orthogonality of Eigenvectors: Orthogonal Diagonalization

In the case of a symmetric matrix A, any eigenvectors corresponding to distinct eigenvalues are orthogonal. This is easily seen by assuming that c_1 and c_2 are unequal eigenvalues with corresponding eigenvectors v_1 and v_2. Now consider $v_1^{\mathrm{T}} v_2$. Multiplying this by c_2, we get

$$c_2 v_1^{\mathrm{T}} v_2 = v_1^{\mathrm{T}} A v_2 = v_2^{\mathrm{T}} A v_1 = c_1 v_2^{\mathrm{T}} v_1 = c_1 v_1^{\mathrm{T}} v_2.$$

Because $c_1 \neq c_2$, we have $v_1^{\mathrm{T}} v_2 = 0$.

Now, consider two eigenvalues $c_i = c_j$, that is, an eigenvalue of multiplicity greater than 1 and distinct associated eigenvectors v_i and v_j. By what we just saw, an eigenvector associated with $c_k \neq c_i$ is orthogonal to the space spanned by v_i and v_j. Assume v_i is normalized and apply a Gram-Schmidt transformation to form

$$\tilde{v}_j = \frac{1}{\|v_j - \langle v_i, v_j \rangle v_i\|} \left(v_j - \langle v_i, v_j \rangle v_i\right),$$

as in equation (2.56) on page 38, yielding a vector orthogonal to v_i. Now, we have

$$\begin{aligned} A\tilde{v}_j &= \frac{1}{\|v_j - \langle v_i, v_j\rangle v_i\|}\,(Av_j - \langle v_i, v_j\rangle Av_i) \\ &= \frac{1}{\|v_j - \langle v_i, v_j\rangle v_i\|}\,(c_j v_j - \langle v_i, v_j\rangle c_i v_i) \\ &= c_j \frac{1}{\|v_j - \langle v_i, v_j\rangle v_i\|}\,(v_j - \langle v_i, v_j\rangle v_i) \\ &= c_j \tilde{v}_j; \end{aligned}$$

hence, $\tilde{v}_j$ is an eigenvector of A associated with c_j. We conclude therefore that the eigenvectors of a symmetric matrix can be chosen to be orthogonal.

A symmetric matrix is *orthogonally diagonalizable*, because the V in equation (3.247) can be chosen to be orthogonal, and can be written as

$$A = VCV^{\mathrm{T}}, \tag{3.252}$$

where $VV^{\mathrm{T}} = V^{\mathrm{T}}V = I$, and so we also have

$$V^{\mathrm{T}}AV = C. \tag{3.253}$$

Such a matrix is orthogonally similar to a diagonal matrix formed from its eigenvalues.

Not only is a symmetric matrix orthogonally diagonalizable, any matrix that is orthogonally diagonalizable is symmetric. This is easy to see. Suppose $B = VCV^{\mathrm{T}}$, where V is orthogonal and C is diagonal. Then

$$B^{\mathrm{T}} = (VCV^{\mathrm{T}})^{\mathrm{T}} = VCV^{\mathrm{T}} = B; \tag{3.254}$$

hence, B is symmetric.

3.8.10.2 Spectral Decomposition

When A is symmetric and the eigenvectors v_i are chosen to be orthonormal,

$$I = \sum_i v_i v_i^{\mathrm{T}}, \tag{3.255}$$

so

$$\begin{aligned} A &= A\sum_i v_i v_i^{\mathrm{T}} \\ &= \sum_i Av_i v_i^{\mathrm{T}} \\ &= \sum_i c_i v_i v_i^{\mathrm{T}}. \end{aligned} \tag{3.256}$$

This representation is called the *spectral decomposition* of the symmetric matrix A. It is essentially the same as equation (3.252), so $A = VCV^{\mathrm{T}}$ is also called the spectral decomposition.

The representation is unique except for the ordering and the choice of eigenvectors for eigenvalues with multiplicities greater than 1. If the rank of the matrix is r, we have $|c_1| \geq \cdots \geq |c_r| > 0$, and if $r < n$, then $c_{r+1} = \cdots = c_n = 0$.

Note that the matrices in the spectral decomposition are projection matrices that are orthogonal to each other (but they are not orthogonal matrices) and they sum to the identity. Let

$$P_i = v_i v_i^{\mathrm{T}}. \tag{3.257}$$

Then we have

$$P_i P_i = P_i, \tag{3.258}$$

$$P_i P_j = 0 \text{ for } i \neq j, \tag{3.259}$$

$$\sum_i P_i = I, \tag{3.260}$$

and the spectral decomposition,

$$A = \sum_i c_i P_i. \tag{3.261}$$

The P_i are called *spectral projectors*.

The spectral decomposition also applies to powers of A,

$$A^k = \sum_i c_i^k v_i v_i^{\mathrm{T}}, \tag{3.262}$$

where k is an integer. If A is nonsingular, k can be negative in the expression above.

The spectral decomposition is one of the most important tools in working with symmetric matrices.

Although we will not prove it here, all diagonalizable matrices have a spectral decomposition in the form of equation (3.261) with projection matrices that satisfy properties (3.258) through (3.260). These projection matrices cannot necessarily be expressed as outer products of eigenvectors, however. The eigenvalues and eigenvectors of a nonsymmetric matrix might not be real, the left and right eigenvectors might not be the same, and two eigenvectors might not be mutually orthogonal. In the spectral representation $A = \sum_i c_i P_i$, however, if c_j is a simple eigenvalue with associated left and right eigenvectors y_j and x_j, respectively, then the projection matrix P_j is $x_j y_j^{\mathrm{H}} / y_j^{\mathrm{H}} x_j$. (Note that because the eigenvectors may not be real, we take the conjugate transpose.) This is Exercise 3.30.

3.8.10.3 Kronecker Products of Symmetric Matrices: Orthogonal Diagonalization

If A and B are symmetric, then $A \otimes B$ is symmetric. (We have already mentioned this fact, but it is easy to see using equation (3.100) on page 96.)

Now if A and B are symmetric, we can orthogonally diagonalize them as in equation (3.252): $A = VCV^{\mathrm{T}}$ and $B = UDU^{\mathrm{T}}$. This immediately yields an orthogonal diagonalization of the symmetric matrix $A \otimes B$:

$$\begin{aligned} A \otimes B &= VCV^{\mathrm{T}} \otimes UDU^{\mathrm{T}} \\ &= (V \otimes U)(C \otimes D)(V^{\mathrm{T}} \otimes U^{\mathrm{T}}), \end{aligned} \tag{3.263}$$

which we obtain by using equation (3.101) twice. Using equation (3.101) again, we have $(V \otimes U)(V^{\mathrm{T}} \otimes U^{\mathrm{T}}) = (VV^{\mathrm{T}} \otimes UU^{\mathrm{T}}) = I$ and since $C \otimes D$ is obviously diagonal, equation (3.263) is in the orthogonally diagonalized form of equation (3.252).

3.8.10.4 Quadratic Forms and the Rayleigh Quotient

Equation (3.256) yields important facts about quadratic forms in A. Because V is of full rank, an arbitrary vector x can be written as Vb for some vector b. Therefore, for the quadratic form $x^{\mathrm{T}}Ax$ we have

$$\begin{aligned} x^{\mathrm{T}}Ax &= x^{\mathrm{T}} \sum_i c_i v_i v_i^{\mathrm{T}} x \\ &= \sum_i b^{\mathrm{T}} V^{\mathrm{T}} v_i v_i^{\mathrm{T}} V b c_i \\ &= \sum_i b_i^2 c_i. \end{aligned}$$

This immediately gives the inequality

$$x^{\mathrm{T}}Ax \leq \max\{c_i\} b^{\mathrm{T}} b.$$

(Notice that $\max\{c_i\}$ here is not necessarily c_1; in the important case when all of the eigenvalues are nonnegative, it is, however.) Furthermore, if $x \neq 0$, $b^{\mathrm{T}}b = x^{\mathrm{T}}x$, and we have the important inequality

$$\frac{x^{\mathrm{T}}Ax}{x^{\mathrm{T}}x} \leq \max\{c_i\}. \tag{3.264}$$

Equality is achieved if x is the eigenvector corresponding to $\max\{c_i\}$, so we have

$$\max_{x \neq 0} \frac{x^{\mathrm{T}}Ax}{x^{\mathrm{T}}x} = \max\{c_i\}. \tag{3.265}$$

If $c_1 > 0$, this is the spectral radius, $\rho(A)$.

The expression on the left-hand side in (3.264) as a function of x is called the *Rayleigh quotient* of the symmetric matrix A and is denoted by $R_A(x)$:

$$\begin{aligned} R_A(x) &= \frac{x^{\mathrm{T}}Ax}{x^{\mathrm{T}}x} \\ &= \frac{\langle x,\, Ax\rangle}{\langle x,\, x\rangle}. \end{aligned} \tag{3.266}$$

Because if $x \neq 0$, $x^{\mathrm{T}}x > 0$, it is clear that the Rayleigh quotient is nonnegative for all x if and only if A is nonnegative definite, and it is positive for all x if and only if A is positive definite.

3.8.10.5 The Fourier Expansion

The $v_i v_i^{\mathrm{T}}$ matrices in equation (3.256) have the property that $\langle v_i v_i^{\mathrm{T}}, v_j v_j^{\mathrm{T}}\rangle = 0$ for $i \neq j$ and $\langle v_i v_i^{\mathrm{T}}, v_i v_i^{\mathrm{T}}\rangle = 1$, and so the spectral decomposition is a Fourier expansion as in equation (3.113) and the eigenvalues are Fourier coefficients. From equation (3.114), we see that the eigenvalues can be represented as the inner product

$$c_i = \langle A,\, v_i v_i^{\mathrm{T}}\rangle. \tag{3.267}$$

The eigenvalues c_i have the same properties as the Fourier coefficients in any orthonormal expansion. In particular, the best approximating matrices within the subspace of $n \times n$ symmetric matrices spanned by $\{v_1 v_1^{\mathrm{T}}, \ldots, v_n v_n^{\mathrm{T}}\}$ are partial sums of the form of equation (3.256). In Sect. 3.10, however, we will develop a stronger result for approximation of matrices that does not rely on the restriction to this subspace and which applies to general, nonsquare matrices.

3.8.10.6 Powers of a Symmetric Matrix

If (c, v) is an eigenpair of the symmetric matrix A with $v^{\mathrm{T}}v = 1$, then for any $k = 1, 2, \ldots,$

$$\left(A - cvv^{\mathrm{T}}\right)^k = A^k - c^k vv^{\mathrm{T}}. \tag{3.268}$$

This follows from induction on k, for it clearly is true for $k = 1$, and if for a given k it is true that for $k - 1$

$$\left(A - cvv^{\mathrm{T}}\right)^{k-1} = A^{k-1} - c^{k-1} vv^{\mathrm{T}},$$

then by multiplying both sides by $(A - cvv^{\mathrm{T}})$, we see it is true for k:

$$\begin{aligned} \left(A - cvv^{\mathrm{T}}\right)^k &= \left(A^{k-1} - c^{k-1}vv^{\mathrm{T}}\right)\left(A - cvv^{\mathrm{T}}\right) \\ &= A^k - c^{k-1}vv^{\mathrm{T}}A - cA^{k-1}vv^{\mathrm{T}} + c^k vv^{\mathrm{T}} \\ &= A^k - c^k vv^{\mathrm{T}} - c^k vv^{\mathrm{T}} + c^k vv^{\mathrm{T}} \end{aligned}$$

$$= A^k - c^k v v^{\mathrm{T}}.$$

There is a similar result for nonsymmetric square matrices, where w and v are left and right eigenvectors, respectively, associated with the same eigenvalue c that can be scaled so that $w^{\mathrm{T}}v = 1$. (Recall that an eigenvalue of A is also an eigenvalue of A^{T}, and if w is a left eigenvector associated with the eigenvalue c, then $A^{\mathrm{T}}w = cw$.) The only property of symmetry used above was that we could scale $v^{\mathrm{T}}v$ to be 1; hence, we just need $w^{\mathrm{T}}v \neq 0$. This is clearly true for a diagonalizable matrix (from the definition). It is also true if c is simple (which is somewhat harder to prove). It is thus true for the dominant eigenvalue, which is simple, in two important classes of matrices we will consider in Sects. 8.7.2 and 8.7.3, positive matrices and irreducible nonnegative matrices.

If w and v are left and right eigenvectors of A associated with the same eigenvalue c and $w^{\mathrm{T}}v = 1$, then for $k = 1, 2, \ldots$,

$$\left(A - cvw^{\mathrm{T}}\right)^k = A^k - c^k v w^{\mathrm{T}}. \tag{3.269}$$

We can prove this by induction as above.

3.8.10.7 The Trace and Sums of Eigenvalues

For a general $n \times n$ matrix A with eigenvalues $c_1, \ldots, c_n$, we have $\mathrm{tr}(A) = \sum_{i=1}^n c_i$. (This is equation (3.228).) This is particularly easy to see for symmetric matrices because of equation (3.252), rewritten as $V^{\mathrm{T}}AV = C$, the diagonal matrix of the eigenvalues. For a symmetric matrix, however, we have a stronger result.

If A is an $n \times n$ symmetric matrix with eigenvalues $c_1 \geq \cdots \geq c_n$, and U is an $n \times k$ orthogonal matrix, with $k \leq n$, then

$$\mathrm{tr}(U^{\mathrm{T}}AU) \leq \sum_{i=1}^{k} c_i. \tag{3.270}$$

To see this, we represent U in terms of the columns of V, which span $\mathbb{R}^n$, as $U = VX$. Hence,

$$\begin{aligned} \mathrm{tr}(U^{\mathrm{T}}AU) &= \mathrm{tr}(X^{\mathrm{T}}V^{\mathrm{T}}AVX) \\ &= \mathrm{tr}(X^{\mathrm{T}}CX) \\ &= \sum_{i=1}^{n} x_i^{\mathrm{T}} x_i \, c_i, \end{aligned} \tag{3.271}$$

where x_i^{T} is the i^{th} row of X.

Now $X^{\mathrm{T}}X = X^{\mathrm{T}}V^{\mathrm{T}}VX = U^{\mathrm{T}}U = I_k$, so either $x_i^{\mathrm{T}}x_i = 0$ or $x_i^{\mathrm{T}}x_i = 1$, and $\sum_{i=1}^n x_i^{\mathrm{T}}x_i = k$. Because $c_1 \geq \cdots \geq c_n$, therefore $\sum_{i=1}^n x_i^{\mathrm{T}}x_i \, c_i \leq \sum_{i=1}^k c_i$, and so from equation (3.271) we have $\mathrm{tr}(U^{\mathrm{T}}AU) \leq \sum_{i=1}^k c_i$.

3.8.11 Positive Definite and Nonnegative Definite Matrices

The factorization of symmetric matrices in equation (3.252) yields some useful properties of positive definite and nonnegative definite matrices (introduced on page 91). We will briefly discuss these properties here and then return to the subject in Sect. 8.3 and discuss more properties of positive definite and nonnegative definite matrices.

3.8.11.1 Eigenvalues of Positive and Nonnegative Definite Matrices

In this book, we use the terms "nonnegative definite" and "positive definite" only for real symmetric matrices, so the eigenvalues of nonnegative definite or positive definite matrices are real.

Any real symmetric matrix is positive (nonnegative) definite if and only if all of its eigenvalues are positive (nonnegative). We can see this using the factorization (3.252) of a symmetric matrix. One factor is the diagonal matrix C of the eigenvalues, and the other factors are orthogonal. Hence, for any x, we have $x^{\mathrm{T}}Ax = x^{\mathrm{T}}VCV^{\mathrm{T}}x = y^{\mathrm{T}}Cy$, where $y = V^{\mathrm{T}}x$, and so

$$x^{\mathrm{T}}Ax > (\geq)\, 0$$

if and only if

$$y^{\mathrm{T}}Cy > (\geq)\, 0.$$

This, together with the resulting inequality (3.161) on page 114, implies that if P is a nonsingular matrix and D is a diagonal matrix, $P^{\mathrm{T}}DP$ is positive (nonnegative) if and only if the elements of D are positive (nonnegative).

A matrix (whether symmetric or not and whether real or not) all of whose eigenvalues have positive real parts is said to be *positive stable*. Positive stability is an important property in some applications, such as numerical solution of systems of nonlinear differential equations. Clearly, a positive definite matrix is positive stable.

3.8.11.2 Inverse of Positive Definite Matrices

If A is positive definite and $A = VCV^{\mathrm{T}}$ as in equation (3.252), then $A^{-1} = VC^{-1}V^{\mathrm{T}}$, and A^{-1} is positive definite because the elements of C^{-1} are positive.

3.8.11.3 Diagonalization of Positive Definite Matrices

If A is positive definite, the elements of the diagonal matrix C in equation (3.252) are positive, and so their square roots can be absorbed into V to form a nonsingular matrix P. The diagonalization in equation (3.253), $V^{\mathrm{T}}AV = C$, can therefore be reexpressed as

$$P^{\mathrm{T}}AP = I. \tag{3.272}$$

3.8.11.4 Square Roots of Positive and Nonnegative Definite Matrices

The factorization (3.252) together with the nonnegativity of the eigenvalues of positive and nonnegative definite matrices allows us to define a square root of such a matrix.

Let A be a nonnegative definite matrix and let V and C be as in equation (3.252): $A = VCV^{\mathrm{T}}$. Now, let S be a diagonal matrix whose elements are the nonnegative square roots of the corresponding elements of C. Then $(VSV^{\mathrm{T}})^2 = A$; hence, we write

$$A^{\frac{1}{2}} = VSV^{\mathrm{T}} \tag{3.273}$$

and call this matrix the *square root* of A. This definition of the square root of a matrix is an instance of equation (3.250) with $f(x) = \sqrt{x}$. We also can similarly define $A^{\frac{1}{r}}$ for $r > 0$.

We see immediately that $A^{\frac{1}{2}}$ is symmetric because A is symmetric.

Notice that if $A = I_2$ (the identity) and if $J = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$, then $J^2 = A$, but by the definition, J is not a square root of A.

If A is positive definite, A^{-1} exists and is positive definite. It therefore has a square root, which we denote as $A^{-\frac{1}{2}}$.

The square roots are nonnegative, and so $A^{\frac{1}{2}}$ is nonnegative definite. Furthermore, $A^{\frac{1}{2}}$ and $A^{-\frac{1}{2}}$ are positive definite if A is positive definite.

In Sect. 5.9.1, we will show that this $A^{\frac{1}{2}}$ is unique, so our reference to it as the square root is appropriate. (There is occasionally some ambiguity in the terms "square root" and "second root" and the symbols used to denote them. If x is a nonnegative scalar, the usual meaning of its square root, denoted by $\sqrt{x}$, is a nonnegative number, while its second roots, which may be denoted by $x^{\frac{1}{2}}$, are usually considered to be either of the numbers $\pm\sqrt{x}$. In our notation $A^{\frac{1}{2}}$, we mean *the* square root; that is, the nonnegative matrix, if it exists. Otherwise, we say the square root of the matrix does not exist.)

3.8.12 Generalized Eigenvalues and Eigenvectors

The characterization of an eigenvalue as a root of the determinant equation (3.220) can be extended to define a *generalized eigenvalue* of the square matrices A and B to be a root in c of the equation

$$\det(A - cB) = 0 \tag{3.274}$$

if a root exists.

Equation (3.274) is equivalent to $A - cB$ being singular; that is, for some c and some nonzero, finite v,

$$Av = cBv. \tag{3.275}$$

Such a c (if it exists) is called a *generalized eigenvalue* of A and B, and such a v (if it exists) is called a *generalized eigenvector* of A and B. In contrast to the existence of eigenvalues of any square matrix with finite elements, the generalized eigenvalues of two matrices may not exist; that is, they may be infinite. Notice that if (and only if) c is nonzero and finite, the roles of A and B can be interchanged in equation (3.275), and the generalized eigenvalue of B and A is $1/c$.

If A and B are $n \times n$ (that is, square) and B is nonsingular, then all n generalized eigenvalues of A and B exist (and are finite). These generalized eigenvalues are the eigenvalues of AB^{-1} or $B^{-1}A$. We see this because $\det(B) \neq 0$, and so if c_0 is any of the n (finite) eigenvalues of AB^{-1} or $B^{-1}A$, then $0 = \det(AB^{-1} - c_0 I) = \det(B^{-1}A - c_0 I) = \det(A - c_0 B) = 0$. Likewise, we see that any eigenvector of AB^{-1} or $B^{-1}A$ is a generalized eigenvector of A and B.

In the case of ordinary eigenvalues, we have seen that symmetry of the matrix induces some simplifications. In the case of generalized eigenvalues, symmetry together with positive definiteness also yields some useful properties, which we will discuss in Sect. 7.6.

Generalized eigenvalue problems often arise in multivariate statistical applications. Roy's maximum root statistic, for example, is the largest generalized eigenvalue of two matrices that result from operations on a partitioned matrix of sums of squares.

3.8.12.1 Matrix Pencils

As c ranges over the reals (or, more generally, the complex numbers), the set of matrices of the form $A - cB$ is called the *matrix pencil*, or just the *pencil*, generated by A and B, denoted as

$$(A, B).$$

(In this definition, A and B do not need to be square.) A generalized eigenvalue of the square matrices A and B is called an eigenvalue of the pencil.

A pencil is said to be *regular* if $\det(A - cB)$ is not identically 0 (assuming, of course, that $\det(A - cB)$ is defined, meaning A and B are square). An interesting special case of a regular pencil is when B is nonsingular. As we have seen, in that case, eigenvalues of the pencil (A, B) exist (and are finite) and are the same as the ordinary eigenvalues of AB^{-1} or $B^{-1}A$, and the ordinary eigenvectors of AB^{-1} or $B^{-1}A$ are eigenvectors of the pencil (A, B).

3.8.13 Singular Values and the Singular Value Decomposition (SVD)

An $n \times m$ matrix A can be factored as

$$A = UDV^{\mathrm{T}}, \tag{3.276}$$

where U is an $n \times n$ orthogonal matrix, V is an $m \times m$ orthogonal matrix, and D is an $n \times m$ diagonal matrix with nonnegative entries. (An $n \times m$ diagonal matrix has $\min(n, m)$ elements on the diagonal, and all other entries are zero.)

The factorization (3.276) is called the *singular value decomposition* (SVD) or the *canonical singular value factorization* of A. The elements on the diagonal of D, d_i, are called the *singular values* of A.

The SVD, which is unique, as we establish below, is one of the most important and most useful decompositions in all of matrix theory and applications.

There are $\min(n, m)$ singular values. We can rearrange the entries in D so that $d_1 \geq \cdots \geq d_{\min(n,m)}$, and by rearranging the columns of U correspondingly, nothing is changed.

We see that the SVD exists for any matrix by forming a square symmetric matrix and then using the decomposition in equation (3.252) on page 154. Let A be an $n \times m$ matrix A, and form $A^{\mathrm{T}}A = VCV^{\mathrm{T}}$. If $n \geq m$, we have

$$\begin{aligned} A^{\mathrm{T}}A &= VCV^{\mathrm{T}} \\ &= V[I\ 0] \begin{bmatrix} C \\ 0 \end{bmatrix} V^{\mathrm{T}} \\ &= \begin{bmatrix} V & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} C \\ 0 \end{bmatrix} V^{\mathrm{T}} \\ &= UDV^{\mathrm{T}}, \end{aligned}$$

as above. Note if $n = m$, the 0 partitions in the matrices are nonexistent. If, on the other hand, $n < m$, we form $D = [C\ 0]$ and proceed as before.

This same development follows for AA^{T}, and it is clear that the nonzero elements of the corresponding "C" matrix are the same (or property 14 on page 140 ensures that they are the same.)

The number of positive entries in D is the same as the rank of A. (We see this by first recognizing that the number of nonzero entries of D is obviously the rank of D, and multiplication by the full rank matrices U and V^{T} yields a product with the same rank from equations (3.158) and (3.159).)

From this development, we see that the squares of the singular values of A are the eigenvalues of $A^{\mathrm{T}}A$ and of AA^{T}, which are necessarily nonnegative. To state this more clearly (and using some additional facts developed previously, including property 13 on page 140), let A be an $n \times m$ matrix with rank r, and let d be a singular value of A. We have

- $c = d^2$ is an eigenvalue of $A^{\mathrm{T}}A$;
- $c = d^2$ is an eigenvalue of AA^{T};
- if c is a nonzero eigenvalue of $A^{\mathrm{T}}A$, then there is a singular value d of A such that $d^2 = c$; and
- there are r nonzero singular values of A, and r nonzero eigenvalues of $A^{\mathrm{T}}A$ and of AA^{T}.

These relationships between singular values and eigenvalues are some of the most important properties of singular values and the singular value de-

composition. In particular, from these we can see that the singular value decomposition is unique (with the same qualifications attendant to uniqueness of eigenvalues and eigenvectors, relating to ordering of the elements and selection of vectors corresponding to nonunique values).

An additional observation is that if A is symmetric, the singular values of A are the absolute values of the eigenvalues of A.

From the factorization (3.276) defining the singular values, we see that

- the singular values of A^{T} are the same as those of A.

As pointed out above, for a matrix with more rows than columns, in an alternate definition of the singular value decomposition, the matrix U is $n \times m$ with orthogonal columns, and D is an $m \times m$ diagonal matrix with nonnegative entries. Likewise, for a matrix with more columns than rows, the singular value decomposition can be defined as above but with the matrix V being $m \times n$ with orthogonal columns and D being $n \times n$ and diagonal with nonnegative entries.

We often partition D to separate the zero singular values. If the rank of the matrix is r, we have $d_1 \geq \cdots \geq d_r > 0$ (with the common indexing), and if $r < \min(n, m)$, then $d_{r+1} = \cdots = d_{\min(n,m)} = 0$. In this case

$$D = \begin{bmatrix} D_r & 0 \\ 0 & 0 \end{bmatrix},$$

where $D_r = \mathrm{diag}((d_1, \ldots, d_r))$.

3.8.13.1 The Fourier Expansion in Terms of the Singular Value Decomposition

From equation (3.276), we see that the general matrix A with rank r also has a Fourier expansion, similar to equation (3.256), in terms of the singular values and outer products of the columns of the U and V matrices:

$$A = \sum_{i=1}^{r} d_i u_i v_i^{\mathrm{T}}. \tag{3.277}$$

This is also called a spectral decomposition. The $u_i v_i^{\mathrm{T}}$ matrices in equation (3.277) have the property that $\langle u_i v_i^{\mathrm{T}}, u_j v_j^{\mathrm{T}} \rangle = 0$ for $i \neq j$ and $\langle u_i v_i^{\mathrm{T}}, u_i v_i^{\mathrm{T}} \rangle = 1$, and so the spectral decomposition is a Fourier expansion as in equation (3.113), and the singular values are Fourier coefficients.

The singular values d_i have the same properties as the Fourier coefficients in any orthonormal expansion. For example, from equation (3.114), we see that the singular values can be represented as the inner product

$$d_i = \langle A, u_i v_i^{\mathrm{T}} \rangle.$$

After we have discussed matrix norms in the next section, we will formulate Parseval's identity for this Fourier expansion.

The spectral decomposition is a *rank-one decomposition*, since each of the matrices $u_i v_i^{\mathrm{T}}$ has rank one.

3.9 Matrix Norms

Norms on matrices are scalar functions of matrices with the three properties on page 25 that define a norm in general. Matrix norms are often required to have another property, called the *consistency property*, in addition to the properties listed on page 25, which we repeat here for convenience. Assume A and B are matrices conformable for the operations shown.

1. Nonnegativity and mapping of the identity:
   if $A \neq 0$, then $\|A\| > 0$, and $\|0\| = 0$.
2. Relation of scalar multiplication to real multiplication:
   $\|aA\| = |a| \, \|A\|$ for real a.
3. Triangle inequality:
   $\|A + B\| \leq \|A\| + \|B\|$.
4. Consistency property:
   $\|AB\| \leq \|A\| \, \|B\|$.

Some people do not require the consistency property for a matrix norm. Most useful matrix norms have the property, however, and we will consider it to be a requirement in the definition. The consistency property for multiplication is similar to the triangular inequality for addition.

Any function from $\mathbb{R}^{n \times m}$ to $\mathbb{R}$ that satisfies these four properties is a matrix norm.

A matrix norm, as any norm, is necessarily convex. (See page 26.)

We note that the four properties of a matrix norm do not imply that it is invariant to transposition of a matrix, and in general, $\|A^{\mathrm{T}}\| \neq \|A\|$. Some matrix norms are the same for the transpose of a matrix as for the original matrix. For instance, because of the property of the matrix inner product given in equation (3.110), we see that a norm defined by that inner product would be invariant to transposition.

For a square matrix A, the consistency property for a matrix norm yields

$$\|A^k\| \leq \|A\|^k \tag{3.278}$$

for any positive integer k.

A matrix norm $\|\cdot\|$ is *orthogonally invariant* if A and B being orthogonally similar implies $\|A\| = \|B\|$.

3.9.1 Matrix Norms Induced from Vector Norms

Some matrix norms are defined in terms of vector norms. For clarity, we will denote a vector norm as $\|\cdot\|_{\mathrm{v}}$ and a matrix norm as $\|\cdot\|_{\mathrm{M}}$. (This notation is meant to be generic; that is, $\|\cdot\|_{\mathrm{v}}$ represents any vector norm.) For the matrix $A \in \mathbb{R}^{n\times m}$, the matrix norm $\|\cdot\|_{\mathrm{M}}$ *induced* by $\|\cdot\|_{\mathrm{v}}$ is defined by

$$\|A\|_{\mathrm{M}} = \max_{x\neq 0} \frac{\|Ax\|_{\mathrm{v}}}{\|x\|_{\mathrm{v}}}. \tag{3.279}$$

(Note that there are some minor subtleties here; $Ax \in \mathbb{R}^n$ while $x \in \mathbb{R}^m$, so the two vector norms are actually different. Of course, in practice, an induced norm is defined in terms of vector norms of the same "type", for example L_p norms with the same p.)

An induced matrix norm is also sometimes called an *operator norm*.

It is easy to see that an induced norm is indeed a matrix norm. The first three properties of a norm are immediate, and the consistency property can be verified by applying the definition (3.279) to AB and replacing Bx with y; that is, using Ay.

We usually drop the v or M subscript, and the notation $\|\cdot\|$ is overloaded to mean either a vector or matrix norm. (Overloading of symbols occurs in many contexts, and we usually do not even recognize that the meaning is context-dependent. In computer language design, overloading must be recognized explicitly because the language specifications must be explicit.)

The induced norm of A given in equation (3.279) is sometimes called the *maximum magnification* by A. The expression looks very similar to the maximum eigenvalue, and indeed it is in some cases.

For any vector norm and its induced matrix norm, we see from equation (3.279) that

$$\|Ax\| \leq \|A\|\,\|x\| \tag{3.280}$$

because $\|x\| \geq 0$.

3.9.1.1 L_p Matrix Norms

The matrix norms that correspond to the L_p vector norms are defined for the $n \times m$ matrix A as

$$\|A\|_p = \max_{\|x\|_p=1} \|Ax\|_p. \tag{3.281}$$

(Notice that the restriction on $\|x\|_p$ makes this an induced norm as defined in equation (3.279). Notice also the overloading of the symbols; the norm on the left that is being defined is a matrix norm, whereas those on the right of the equation are vector norms.) It is clear that the L_p matrix norms satisfy the consistency property, because they are induced norms.

The L_1 and L_∞ norms have interesting simplifications of equation (3.279):

$$\|A\|_1 = \max_j \sum_i |a_{ij}|, \tag{3.282}$$

so the L_1 is also called the *column-sum norm*; and

$$\|A\|_\infty = \max_i \sum_j |a_{ij}|, \tag{3.283}$$

so the L_∞ is also called the *row-sum norm*. We see these relationships by considering the L_p norm of the vector

$$v = (a_{1*}^{\mathrm{T}}x, \ldots, a_{n*}^{\mathrm{T}}x),$$

where a_{i*} is the i^{th} row of A, with the restriction that $\|x\|_p = 1$. The L_p norm of this vector is based on the absolute values of the elements; that is, $|\sum_j a_{ij}x_j|$ for $i = 1, \ldots, n$. Because we are free to choose x (subject to the restriction that $\|x\|_p = 1$), for a given i, we can choose the sign of each x_j to maximize the overall expression. For example, for a fixed i, we can choose each x_j to have the same sign as a_{ij}, and so $|\sum_j a_{ij}x_j|$ is the same as $\sum_j |a_{ij}|\,|x_j|$.

For the column-sum norm, the L_1 norm of v is $\sum_i |a_{i*}^{\mathrm{T}}x|$. The elements of x are chosen to maximize this under the restriction that $\sum |x_j| = 1$. The maximum of the expression is attained by setting $x_k = \text{sign}(\sum_i a_{ik})$, where k is such that $|\sum_i a_{ik}| \geq |\sum_i a_{ij}|$, for $j = 1, \ldots, m$, and $x_q = 0$ for $q = 1, \ldots m$ and $q \neq k$. (If there is no unique k, any choice will yield the same result.) This yields equation (3.282).

For the row-sum norm, the L_∞ norm of v is

$$\max_i |a_{i*}^{\mathrm{T}}x| = \max_i \sum_j |a_{ij}|\,|x_j|$$

when the sign of x_j is chosen appropriately (for a given i). The elements of x must be chosen so that $\max|x_j| = 1$; hence, each x_j is chosen as ± 1. The maximum $|a_{i*}^{\mathrm{T}}x|$ is attained by setting $x_j = \text{sign}(a_{kj})$, for $j = 1, \ldots m$, where k is such that $\sum_j |a_{kj}| \geq \sum_j |a_{ij}|$, for $i = 1, \ldots, n$. This yields equation (3.283).

From equations (3.282) and (3.283), we see that

$$\|A^{\mathrm{T}}\|_\infty = \|A\|_1. \tag{3.284}$$

Alternative formulations of the L_2 norm of a matrix are not so obvious from equation (3.281). It is related to the eigenvalues (or the singular values) of the matrix. The L_2 matrix norm is related to the spectral radius (page 142):

$$\|A\|_2 = \sqrt{\rho(A^{\mathrm{T}}A)}, \tag{3.285}$$

(see Exercise 3.34, page 182). Because of this relationship, the L_2 matrix norm is also called the *spectral norm*.

From the invariance of the singular values to matrix transposition, we see that positive eigenvalues of $A^{\mathrm{T}}A$ are the same as those of AA^{T}; hence, $\|A^{\mathrm{T}}\|_2 = \|A\|_2$.

For Q orthogonal, the L_2 vector norm has the important property

$$\|Qx\|_2 = \|x\|_2 \tag{3.286}$$

(see Exercise 3.35a, page 182). For this reason, an orthogonal matrix is sometimes called an *isometric matrix*. By the proper choice of x, it is easy to see from equation (3.286) that

$$\|Q\|_2 = 1. \tag{3.287}$$

Also from this we see that if A and B are orthogonally similar, then $\|A\|_2 = \|B\|_2$; hence, the spectral matrix norm is orthogonally invariant.

The L_2 matrix norm is a Euclidean-type norm since it is induced by the Euclidean vector norm (but it is not called the Euclidean matrix norm; see below).

3.9.1.2 L_1, L_2, and L_∞ Norms of Symmetric Matrices

For a symmetric matrix A, we have the obvious relationships

$$\|A\|_1 = \|A\|_\infty \tag{3.288}$$

and, from equation (3.285),

$$\|A\|_2 = \rho(A). \tag{3.289}$$

3.9.2 The Frobenius Norm—The "Usual" Norm

The *Frobenius norm* is defined as

$$\|A\|_{\mathrm{F}} = \sqrt{\sum_{i,j} a_{ij}^2}. \tag{3.290}$$

It is easy to see that this measure has the consistency property (Exercise 3.37), as a norm must. The Frobenius norm is sometimes called the *Euclidean matrix norm* and denoted by $\|\cdot\|_{\mathrm{E}}$, although the L_2 matrix norm is more directly based on the Euclidean vector norm, as we mentioned above. We will usually use the notation $\|\cdot\|_{\mathrm{F}}$ to denote the Frobenius norm. Occasionally we use $\|\cdot\|$ without the subscript to denote the Frobenius norm, but usually the symbol without the subscript indicates that any norm could be used in the expression. The Frobenius norm is also often called the "usual norm", which emphasizes the fact that it is one of the most useful matrix norms. Other names sometimes used to refer to the Frobenius norm are *Hilbert-Schmidt norm* and *Schur norm*.

From the definition, we have $\|A^{\mathrm{T}}\|_{\mathrm{F}} = \|A\|_{\mathrm{F}}$. We have seen that the L_2 matrix norm also has this property.

Another important property of the Frobenius norm that is obvious from the definition is

$$\|A\|_{\mathrm{F}} = \sqrt{\mathrm{tr}(A^{\mathrm{T}}A)} \tag{3.291}$$

$$= \sqrt{\langle A, A\rangle}; \tag{3.292}$$

that is,

- the Frobenius norm is the norm that arises from the matrix inner product (see page 97).

The complete vector space $\mathbb{R}^{n\times m}$ with the Frobenius norm is therefore a Hilbert space.

Another thing worth noting for a square A is the relationship of the Frobenius norm to the eigenvalues c_i of A:

$$\|A\|_{\mathrm{F}} = \sqrt{\sum c_i \bar{c}_i}, \tag{3.293}$$

and if A is also symmetric,

$$\|A\|_{\mathrm{F}} = \sqrt{\sum c_i^2}, \tag{3.294}$$

These follow from equation (3.291) and equation (3.228) on page 141.

Similar to defining the angle between two vectors in terms of the inner product and the norm arising from the inner product, we define the *angle* between two matrices A and B of the same size and shape as

$$\mathrm{angle}(A, B) = \cos^{-1}\left(\frac{\langle A, B\rangle}{\|A\|_{\mathrm{F}}\|B\|_{\mathrm{F}}}\right). \tag{3.295}$$

If Q is an $n \times m$ orthogonal matrix, then

$$\|Q\|_{\mathrm{F}} = \sqrt{m} \tag{3.296}$$

(see equation (3.216)).

If A and B are orthogonally similar (see equation (3.242)), then

$$\|A\|_{\mathrm{F}} = \|B\|_{\mathrm{F}};$$

that is, the Frobenius norm is an orthogonally invariant norm. To see this, let $A = Q^{\mathrm{T}}BQ$, where Q is an orthogonal matrix. Then

$$\begin{aligned}\|A\|_{\mathrm{F}}^2 &= \mathrm{tr}(A^{\mathrm{T}}A)\\ &= \mathrm{tr}(Q^{\mathrm{T}}B^{\mathrm{T}}QQ^{\mathrm{T}}BQ)\end{aligned}$$

$$\begin{aligned} &= \mathrm{tr}(B^{\mathrm{T}}BQQ^{\mathrm{T}}) \\ &= \mathrm{tr}(B^{\mathrm{T}}B) \\ &= \|B\|_{\mathrm{F}}^2. \end{aligned}$$

(The norms are nonnegative, of course, and so equality of the squares is sufficient.)

3.9.2.1 The Frobenius Norm and the Singular Values

Several important properties result because the Frobenius norm arises from an inner product. For example, following the Fourier expansion in terms of the singular value decomposition, equation (3.277), we mentioned that the singular values have the general properties of Fourier coefficients; for example, they satisfy Parseval's identity, equation (2.60), on page 41. This identity states that the sum of the squares of the Fourier coefficients is equal to the square of the norm that arises from the inner product used in the Fourier expansion. Hence, we have the important property of the Frobenius norm that it is the L_2 norm of the vector of singular values of the matrix. For the $n \times m$ matrix A, let d be the $\min(n, m)$-vector of singular values of A. Then

$$\|A\|_{\mathrm{F}}^2 = \|d\|_2. \tag{3.297}$$

Compare equations (3.293) and (3.294) for square matrices.

3.9.3 Other Matrix Norms

There are two different ways of generalizing the Frobenius norm. One is a simple generalization of the definition in equation (3.290). For $p \geq 1$, it is the *Frobenius p norm*:

$$\|A\|_{\mathrm{F}_p} = \left(\sum_{i,j} |a_{ij}|^p \right)^{1/p}. \tag{3.298}$$

Some people refer to this as the L_p norm of the matrix. As we have seen, the L_p matrix norm is different, but there is a simple relationship of the Frobenius p matrix norm to the L_p vector norm:

$$\|A\|_{\mathrm{F}_p} = \|\mathrm{vec}(A)\|_p. \tag{3.299}$$

This relationship of the matrix norm to a vector norm sometimes makes computational problems easier.

The Frobenius 2 norm is the ordinary Frobenius norm.

Another generalization of the Frobenius norm arises from its relation to the singular values given in equation (3.297). For $p \geq 1$, it is the *Schatten p norm*:

$$\|A\|_{\mathrm{S}_p} = \|d\|_p, \tag{3.300}$$

where d is the vector of singular values of A.

The Schatten 2 norm is the ordinary Frobenius norm.

The Schatten 1 norm is called the *nuclear norm* (because of its relationship to "nuclear operators", which are linear operators that preserve local convexity). It is also sometimes called the *trace norm*, because

$$\|d\|_1 = \mathrm{tr}\left((A^{\mathrm{T}}A)^{1/2}\right). \tag{3.301}$$

The Schatten ∞ norm is the spectral norm.

3.9.4 Matrix Norm Inequalities

There is an equivalence among any two matrix norms similar to that of expression (2.39) for vector norms (over finite-dimensional vector spaces). If $\|\cdot\|_a$ and $\|\cdot\|_b$ are matrix norms, then there are positive numbers r and s such that, for any matrix A,

$$r\|A\|_b \leq \|A\|_a \leq s\|A\|_b. \tag{3.302}$$

We will not prove this result in general but, in Exercise 3.39, ask the reader to do so for matrix norms induced by vector norms. These induced norms include the matrix L_p norms of course.

If A is an $n \times m$ real matrix, we have some specific instances of (3.302):

$$\|A\|_\infty \leq \sqrt{m}\,\|A\|_{\mathrm{F}}, \tag{3.303}$$

$$\|A\|_{\mathrm{F}} \leq \sqrt{\min(n,m)}\,\|A\|_2, \tag{3.304}$$

$$\|A\|_2 \leq \sqrt{m}\,\|A\|_1, \tag{3.305}$$

$$\|A\|_1 \leq \sqrt{n}\,\|A\|_2, \tag{3.306}$$

$$\|A\|_2 \leq \|A\|_{\mathrm{F}}, \tag{3.307}$$

$$\|A\|_{\mathrm{F}} \leq \sqrt{n}\,\|A\|_\infty. \tag{3.308}$$

See Exercise 3.40 on page 182.

Compare these inequalities with those for L_p vector norms on page 28. Recall specifically that for vector L_p norms we had the useful fact that for a given x and for $p \geq 1$, $\|x\|_p$ is a nonincreasing function of p; and specifically we had inequality (2.34):

$$\|x\|_\infty \leq \|x\|_2 \leq \|x\|_1.$$

There is a related inequality involving matrices:

$$\|A\|_2^2 \leq \|A\|_1 \|A\|_\infty. \tag{3.309}$$

3.9.5 The Spectral Radius

The spectral radius is the appropriate measure of the condition of a square matrix for certain iterative algorithms. Except in the case of symmetric matrices, as shown in equation (3.289), the spectral radius is not a norm (see Exercise 3.41a).

We have for any norm $\|\cdot\|$ and any square matrix A that

$$\rho(A) \leq \|A\|. \tag{3.310}$$

To see this, we consider the associated eigenvalue and eigenvector, c_i and v_i, and form the matrix $V = [v_i|0|\cdots|0]$. This yields $c_i V = AV$, and by the consistency property of any matrix norm,

$$\begin{aligned} |c_i|\|V\| &= \|c_i V\| \\ &= \|AV\| \\ &\leq \|A\|\,\|V\|, \end{aligned}$$

or

$$|c_i| \leq \|A\|,$$

(see also Exercise 3.41b).

The inequality (3.310) and the L_1 and L_∞ norms yield useful bounds on the eigenvalues and the maximum absolute row and column sums of matrices: the modulus of any eigenvalue is no greater than the largest sum of absolute values of the elements in any row or column. (These were inequalities (3.235) and (3.236) on page 142.)

The inequality (3.310) and equation (3.289) also yield a minimum property of the L_2 norm of a symmetric matrix A:

$$\|A\|_2 \leq \|A\|.$$

3.9.6 Convergence of a Matrix Power Series

We define the convergence of a sequence of matrices in terms of the convergence of a sequence of their norms, just as we did for a sequence of vectors (on page 32). We say that a sequence of matrices $A_1, A_2, \ldots$ (of the same shape) converges to the matrix A with respect to the norm $\|\cdot\|$ if the sequence of real numbers $\|A_1 - A\|$, $\|A_2 - A\|$, $\ldots$ converges to 0. Because of the equivalence property of norms, the choice of the norm is irrelevant. Also, because of inequality (3.310), we see that the convergence of the sequence of spectral radii $\rho(A_1 - A), \rho(A_2 - A), \ldots$ to 0 must imply the convergence of $A_1, A_2, \ldots$ to A.

3.9.6.1 Conditions for Convergence of a Sequence of Powers to 0

For a square matrix A, we have the important fact that

$$A^k \to 0, \quad \text{if } \|A\| < 1, \tag{3.311}$$

where 0 is the square zero matrix of the same order as A and $\|\cdot\|$ is any matrix norm. (The consistency property is required.) This convergence follows from inequality (3.278) because that yields $\lim_{k\to\infty} \|A^k\| \leq \lim_{k\to\infty} \|A\|^k$, and so if $\|A\| < 1$, then $\lim_{k\to\infty} \|A^k\| = 0$.

Now consider the spectral radius. Because of the spectral decomposition, we would expect the spectral radius to be related to the convergence of a sequence of powers of a matrix. If $A^k \to 0$, then for any conformable vector x, $A^k x \to 0$; in particular, for the eigenvector $v_1 \neq 0$ corresponding to the dominant eigenvalue c_1, we have $A^k v_1 = c_1^k v_1 \to 0$. For $c_1^k v_1$ to converge to zero, we must have $|c_1| < 1$; that is, $\rho(A) < 1$. We can also show the converse:

$$A^k \to 0 \quad \text{if } \rho(A) < 1. \tag{3.312}$$

We will do this by defining a norm $\|\cdot\|_d$ in terms of the L_1 matrix norm in such a way that $\rho(A) < 1$ implies $\|A\|_d < 1$. Then we can use equation (3.311) to establish the convergence.

Let $A = QTQ^{\mathrm{T}}$ be the Schur factorization of the $n \times n$ matrix A, where Q is orthogonal and T is upper triangular with the same eigenvalues as A, $c_1, \ldots, c_n$. Now for any $d > 0$, form the diagonal matrix $D = \mathrm{diag}((d^1, \ldots, d^n))$. Notice that DTD^{-1} is an upper triangular matrix and its diagonal elements (which are its eigenvalues) are the same as the eigenvalues of T and A. Consider the column sums of the absolute values of the elements of DTD^{-1}:

$$|c_j| + \sum_{i=1}^{j-1} d^{-(j-i)}|t_{ij}|.$$

Now, because $|c_j| \leq \rho(A)$ for given $\epsilon > 0$, by choosing d large enough, we have

$$|c_j| + \sum_{i=1}^{j-1} d^{-(j-i)}|t_{ij}| < \rho(A) + \epsilon,$$

or

$$\|DTD^{-1}\|_1 = \max_j \left(|c_j| + \sum_{i=1}^{j-1} d^{-(j-i)}|t_{ij}| \right) < \rho(A) + \epsilon.$$

Now define $\|\cdot\|_d$ for any $n \times n$ matrix X, where Q is the orthogonal matrix in the Schur factorization and D is as defined above, as

$$\|X\|_d = \|(QD^{-1})^{-1} X (QD^{-1})\|_1. \tag{3.313}$$

It is clear that $\|\cdot\|_d$ is a norm (Exercise 3.42). Furthermore,

$$\begin{aligned}\|A\|_d &= \|(QD^{-1})^{-1}A(QD^{-1})\|_1 \\ &= \|DTD^{-1}\|_1 \\ &< \rho(A) + \epsilon,\end{aligned}$$

and so if $\rho(A) < 1$, ϵ and d can be chosen so that $\|A\|_d < 1$, and by equation (3.311) above, we have $A^k \to 0$; hence, we conclude that

$$A^k \to 0 \quad \text{if and only if } \rho(A) < 1. \tag{3.314}$$

Informally, we see that A^k goes to 0 more rapidly the smaller is $\rho(A)$.

We will discuss convergence of a sequence of powers of an important special class of matrices with spectral radii possibly greater than or equal to 1 on page 378.

3.9.6.2 Another Perspective on the Spectral Radius: Relation to Norms

From inequality (3.310) and the fact that $\rho(A^k) = \rho(A)^k$, we have

$$\rho(A) \leq \|A^k\|^{1/k}, \tag{3.315}$$

where $\|\cdot\|$ is any matrix norm. Now, for any $\epsilon > 0$, $\rho\big(A/(\rho(A)+\epsilon)\big) < 1$ and so

$$\lim_{k\to\infty} \big(A/(\rho(A)+\epsilon)\big)^k = 0$$

from expression (3.314); hence,

$$\lim_{k\to\infty} \frac{\|A^k\|}{(\rho(A)+\epsilon)^k} = 0.$$

There is therefore a positive integer M_ϵ such that $\|A^k\|/(\rho(A)+\epsilon)^k < 1$ for all $k > M_\epsilon$, and hence $\|A^k\|^{1/k} < (\rho(A)+\epsilon)$ for $k > M_\epsilon$. We have therefore, for any $\epsilon > 0$,

$$\rho(A) \leq \|A^k\|^{1/k} < \rho(A) + \epsilon \quad \text{for } k > M_\epsilon,$$

and thus

$$\lim_{k\to\infty} \|A^k\|^{1/k} = \rho(A). \tag{3.316}$$

Compare this with the inequality (3.310).

3.9.6.3 Convergence of a Power Series: Inverse of $I - A$

Consider the power series in an $n \times n$ matrix such as in equation (3.186) on page 120,

$$I + A + A^2 + A^3 + \cdots .$$

In the standard fashion for dealing with series, we form the partial sum

$$S_k = I + A + A^2 + A^3 + \cdots A^k$$

and consider $\lim_{k\to\infty} S_k$. We first note that

$$(I - A)S_k = I - A^{k+1}$$

and observe that if $A^{k+1} \to 0$, then $S_k \to (I-A)^{-1}$, which is equation (3.186). Therefore,

$$(I - A)^{-1} = I + A + A^2 + A^3 + \cdots \quad \text{if } \|A\| < 1. \tag{3.317}$$

3.9.6.4 Nilpotent Matrices

As we discussed on page 77, for some nonzero square matrices, $A^k = 0$ for a finite integral value of k. If $A^2 = 0$, such a matrix is a *nilpotent* matrix (otherwise, it is nilpotent with an index greater than 2). A matrix such as we discussed above for which $A^k \to 0$, but for any finite k, $A^k \neq 0$, is not called a nilpotent matrix.

From the definition, it is clear that the Drazin inverse of any nilpotent matrix is 0.

We have seen in equation (3.314) that $A^k \to 0$ if and only if $\rho(A) < 1$. The condition in equation (3.311) on any norm is not necessary, however; that is, if $A^k \to 0$, it may be the case that, for some norm, $\|A\| > 1$. In fact, even for an idempotent matrix (for which $A^k = 0$ for finite k), it may be the case that $\|A\| > 1$. A simple example is

$$A = \begin{bmatrix} 0 & 2 \\ 0 & 0 \end{bmatrix}.$$

For this matrix, $A^2 = 0$, yet $\|A\|_1 = \|A\|_2 = \|A\|_\infty = \|A\|_{\mathrm{F}} = 2$.

At this point, I list some more properties of nilpotent matrices that involve concepts we had not introduced when we first discussed nilpotent matrices. It is easy to see that if $A_{n\times n}$ is nilpotent, then

$$\mathrm{tr}(A) = 0, \tag{3.318}$$

$$\det(A) = 0, \tag{3.319}$$

$$\rho(A) = 0, \tag{3.320}$$

(that is, all eigenvalues of A are 0), and

$$\text{rank}(A) \leq n - 1. \tag{3.321}$$

You are asked to supply the proofs of these statements in Exercise 3.43b.

In applications, for example in time series or other stochastic processes, because of expression (3.314), the spectral radius is often the most useful. Stochastic processes may be characterized by whether the absolute value of the dominant eigenvalue (spectral radius) of a certain matrix is less than 1. Interesting special cases occur when the dominant eigenvalue is equal to 1.

3.10 Approximation of Matrices

In Sect. 2.2.6, we discussed the problem of approximating a given vector in terms of vectors from a lower dimensional space. Likewise, it is often of interest to approximate one matrix by another.

In statistical applications, we may wish to find a matrix of smaller rank that contains a large portion of the information content of a matrix of larger rank ("dimension reduction" as on page 428; or variable selection as in Sect. 9.5.2, for example), or we may want to impose conditions on an estimate that it have properties known to be possessed by the estimand (positive definiteness of the correlation matrix, for example, as in Sect. 9.5.6).

In numerical linear algebra, we may wish to find a matrix that is easier to compute or that has properties that ensure more stable computations.

Finally, we may wish to represent a matrix as a sum or a product of other matrices with restrictions on those matrices that do not allow an exact representation. (A nonnegative factorization as discussed in Sect. 5.10.1 is an example.)

3.10.1 Measures of the Difference Between Two Matrices

A natural way to assess the goodness of the approximation is by a norm of the difference (that is, by a *metric induced by a norm*), as discussed on page 32. If $\widetilde{A}$ is an approximation to A, we could measure the quality of the approximation by $\Delta(A, \widetilde{A}) = \|A - \widetilde{A}\|$ for some norm $\|\cdot\|$. The measure $\Delta(A, \widetilde{A})$ is a metric, as defined on page 32, and is a common way of measuring the "distance" between two matrices.

Other ways of measuring the difference between two matrices may be based on how much the entropy of one divergences from that of the other. This may make sense if all elements in the matrices are positive. The Kullback-Leibler divergence between distributions is based on this idea. Because one distribution is used to normalize the other one, the Kullback-Leibler divergence is not a metric. If all elements of the matrices $\widetilde{A}$ and A are positive, the Kullback-Leibler divergence measure for how much the matrix $\widetilde{A}$ differs from A is

$$d_{\mathrm{KL}}(A - \widetilde{A}) = \sum_{ij} \left(\tilde{a}_{ij} \log \left(\frac{\tilde{a}_{ij}}{a_{ij}} \right) - \tilde{a}_{ij} + a_{ij} \right). \tag{3.322}$$

The most commonly-used measure of the goodness of an approximation uses the norm that arises from the inner product (the Frobenius norm).

3.10.2 Best Approximation with a Matrix of Given Rank

Suppose we want the best approximation to an $n \times m$ matrix A of rank r by a matrix $\widetilde{A}$ in $\mathbb{R}^{n \times m}$ but with smaller rank, say k; that is, we want to find $\widetilde{A}$ of rank k such that

$$\|A - \widetilde{A}\|_{\mathrm{F}} \tag{3.323}$$

is a minimum for all $\widetilde{A} \in \mathbb{R}^{n \times m}$ of rank k.

We have an orthogonal basis in terms of the singular value decomposition, equation (3.277), for some subspace of $\mathbb{R}^{n \times m}$, and we know that the Fourier coefficients provide the best approximation for any subset of k basis matrices, as in equation (2.65). This Fourier fit would have rank k as required, but it would be the best only within that set of expansions. (This is the limitation imposed in equation (2.65).) Another approach to determine the best fit could be developed by representing the columns of the approximating matrix as linear combinations of the given matrix A and then expanding $\|A - \widetilde{A}\|_{\mathrm{F}}^2$. Neither the Fourier expansion nor the restriction $\mathcal{V}(\widetilde{A}) \subset \mathcal{V}(A)$ permits us to address the question of what is the overall best approximation of rank k within $\mathbb{R}^{n \times m}$. As we see below, however, there is a minimum of expression (3.323) that occurs within $\mathcal{V}(A)$, and a minimum is at the truncated Fourier expansion in the singular values (equation (3.277)).

To state this more precisely, let A be an $n \times m$ matrix of rank r with singular value decomposition

$$A = U \begin{bmatrix} D_r & 0 \\ 0 & 0 \end{bmatrix} V^{\mathrm{T}},$$

where $D_r = \mathrm{diag}((d_1, \ldots, d_r))$, and the singular values are indexed so that $d_1 \geq \cdots \geq d_r > 0$. Then, for all $n \times m$ matrices X with rank $k < r$,

$$\|A - X\|_{\mathrm{F}}^2 \geq \sum_{i=k+1}^{r} d_i^2, \tag{3.324}$$

and this minimum occurs for $X = \widetilde{A}$, where

$$\widetilde{A} = U \begin{bmatrix} D_k & 0 \\ 0 & 0 \end{bmatrix} V^{\mathrm{T}}. \tag{3.325}$$

To see this, for any X, let Q be an $n \times k$ matrix whose columns are an orthonormal basis for $\mathcal{V}(X)$, and let $X = QY$, where Y is a $k \times m$ matrix, also of rank k. The minimization problem now is

$$\min_{Y} \|A - QY\|_{\mathrm{F}}$$

with the restriction $\mathrm{rank}(Y) = k$.

Now, expanding, completing the Gramian and using its nonnegative definiteness, and permuting the factors within a trace, we have

$$\begin{aligned}
\|A - QY\|_{\mathrm{F}}^2 &= \mathrm{tr}\left((A - QY)^{\mathrm{T}}(A - QY)\right) \\
&= \mathrm{tr}\left(A^{\mathrm{T}}A\right) + \mathrm{tr}\left(Y^{\mathrm{T}}Y - A^{\mathrm{T}}QY - Y^{\mathrm{T}}Q^{\mathrm{T}}A\right) \\
&= \mathrm{tr}\left(A^{\mathrm{T}}A\right) + \mathrm{tr}\left((Y - Q^{\mathrm{T}}A)^{\mathrm{T}}(Y - Q^{\mathrm{T}}A)\right) - \mathrm{tr}\left(A^{\mathrm{T}}QQ^{\mathrm{T}}A\right) \\
&\geq \mathrm{tr}\left(A^{\mathrm{T}}A\right) - \mathrm{tr}\left(Q^{\mathrm{T}}AA^{\mathrm{T}}Q\right).
\end{aligned}$$

The squares of the singular values of A are the eigenvalues of $A^{\mathrm{T}}A$, and so $\mathrm{tr}(A^{\mathrm{T}}A) = \sum_{i=1}^{r} d_i^2$. The eigenvalues of $A^{\mathrm{T}}A$ are also the eigenvalues of AA^{T}, and so, from inequality (3.270), $\mathrm{tr}(Q^{\mathrm{T}}AA^{\mathrm{T}}Q) \leq \sum_{i=1}^{k} d_i^2$, and so

$$\|A - X\|_{\mathrm{F}}^2 \geq \sum_{i=1}^{r} d_i^2 - \sum_{i=1}^{k} d_i^2;$$

hence, we have inequality (3.324). (This technique of "completing the Gramian" when an orthogonal matrix is present in a sum is somewhat similar to the technique of completing the square; it results in the difference of two Gramian matrices, which are defined in Sect. 3.3.10.)

Direct expansion of $\|A - \widetilde{A}\|_{\mathrm{F}}^2$ yields

$$\mathrm{tr}\left(A^{\mathrm{T}}A\right) - 2\mathrm{tr}\left(A^{\mathrm{T}}\widetilde{A}\right) + \mathrm{tr}\left(\widetilde{A}^{\mathrm{T}}\widetilde{A}\right) = \sum_{i=1}^{r} d_i^2 - \sum_{i=1}^{k} d_i^2,$$

and hence $\widetilde{A}$ is the best rank k approximation to A under the Frobenius norm.

Equation (3.325) can be stated another way: the best approximation of A of rank k is

$$\widetilde{A} = \sum_{i=1}^{k} d_i u_i v_i^{\mathrm{T}}. \tag{3.326}$$

This result for the best approximation of a given matrix by one of lower rank was first shown by Eckart and Young (1936). On page 342, we will discuss a bound on the difference between two symmetric matrices whether of the same or different ranks.

In applications, the rank k may be stated a priori or we examine a sequence $k = r - 1,\ r - 2,\ \ldots$, and determine the norm of the best fit at each rank. If s_k is the norm of the best approximating matrix, the sequence $s_{r-1},\ s_{r-2},\ \ldots$ may suggest a value of k for which the reduction in rank is sufficient for our purposes and the loss in closeness of the approximation is not too great. Principal components analysis is a special case of this process (see Sect. 9.4).

Exercises

3.1. Vector spaces of matrices.
   a) Exhibit a basis set for $\mathbb{R}^{n\times m}$ for $n \geq m$.
   b) Does the set of $n \times m$ diagonal matrices form a vector space? (The answer is yes.) Exhibit a basis set for this vector space (assuming $n \geq m$).
   c) Exhibit a basis set for the vector space of $n \times n$ symmetric matrices. (First, of course, we must ask is this a vector space. The answer is yes.)
   d) Show that the cardinality of any basis set for the vector space of $n \times n$ symmetric matrices is $n(n+1)/2$.

3.2. By expanding the expression on the left-hand side, derive equation (3.92) on page 93.

3.3. Show that for any quadratic form $x^{\mathrm{T}}Ax$ there is a symmetric matrix A_s such that $x^{\mathrm{T}}A_sx = x^{\mathrm{T}}Ax$. (The proof is by construction, with $A_s = \frac{1}{2}(A+A^{\mathrm{T}})$, first showing A_s is symmetric and then that $x^{\mathrm{T}}A_sx = x^{\mathrm{T}}Ax$.)

3.4. For $a, b, c \in \mathbb{R}$, give conditions on a, b, and c for the matrix below to be positive definite.

$$\begin{bmatrix} a & b \\ b & c \end{bmatrix}.$$

3.5. Show that the Mahalanobis distance defined in equation (3.95) is a metric (that is, show that it satisfies the properties listed on page 32).

3.6. Verify the relationships for Kronecker products shown in equations (3.97) through (3.103) on page 95.
Hint: Make liberal use of equation (3.96) and previously verified equations.

3.7. Verify the relationship between the vec function and Kronecker multiplication given in equation (3.106), $\mathrm{vec}(ABC) = (C^{\mathrm{T}} \otimes A)\mathrm{vec}(B)$.
Hint: Just determine an expression for the i^{th} term in the vector on either side of the equation.

3.8. Cauchy-Schwarz inequalities for matrices.
   a) Prove the Cauchy-Schwarz inequality for the dot product of matrices ((3.111), page 98), which can also be written as

$$(\mathrm{tr}(A^{\mathrm{T}}B))^2 \leq \mathrm{tr}(A^{\mathrm{T}}A)\mathrm{tr}(B^{\mathrm{T}}B).$$

   b) Prove the Cauchy-Schwarz inequality for determinants of matrices A and B of the same shape:

$$\det(A^{\mathrm{T}}B)^2 \leq \det(A^{\mathrm{T}}A)\det(B^{\mathrm{T}}B).$$

   Under what conditions is equality achieved?
   c) Let A and B be matrices of the same shape, and define

$$p(A, B) = \det(A^{\mathrm{T}}B).$$

   Is $p(\cdot,\cdot)$ an inner product? Why or why not?

3.9. Prove that a square matrix that is either row or column (strictly) diagonally dominant is nonsingular.

3.10. Prove that a positive definite matrix is nonsingular.

3.11. Let A be an $n \times m$ matrix.

a) Under what conditions does A have a Hadamard multiplicative inverse?

b) If A has a Hadamard multiplicative inverse, what is it?

3.12. Bounds on ranks.

a) Show that the bound in inequality (3.128) is sharp by finding two matrices A and B such that $\text{rank}(AB) = \min(\text{rank}(A), \text{rank}(B))$.

b) Show that the bound in inequality (3.170) is sharp by finding an $n \times n$ matrix A and a matrix B with n rows such that $\text{rank}(AB) = \text{rank}(A) + \text{rank}(B) - n$.

c) Show that the bound in inequality (3.129) is sharp by finding two matrices A and B such that $\text{rank}(A + B) = \text{rank}(A) + \text{rank}(B)$.

d) Show that the bound in inequality (3.130) is sharp by finding two matrices A and B such that $\text{rank}(A + B) = |\text{rank}(A) - \text{rank}(B)|$.

3.13. The affine group $\mathcal{A}L(n)$.

a) What is the identity in $\mathcal{A}L(n)$?

b) Let (A, v) be an element of $\mathcal{A}L(n)$. What is the inverse of (A, v)?

3.14. Let A be an $n \times m$ matrix of rank one. Show that A can be written as an outer product

$$A = xy^{\mathrm{T}},$$

where x is some n-vector and y is some m-vector.

3.15. In computational explorations involving matrices, it is often convenient to work with matrices whose elements are integers. If an inverse is involved, it would be nice to know that the elements of the inverse are also integers. Equation (3.172) on page 118 provides us a way of ensuring this.

Show that if the elements of the square matrix A are integers and if $\det(A) = \pm 1$, then (A^{-1} exists and) the elements of A^{-1} are integers.

3.16. Verify the relationships shown in equations (3.176) through (3.183) on page 119. Do this by multiplying the appropriate matrices. For example, equation (3.176) is verified by the equations

$$(I + A^{-1})A(I + A)^{-1} = (A + I)(I + A)^{-1} = (I + A)(I + A)^{-1} = I.$$

Make liberal use of equation (3.173) and previously verified equations. Of course it is much more interesting to derive relationships such as these rather than merely to verify them. The verification, however, often gives an indication of how the relationship would arise naturally.

3.17. Verify equation (3.184).

3.18. In equations (3.176) through (3.183) on page 119, drop the assumptions of nonsingularity of matrices, and assume only that the matrices are

conformable for the operations in the expressions. Replace the inverse with the Moore-Penrose inverse.
Now, determine which of these relationships are true. For those that are true, show that they are (for *general* but conformable matrices). If the relationship does not hold, give a counterexample.

3.19. Prove that if A is nonsingular and lower triangular, then A^{-1} is lower triangular.

3.20. By writing $AA^{-1} = I$, derive the expression for the inverse of a partitioned matrix given in equation (3.190).

3.21. Show that the expression given in equation (3.209) on page 128 is a Moore-Penrose inverse of A. (Show that properties 1 through 4 hold.)

3.22. Properties of Drazin inverses (page 129).

a) Show that a Drazin inverse is a g_1 inverse; that is,

$$AA^{\mathrm{D}}A = A.$$

b) Prove equation (3.212), for any square matrix A and any positive integer k,

$$A^{\mathrm{D}} = A^k(A^{2k+1})^+A^k;$$

inter alia, show that for positive integers j and k,

$$A^j(A^{2j+1})^+A^j = A^k(A^{2k+1})^+A^k.$$

3.23. Show that the expression given for the generalized inverse in equation (3.214) on page 131 is correct.

3.24. In computational explorations involving matrices, it is often convenient to work with matrices whose elements are integers. If eigenvalues are involved, it would be nice to know that the eigenvalues are also integers. This is similar in spirit to matrices with integral elements whose inverses also have integral elements, as was the problem considered in Exercise 3.15. Matrices like this also provide convenient test problems for algorithms or sogtware.
The use of the companion matrix (equation (3.225)) gives us a convenient way of obtaining "nice" matrices for numerically exploring properties of eigenvalues. Using other properties of eigenvalues/vectors such as those listed on page 136 and with similarity transforms, we can generate "interesting" matrices that have nice eigenvalues.
For instance, a 3×3 matrix in equation (3.237) was generated by choosing a set of eigenvalues $\{a, 1+\mathrm{i}, 1-\mathrm{i}\}$.
Next, I used the relationship between the eigenvalues of A and $A - dI$, and finally, I squared the matrix, so that the eigenvalues are squared. The resulting spectrum is $\{(a-d)^2, (1-d+\mathrm{i})^2, (1-d-\mathrm{i}\}$. After initializing a and d, the R statements are

```
B <- matrix(c(-d,0,2*a, 1,-d,-2*a+2, 0,1,a+2-d),nrow=3)
A <- B%*%B
```

```
eigen(A)
```

and the Matlab statements are

```
B = [-d, 0, 2*a;  1, -d, -2*a+2;  0, 1, a+2-d];
A = B*B;
eigen(A)
```

Can you tell what values of a and d were used in generating the matrix in equation (3.237) following these steps?

a) Using R, Matlab, or some other system you like, construct *two different* 3×3 matrices whose elements are all integers and whose eigenvalues are $\{7, 5, 3\}$.
b) Determine the six Gershgorin disks for each of your matrices. (Are they the same?)

3.25. Write formal proofs of the properties of eigenvalues/vectors listed on page 136.

3.26. Let A be a square matrix with an eigenvalue c and corresponding eigenvector v. Consider the matrix polynomial in A

$$p(A) = b_0 I + b_1 A + \cdots + b_k A^k.$$

Show that if (c, v) is an eigenpair of A, then $p(c)$, that is,

$$b_0 + b_1 c + \cdots + b_k c^k,$$

is an eigenvalue of $p(A)$ with corresponding eigenvector v. (Technically, the symbol $p(\cdot)$ is overloaded in these two instances.)

3.27. Write formal proofs of the properties of eigenvalues/vectors listed on page 139.

3.28. Prove that for any square matrix, the algebraic multiplicity of a given eigenvalue is at least as great as the geometric multiplicity of that eigenvalue.

3.29. a) Show that the unit vectors are eigenvectors of a diagonal matrix.
b) Give an example of two similar matrices whose eigenvectors are not the same.
Hint: In equation (3.241), let A be a 2×2 diagonal matrix (so you know its eigenvalues and eigenvectors) with unequal values along the diagonal, and let P be a 2×2 upper triangular matrix, so that you can invert it. Form B and check the eigenvectors.

3.30. Let A be a diagonalizable matrix (not necessarily symmetric) with a spectral decomposition of the form of equation (3.261), $A = \sum_i c_i P_i$. Let c_j be a simple eigenvalue with associated left and right eigenvectors y_j and x_j, respectively. (Note that because A is not symmetric, it may have nonreal eigenvalues and eigenvectors.)

a) Show that $y_j^{\mathrm{H}} x_j \neq 0$.
b) Show that the projection matrix P_j is $x_j y_j^{\mathrm{H}} / y_j^{\mathrm{H}} x_j$.

3.31. If A is nonsingular, show that for any (conformable) vector x

$$(x^{\mathrm{T}} Ax)(x^{\mathrm{T}} A^{-1} x) \geq (x^{\mathrm{T}} x)^2.$$

Hint: Use the square roots and the Cauchy-Schwarz inequality.

3.32. Prove that the induced norm (page 165) is a matrix norm; that is, prove that it satisfies the consistency property.

3.33. Prove the inequality (3.280) for an induced matrix norm on page 165:

$$\|Ax\| \leq \|A\| \, \|x\|.$$

3.34. Prove that, for the square matrix A,

$$\|A\|_2^2 = \rho(A^{\mathrm{T}} A).$$

Hint: Show that $\|A\|_2^2 = \max x^{\mathrm{T}} A^{\mathrm{T}} Ax$ for any normalized vector x.

3.35. Let Q be an $n \times n$ orthogonal matrix, and let x be an n-vector.
a) Prove equation (3.286):

$$\|Qx\|_2 = \|x\|_2.$$

Hint: Write $\|Qx\|_2$ as $\sqrt{(Qx)^{\mathrm{T}} Qx}$.
b) Give examples to show that this does not hold for other norms.

3.36. The triangle inequality for matrix norms: $\|A + B\| \leq \|A\| + \|B\|$.
a) Prove the triangle inequality for the matrix L_1 norm.
b) Prove the triangle inequality for the matrix L_∞ norm.
c) Prove the triangle inequality for the matrix Frobenius norm.

3.37. Prove that the Frobenius norm satisfies the consistency property.

3.38. The Frobenius p norm and the Shatten p norm.
a) Prove that the expression in equation (3.298), the "Frobenius p norm", is indeed a norm.
b) Prove that the expression in equation (3.300), the "Shatten p norm", is indeed a norm.
c) Prove equation (3.301).

3.39. If $\|\cdot\|_a$ and $\|\cdot\|_b$ are matrix norms induced respectively by the vector norms $\|\cdot\|_{v_a}$ and $\|\cdot\|_{v_b}$, prove inequality (3.302); that is, show that there are positive numbers r and s such that, for any A,

$$r\|A\|_b \leq \|A\|_a \leq s\|A\|_b.$$

3.40. Prove inequalities (3.303) through (3.309), and show that the bounds are sharp by exhibiting instances of equality.

3.41. The spectral radius, $\rho(A)$.

a) We have seen by an example that $\rho(A) = 0$ does not imply $A = 0$. What about other properties of a matrix norm? For each, either show that the property holds for the spectral radius or, by means of an example, that it does not hold.

b) Use the outer product of an eigenvector and the one vector to show that for any norm $\|\cdot\|$ and any matrix A, $\rho(A) \leq \|A\|$.

3.42. Show that the function $\|\cdot\|_d$ defined in equation (3.313) is a norm.
Hint: Just verify the properties on page 164 that define a norm.

3.43. Nilpotent matrices.

a) Prove that a nilpotent matrix is singular without using the properties listed on page 174.

b) Prove equations (3.318) through (3.321).

3.44. Prove equations (3.324) and (3.325) under the restriction that $\mathcal{V}(X) \subseteq \mathcal{V}(A)$; that is, where $X = BL$ for a matrix B whose columns span $\mathcal{V}(A)$.

4

Vector/Matrix Derivatives and Integrals

The operations of differentiation and integration of vectors and matrices are logical extensions of the corresponding operations on scalars. There are three objects involved in this operation:

- the variable of the operation;
- the operand (the function being differentiated or integrated); and
- the result of the operation.

In the simplest case, all three of these objects are of the same type, and they are scalars. If either the variable or the operand is a vector or a matrix, however, the structure of the result may be more complicated. This statement will become clearer as we proceed to consider specific cases.

In this chapter, we state or show the form that the derivative takes in terms of simpler derivatives. We state high-level rules for the nature of the differentiation in terms of simple partial differentiation of a scalar with respect to a scalar. We do not consider whether or not the derivatives exist. In general, if the simpler derivatives we write that comprise the more complicated object exist, then the derivative of that more complicated object exists. Once a shape of the derivative is determined, definitions or derivations in ϵ-δ terms could be given, but we will refrain from that kind of formal exercise. The purpose of this chapter is not to develop a calculus for vectors and matrices but rather to consider some cases that find wide applications in statistics. For a more careful treatment of differentiation of vectors and matrices, the reader is referred to Magnus and Neudecker (1999) or to Kollo and von Rosen (2005). Anderson (2003), Muirhead (1982), and Nachbin (1965) also cover various aspects of integration with respect to vector or matrix differentials.

Differentiation is a common operation in solving optimization problems such as least squares or maximum likelihood, and in Sect. 4.4 we illustrate some of the uses of vector/matrix differentiation.

Because integration is a key operation in work with probability distributions (it is the definition of an expected value), in Sect. 4.5, we briefly discuss

J.E. Gentle, *Matrix Algebra*, Springer Texts in Statistics,
DOI 10.1007/978-3-319-64867-5_4

multivariate probability distributions and some expectations of multivariate random variables.

4.1 Functions of Vectors and Matrices

There are various types of functions of vectors and matrices. Some functions of matrices, such as the trace, the determinant, and the different norms are all functions from $\mathbb{R}^{n\times n}$ into $\mathbb{R}$ or into the nonnegative reals, $\bar{\mathbb{R}}_+$. Other functions of vectors and matrices are just those defined by elementwise operations, such as $\sin(A) = (\sin(a_{ij}))$ and $\exp(A) = (\exp(a_{ij}))$. That is, a standard function that maps $\mathbb{R}$ to $\mathbb{R}$, when evaluated on $\mathbb{R}^{n\times m}$ maps to $\mathbb{R}^{n\times m}$ in a very direct way. The derivative of a function defined in this way is simply the derivative of the individual elements if they all exist. Most of the mathematical software, such as R, Matlab, and modern Fortran, interpret builtin functions this way when a matrix if given as the argument.

For a diagonalizable matrix, another way of defining a function of a matrix that corresponds to some function of a scalar, $f(x)$, is to use the diagonal factorization as in equation (3.250) on page 152:

$$f(A) = V\mathrm{diag}((f(c_1),\ldots,f(c_n)))V^{-1},$$

if $f(\cdot)$ is defined for each eigenvalue c_i.

If a function of a scalar $f(x)$ has a convergent series expansion, another way of defining a corresponding function of a matrix is to use the series expansion. For example, using the power series expansion of $\mathrm{e}^x = \sum_{k=0}^{\infty} \frac{x^k}{k!}$, we defined the *matrix exponential* for the square matrix A in equation (3.251) as the matrix

$$\mathrm{e}^A = \sum_{k=0}^{\infty} \frac{A^k}{k!}.$$

(Both R and Matlab have a function `expm` for the matrix exponential.) Of course, e^A may also be interpreted as the matrix with elements $(\mathrm{e}^{a_{ij}})$, as mentioned above. The derivative of a function defined as a convergent series is simply the series of the derivatives of the individual terms, if they all exist and if they converge. (Otherwise, it may not be defined.)

The form of the vector or matrix function obviously determines how we must interpret a derivative or an integral involving the function. If differentiation or integration can be done term by term, the nature of the terms must be taken into account.

An extensive coverage of matrix functions is given in Higham (2008).

4.2 Basics of Differentiation

It is useful to recall the heuristic interpretation of a derivative. A derivative of a function is the infinitesimal rate of change of the function with respect

to the variable with which the differentiation is taken. If both the function and the variable are scalars, this interpretation is unambiguous. If, however, the operand of the differentiation, Φ, is a more complicated function, say a vector or a matrix, and/or the variable of the differentiation, Ξ, is a more complicated object, the changes are more difficult to measure. Change in the value both of the function,

$$\delta\Phi = \Phi_{\text{new}} - \Phi_{\text{old}},$$

and of the variable,

$$\delta\Xi = \Xi_{\text{new}} - \Xi_{\text{old}},$$

could be measured in various ways; for example, by using various norms, as discussed in Sects. 2.1.5 and 3.9. (Note that the subtraction is not necessarily ordinary scalar subtraction.)

Furthermore, we cannot just divide the function values by $\delta\Xi$. We do not have a definition for division by that kind of object. We need a mapping, possibly a norm, that assigns a positive real number to $\delta\Xi$. We can define the change in the function value as just the simple difference of the function evaluated at the two points. This yields

$$\lim_{\|\delta\Xi\|\to 0} \frac{\Phi(\Xi + \delta\Xi) - \Phi(\Xi)}{\|\delta\Xi\|}. \tag{4.1}$$

So long as we remember the complexity of $\delta\Xi$, however, we can adopt a simpler approach. Since for both vectors and matrices, we have definitions of multiplication by a scalar and of addition, we can simplify the limit in the usual definition of a derivative, $\delta\Xi \to 0$. Instead of using $\delta\Xi$ as the element of change, we will use $t\Upsilon$, where t is a scalar and Υ is an element to be added to Ξ. The limit then will be taken in terms of $t \to 0$. This leads to

$$\lim_{t\to 0} \frac{\Phi(\Xi + t\Upsilon) - \Phi(\Xi)}{t} \tag{4.2}$$

as a formula for the derivative of Φ with respect to Ξ.

The expression (4.2) may be a useful formula for evaluating a derivative, but we must remember that it is not the derivative. The type of object of this formula is the same as the type of object of the function, Φ; it does not accommodate the type of object of the argument, Ξ, unless Ξ is a scalar. As we will see below, for example, if Ξ is a vector and Φ is a scalar, the derivative must be a vector, yet in that case the expression (4.2) is a scalar.

The expression (4.1) is rarely directly useful in evaluating a derivative, but it serves to remind us of both the generality and the complexity of the concept. Both Φ and its arguments could be functions, for example. (In functional analysis, various kinds of functional derivatives are defined, such as a Gâteaux derivative. These derivatives find applications in developing robust statistical methods; see Shao 2003, for example.) In this chapter, we are interested in the combinations of three possibilities for Φ, namely scalar, vector, and matrix, and the same three possibilities for Ξ and Υ.

4.2.1 Continuity

For the derivative of a function to exist at a point, the function must be continuous at that point. A function of a vector or a matrix is continuous if it is continuous for each element of the vector or matrix. Just as scalar sums and products are continuous, vector/matrix sums and all of the types of vector/matrix products we have discussed are continuous. A continuous function of a continuous function is continuous.

Many of the vector/matrix functions we have discussed are clearly continuous. For example, the L_p vector norms in equation (2.33) are continuous. The determinant of a matrix is continuous, as we see from the definition of the determinant and the fact that sums and scalar products are continuous. The fact that the determinant is a continuous function immediately yields the result that cofactors and hence the adjugate are continuous. From the relationship between an inverse and the adjugate (equation (3.172)), we see that the inverse is a continuous function.

One important function that is not continuous is the rank of a matrix.

4.2.2 Notation and Properties

We write the differential operator with respect to the dummy variable x as $\partial/\partial x$ or $\partial/\partial x^{\mathrm{T}}$. We usually denote differentiation using the symbol for "partial" differentiation, ∂, whether the operator is written ∂x_i for differentiation with respect to a specific scalar variable or ∂x for differentiation with respect to the array x that contains all of the individual elements. Sometimes, however, if the differentiation is being taken with respect to the whole array (the vector or the matrix), we use the notation $\mathrm{d}/\mathrm{d}x$.

The operand of the differential operator $\partial/\partial x$ is a function of x. (If it is not a function of x—that is, if it is a constant function with respect to x—then the operator evaluates to 0.) The result of the operation, written $\partial f/\partial x$, is also a function of x, with the same domain as f, and we sometimes write $\partial f(x)/\partial x$ to emphasize this fact. The value of this function at the fixed point x_0 is written as $\partial f(x_0)/\partial x$. (The derivative of the constant $f(x_0)$ is identically 0, but it is not necessary to write $\partial f(x)/\partial x|_{x_0}$ because $\partial f(x_0)/\partial x$ is interpreted as the value of the function $\partial f(x)/\partial x$ at the fixed point x_0.)

If $\partial/\partial x$ operates on f, and $f : S \to T$, then $\partial/\partial x : S \to U$. The nature of S, or more directly the nature of x, whether it is a scalar, a vector, or a matrix, and the nature of T determine the structure of the result U. For example, if x is an n-vector and $f(x) = x^{\mathrm{T}}x$, then

$$f : \mathbb{R}^n \to \mathbb{R}$$

and

$$\partial f/\partial x : \mathbb{R}^n \to \mathbb{R}^n,$$

as we will see. The outer product, $h(x) = xx^{\mathrm{T}}$, is a mapping to a higher rank array, but the derivative of the outer product is a mapping to an array of the same rank; that is,

$$h : \mathbb{R}^n \to \mathbb{R}^{n \times n}$$

and

$$\partial h / \partial x : \mathbb{R}^n \to \mathbb{R}^n.$$

(Note that "rank" here means the number of dimensions; see page 5.)

As another example, consider $g(\cdot) = \det(\cdot)$, so

$$g : \mathbb{R}^{n \times n} \to \mathbb{R}.$$

In this case,

$$\partial g / \partial X : \mathbb{R}^{n \times n} \to \mathbb{R}^{n \times n};$$

that is, the derivative of the determinant of a square matrix is a square matrix, as we will see later.

Higher-order differentiation is a composition of the $\partial / \partial x$ operator with itself or of the $\partial / \partial x$ operator with the $\partial / \partial x^{\mathrm{T}}$ operator. For example, consider the familiar function in linear least squares

$$f(b) = (y - Xb)^{\mathrm{T}}(y - Xb).$$

This is a mapping from $\mathbb{R}^m$ to $\mathbb{R}$. The first derivative with respect to the m-vector b is a mapping from $\mathbb{R}^m$ to $\mathbb{R}^m$, namely $2X^{\mathrm{T}}Xb - 2X^{\mathrm{T}}y$. The second derivative with respect to b^{T} is a mapping from $\mathbb{R}^m$ to $\mathbb{R}^{m \times m}$, namely, $2X^{\mathrm{T}}X$. (Many readers will already be familiar with these facts. We will discuss the general case of differentiation with respect to a vector in Sect. 4.3.2.)

We see from expression (4.1) that differentiation is a linear operator; that is, if $\mathcal{D}(\Phi)$ represents the operation defined in expression (4.1), Ψ is another function in the class of functions over which $\mathcal{D}$ is defined, and a is a scalar that does not depend on the variable Ξ, then $\mathcal{D}(a\Phi + \Psi) = a\mathcal{D}(\Phi) + \mathcal{D}(\Psi)$. This yields the familiar rules of differential calculus for derivatives of sums or constant scalar products. Other usual rules of differential calculus apply, such as for differentiation of products and composition (the chain rule). We can use expression (4.2) to work these out. For example, for the derivative of the product $\Phi\Psi$, after some rewriting of terms, we have the numerator

$$\begin{aligned}
&\Phi(\Xi)\big(\Psi(\Xi + t\Upsilon) - \Psi(\Xi)\big) \\
&+ \Psi(\Xi)\big(\Phi(\Xi + t\Upsilon) - \Phi(\Xi)\big) \\
&+ \big(\Phi(\Xi + t\Upsilon) - \Phi(\Xi)\big)\big(\Psi(\Xi + t\Upsilon) - \Psi(\Xi)\big).
\end{aligned}$$

Now, dividing by t and taking the limit, assuming that as

$$t \to 0,$$

$$(\Phi(\Xi + t\Upsilon) - \Phi(\Xi)) \to 0,$$

we have

$$\mathcal{D}(\Phi\Psi) = \mathcal{D}(\Phi)\Psi + \Phi\mathcal{D}(\Psi), \tag{4.3}$$

where again $\mathcal{D}$ represents the differentiation operation.

4.2.3 Differentials

For a differentiable scalar function of a scalar variable, $f(x)$, the *differential of f at c with increment u* is $u\mathrm{d}f/\mathrm{d}x|_c$. This is the linear term in a truncated Taylor series expansion:

$$f(c+u) = f(c) + u\frac{\mathrm{d}}{\mathrm{d}x}f(c) + r(c,u). \tag{4.4}$$

Technically, the differential is a function of both x and u, but the notation $\mathrm{d}f$ is used in a generic sense to mean the differential of f. For vector/matrix functions of vector/matrix variables, the differential is defined in a similar way. The structure of the differential is the same as that of the function; that is, for example, the differential of a matrix-valued function is a matrix.

4.3 Types of Differentiation

In the following sections we consider differentiation with respect to different types of objects first, and then we consider differentiation of different types of objects.

4.3.1 Differentiation with Respect to a Scalar

Differentiation of a structure (vector or matrix, for example) with respect to a scalar is quite simple; it just yields the ordinary derivative of each element of the structure in the same structure. Thus, the derivative of a vector or a matrix with respect to a scalar variable is a vector or a matrix, respectively, of the derivatives of the individual elements.

Differentiation with respect to a vector or matrix, which we will consider below, is often best approached by considering differentiation with respect to the individual elements of the vector or matrix, that is, with respect to scalars.

4.3.1.1 Derivatives of Vectors with Respect to Scalars

The derivative of the vector $y(x) = (y_1, \ldots, y_n)$ with respect to the scalar x is the vector

$$\partial y/\partial x = (\partial y_1/\partial x, \ldots, \partial y_n/\partial x). \tag{4.5}$$

The second or higher derivative of a vector with respect to a scalar is likewise a vector of the derivatives of the individual elements; that is, it is an array of higher rank.

4.3.1.2 Derivatives of Matrices with Respect to Scalars

The derivative of the matrix $Y(x) = (y_{ij})$ with respect to the scalar x is the matrix

$$\partial Y(x)/\partial x = (\partial y_{ij}/\partial x). \tag{4.6}$$

The second or higher derivative of a matrix with respect to a scalar is likewise a matrix of the derivatives of the individual elements.

4.3.1.3 Derivatives of Functions with Respect to Scalars

Differentiation of a function of a vector or matrix that is linear in the elements of the vector or matrix involves just the differentiation of the elements, followed by application of the function. For example, the derivative of a trace of a matrix is just the trace of the derivative of the matrix. On the other hand, the derivative of the determinant of a matrix is not the determinant of the derivative of the matrix (see below).

4.3.1.4 Higher-Order Derivatives with Respect to Scalars

Because differentiation with respect to a scalar does not change the rank of the object ("rank" here means rank of an array or "shape"), higher-order derivatives $\partial^k/\partial x^k$ with respect to scalars are merely objects of the same rank whose elements are the higher-order derivatives of the individual elements.

4.3.2 Differentiation with Respect to a Vector

Differentiation of a given object with respect to an n-vector yields a vector for each element of the given object. The basic expression for the derivative, from formula (4.2), is

$$\lim_{t \to 0} \frac{\Phi(x + ty) - \Phi(x)}{t} \tag{4.7}$$

for an arbitrary conformable vector y. The arbitrary y indicates that the derivative is omnidirectional; it is the rate of change of a function of the vector in any direction.

4.3.2.1 Derivatives of Scalars with Respect to Vectors: The Gradient

The derivative of a scalar-valued function with respect to a vector is a vector of the partial derivatives of the function with respect to the elements of the vector. If $f(x)$ is a scalar function of the vector $x = (x_1, \ldots, x_n)$,

$$\frac{\partial f}{\partial x} = \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n} \right), \tag{4.8}$$

if those derivatives exist. This vector is called the *gradient* of the scalar-valued function, and is sometimes denoted by $\mathrm{g}_f(x)$ or $\nabla f(x)$, or sometimes just g_f or ∇f:

$$\mathrm{g}_f = \nabla f = \frac{\partial f}{\partial x}. \tag{4.9}$$

The notation g_f or ∇f implies differentiation with respect to "all" arguments of f, hence, if f is a scalar-valued function of a vector argument, they represent a vector. The symbol ∇ with this interpretation is called "nabla".

This derivative is useful in finding the maximum or minimum of a function. Such applications arise throughout statistical and numerical analysis. In Sect. 6.3.2, we will discuss a method of solving linear systems of equations by formulating the problem as a minimization problem.

Inner products, bilinear forms, norms, and variances are interesting scalar-valued functions of vectors. In these cases, the function Φ in equation (4.7) is scalar-valued and the numerator is merely $\Phi(x+ty)-\Phi(x)$. Consider, for example, the quadratic form $x^{\mathrm{T}}Ax$. Using equation (4.7) to evaluate $\partial x^{\mathrm{T}}Ax/\partial x$, we have

$$\begin{aligned}
&\lim_{t\to 0}\frac{(x+ty)^{\mathrm{T}}A(x+ty)-x^{\mathrm{T}}Ax}{t}\\
&=\lim_{t\to 0}\frac{x^{\mathrm{T}}Ax+ty^{\mathrm{T}}Ax+ty^{\mathrm{T}}A^{\mathrm{T}}x+t^2y^{\mathrm{T}}Ay-x^{\mathrm{T}}Ax}{t} \qquad (4.10)\\
&=y^{\mathrm{T}}(A+A^{\mathrm{T}})x,
\end{aligned}$$

for an arbitrary y (that is, "in any direction"), and so $\partial x^{\mathrm{T}}Ax/\partial x = (A+A^{\mathrm{T}})x$.

This immediately yields the derivative of the square of the Euclidean norm of a vector, $\|x\|_2^2$, and the derivative of the Euclidean norm itself by using the chain rule. Other L_p vector norms may not be differentiable everywhere because of the presence of the absolute value in their definitions. The fact that the Euclidean norm is differentiable everywhere is one of its most important properties.

The derivative of the quadratic form also immediately yields the derivative of the variance. The derivative of the correlation, however, is slightly more difficult because it is a ratio (see Exercise 4.2).

The operator $\partial/\partial x^{\mathrm{T}}$ applied to the scalar function f results in $\mathrm{g}_f^{\mathrm{T}}$.

The second derivative of a scalar-valued function with respect to a vector is a derivative of the first derivative, which is a vector. We will now consider derivatives of vectors with respect to vectors.

4.3.2.2 Derivatives of Vectors with Respect to Vectors: The Jacobian

The derivative of an m-vector-valued function of an n-vector argument consists of nm scalar derivatives. These derivatives could be put into various

structures. Two obvious structures are an $n \times m$ matrix and an $m \times n$ matrix. For a function $f : S \subseteq \mathbb{R}^n \to \mathbb{R}^m$, we define $\partial f^{\mathrm{T}}/\partial x$ to be the $n \times m$ matrix, which is the natural extension of $\partial/\partial x$ applied to a scalar function, and $\partial f/\partial x^{\mathrm{T}}$ to be its transpose, the $m \times n$ matrix. Although the notation $\partial f^{\mathrm{T}}/\partial x$ is more precise because it indicates that the elements of f correspond to the columns of the result, we often drop the transpose in the notation. We have

$$\begin{aligned}\frac{\partial f}{\partial x} &= \frac{\partial f^{\mathrm{T}}}{\partial x} \quad \text{by convention} \\ &= \left[\frac{\partial f_1}{\partial x} \cdots \frac{\partial f_m}{\partial x}\right] \\ &= \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_2}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_1} \\ \frac{\partial f_1}{\partial x_2} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_2} \\ & & \cdots & \\ \frac{\partial f_1}{\partial x_n} & \frac{\partial f_2}{\partial x_n} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}\end{aligned} \tag{4.11}$$

if those derivatives exist. This derivative is called the *matrix gradient* and is denoted by G_f or ∇f for the vector-valued function f. (Note that the nabla symbol can denote either a vector or a matrix, depending on whether the function being differentiated is scalar-valued or vector-valued.)

The $m \times n$ matrix $\partial f/\partial x^{\mathrm{T}} = (\nabla f)^{\mathrm{T}}$ is called the *Jacobian* of f and is denoted by J_f:

$$\mathrm{J}_f = \mathrm{G}_f^{\mathrm{T}} = (\nabla f)^{\mathrm{T}}. \tag{4.12}$$

The absolute value of the determinant of the Jacobian appears in integrals involving a change of variables. (Occasionally, the term "Jacobian" is used to refer to the absolute value of the determinant rather than to the matrix itself.)

To emphasize that the quantities are functions of x, we sometimes write $\partial f(x)/\partial x$, $\mathrm{J}_f(x)$, $\mathrm{G}_f(x)$, or $\nabla f(x)$.

4.3.2.3 Derivatives of Vectors with Respect to Vectors in $\mathbb{R}^3$: The Divergence and the Curl

In some applications in which the order of f and x are the same, the trace of the matrix gradient is of interest. This is particularly the case in $\mathbb{R}^3$, where, for example, the three elements of f may represent the expansion of a gas in the three directions of an orthogonal Cartesian coordinate system. The change in the density of the gas at the point x is the sum of the changes of the density in each of the orthogonal directions:

$$\frac{\partial f_1}{\partial x_1}(x) + \frac{\partial f_2}{\partial x_2}(x) + \frac{\partial f_3}{\partial x_3}(x) = \mathrm{tr}(\nabla f(x)).$$

This type of expression arises so often in physical applications that it is given a name, "divergence", and a special symbol for the operation on the function f, div. Another expression for the divergence that is common in physics arises from the interpretation of ∇ as an ordered list of operators, $(\partial/\partial x_1, \partial/\partial x_2, \ldots)$. The nabla symbol with this interpretation is called "del". The application of ∇ in this form to the function f is analogous to the dot product of ∇ and f; hence, it is often written as $\nabla \cdot f$; that is, in different notations, we have

$$\nabla \cdot f = \mathrm{div}(f) = \mathrm{tr}(\nabla f).$$

In physical applications, when the vector function f above is actually the gradient of a scalar function, the divergence of f determines some interesting characteristics of the underlying scalar function. For example, if $u(x)$ is the temperature at x, then the gradient ∇u is the change in temperature measured in each of the components of x. The divergence of this gradient can be interpreted as the total change in the heat. In a closed system, this should be zero, and if the total heat is changing, this should be proportional to the change in heat. (This fact is called the "heat equation".) There are similar situations in which the divergence of the gradient of a scalar function is of interest.

The operator representing the operation just described is called the Laplace operator or Laplacian operator. In the notation above, it can be represented as $\nabla \cdot \nabla u$, and so the operator itself is sometimes represented as ∇^2, which is called "del-squared"; however, it is sometimes also represented as Δ. We also use the symbol ∇^2 to denote the Hessian of a scalar function. In that context it is called "nabla-squared". The Laplace operator as defined here is the trace of the Hessian matrix as defined in equation (4.16).

There are many important physical applications and special operations for vectors in $\mathbb{R}^3$, as we have discussed in Sect. 2.2.9, beginning on page 46. In particular, we defined the cross product of the vectors x and y, as

$$x \times y = (x_2y_3 - x_3y_2,\ x_3y_1 - x_1y_3,\ x_1y_2 - x_2y_1).$$

Now, suppose that y is a 3-vector function of x, say, $f(x)$, and as we did with the del operator $\nabla = (\partial/\partial x_1, \partial/\partial x_2, \partial/\partial x_3)$ above, let us consider a formal substitution:

$$\nabla \times f = (\partial f_3/\partial x_2 - \partial f_2/\partial x_3, \quad \partial f_1/\partial x_3 - \partial f_3/\partial x_1, \quad \partial f_2/\partial x_1 - \partial f_1/\partial x_2).$$

We call this the curl of f with respect to x and write

$$\mathrm{curl}(f) = \nabla \times f.$$

The uses of the curl in both physical and geometrical applications are far-reaching, but we will not go into them here. An interesting mathematical fact is that the curl of the gradient of a twice-differential scalar function is 0:

$$\mathrm{curl}(\mathrm{g}_f) = 0, \tag{4.13}$$

where f is a twice-differential scalar function. You are asked to show this in Exercise 4.3.

In $\mathbb{R}^3$, for a vector function f that is twice differential, there is an interesting relationship between the divergence and the curl:

$$\text{div}(\text{curl}(f)) = 0 \tag{4.14}$$

which you are asked to show in Exercise 4.4.

4.3.2.4 Derivatives of Matrices with Respect to Vectors

The derivative of a matrix with respect to a vector is a three-dimensional object that results from applying equation (4.8) to each of the elements of the matrix. For this reason, it is simpler to consider only the partial derivatives of the matrix Y with respect to the individual elements of the vector x; that is, $\partial Y/\partial x_i$. The expressions involving the partial derivatives can be thought of as defining one two-dimensional layer of a three-dimensional object.

Using the rules for differentiation of powers that result directly from the definitions, we can write the partial derivatives of the inverse of the matrix Y as

$$\frac{\partial}{\partial x}Y^{-1} = -Y^{-1}\left(\frac{\partial}{\partial x}Y\right)Y^{-1} \tag{4.15}$$

(see Exercise 4.5).

Beyond the basics of differentiation of constant multiples or powers of a variable, the two most important properties of derivatives of expressions are the linearity of the operation and the chaining of the operation. These yield rules that correspond to the familiar rules of the differential calculus. A simple result of the linearity of the operation is the rule for differentiation of the trace:

$$\frac{\partial}{\partial x}\text{tr}(Y) = \text{tr}\left(\frac{\partial}{\partial x}Y\right).$$

4.3.2.5 Higher-Order Derivatives with Respect to Vectors: The Hessian

Higher-order derivatives are derivatives of lower-order derivatives. As we have seen, a derivative of a given function with respect to a vector is a more complicated object than the original function. The simplest higher-order derivative with respect to a vector is the second-order derivative of a scalar-valued function. Higher-order derivatives may become uselessly complicated.

In accordance with the meaning of derivatives of vectors with respect to vectors, the second derivative of a scalar-valued function with respect to a vector is a matrix of the partial derivatives of the function with respect to the elements of the vector. This matrix is called the *Hessian*, and is denoted by

H_f or sometimes by $\nabla\nabla f$ or $\nabla^2 f$ ("nabla-squared", not to confused with the Laplace operator, "del-squared"):

$$\mathrm{H}_f = \frac{\partial^2 f}{\partial x \partial x^{\mathrm{T}}} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_m} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_m} \\ & & \cdots & \\ \frac{\partial^2 f}{\partial x_m \partial x_1} & \frac{\partial^2 f}{\partial x_m \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_m^2} \end{bmatrix}. \tag{4.16}$$

To emphasize that the Hessian is a function of x, we sometimes write $\mathrm{H}_f(x)$ or $\nabla\nabla f(x)$ or $\nabla^2 f(x)$.

4.3.2.6 Summary of Derivatives with Respect to Vectors

As we have seen, the derivatives of functions are complicated by the problem of measuring the change in the function, but often the derivatives of functions with respect to a vector can be determined by using familiar scalar differentiation. In general, we see that

- the derivative of a scalar (a quadratic form) with respect to a vector is a vector and
- the derivative of a vector with respect to a vector is a matrix.

Table 4.1 lists formulas for the vector derivatives of some common expressions. The derivative $\partial f/\partial x^{\mathrm{T}}$ is the transpose of $\partial f/\partial x$.

As noted above, some authors express derivatives in different structures. The values of the individual elements are the same, but their organization in a vector or matrix may be different.

4.3.3 Differentiation with Respect to a Matrix

The derivative of a function with respect to a matrix is a matrix with the same shape consisting of the partial derivatives of the function with respect to the elements of the matrix. This rule defines what we mean by differentiation with respect to a matrix.

For scalar-valued functions, this rule is fairly simple:

$$\frac{\partial f(X)}{\partial X} = \left(\frac{\partial f(X)}{\partial x_{ij}}\right). \tag{4.17}$$

For example, consider the trace. If X is a square matrix and we apply this rule to evaluate $\partial \operatorname{tr}(X)/\partial X$, we get the identity matrix, where the nonzero elements arise only when $j = i$ in $\partial(\sum x_{ii})/\partial x_{ij}$. If AX is a square matrix, we have for the (i, j) term in $\partial \operatorname{tr}(AX)/\partial X$, $\partial \sum_i \sum_k a_{ik} x_{ki}/\partial x_{ij} = a_{ji}$, and so $\partial \operatorname{tr}(AX)/\partial X = A^{\mathrm{T}}$, and likewise, inspecting $\partial \sum_i \sum_k x_{ik} x_{ki}/\partial x_{ij}$, we get

Table 4.1. Formulas for some vector derivatives

$f(x)$	$\partial f/\partial x$
ax	aI
$b^{\mathrm{T}}x$	b
$x^{\mathrm{T}}b$	b
$x^{\mathrm{T}}x$	$I \otimes x + x \otimes I$
xx^{T}	$2x^{\mathrm{T}}$
$b^{\mathrm{T}}Ax$	$A^{\mathrm{T}}b$
$x^{\mathrm{T}}Ab$	Ab
$x^{\mathrm{T}}Ax$	$(A + A^{\mathrm{T}})x$
	$2Ax$, if A is symmetric
$\exp(-\frac{1}{2}x^{\mathrm{T}}Ax)$	$-\exp(-\frac{1}{2}x^{\mathrm{T}}Ax)Ax$, if A is symmetric
$\|x\|_2^2$	$2x$
$\mathrm{V}(x)$	$2x/(n-1)$

In this table, x is an n-vector, a is a constant scalar, b is a constant conformable vector, and A is a constant conformable matrix.

$\partial\,\mathrm{tr}(X^{\mathrm{T}}X)/\partial X = 2X^{\mathrm{T}}$. Likewise for the scalar-valued $a^{\mathrm{T}}Xb$, where a and b are conformable constant vectors, for $\partial \sum_m (\sum_k a_k x_{km}) b_m / \partial x_{ij} = a_i b_j$, so $\partial a^{\mathrm{T}}Xb/\partial X = ab^{\mathrm{T}}$.

Now consider $\partial \det(X)/\partial X$. Using an expansion in cofactors (equation (3.30) or (3.31)), the only term in $\det(X)$ that involves x_{ij} is $x_{ij}(-1)^{i+j}\det(X_{-(i)(j)})$, and the cofactor $(x_{(ij)}) = (-1)^{i+j}\det(X_{-(i)(j)})$ does not involve x_{ij}. Hence, $\partial \det(X)/\partial x_{ij} = (x_{(ij)})$, and so $\partial \det(X)/\partial X = (\mathrm{adj}(X))^{\mathrm{T}}$ from equation (3.33). Using equation (3.172), we can write this as $\partial \det(X)/\partial X = \det(X)X^{-\mathrm{T}}$.

By the definition of differentiation with respect to a matrix X, we see that the derivative $\partial f/\partial X^{\mathrm{T}}$ is the transpose of $\partial f/\partial X$.

The chain rule can be used to evaluate $\partial \log(\det(X))/\partial X$.

Applying the rule stated at the beginning of this section, we see that the derivative of a matrix Y with respect to the matrix X is

$$\frac{\mathrm{d}Y}{\mathrm{d}X} = \frac{\mathrm{d}(\mathrm{vec}(Y))}{\mathrm{d}(\mathrm{vec}(X))}, \tag{4.18}$$

or

$$\frac{\mathrm{d}Y}{\mathrm{d}X} = Y \otimes \frac{\mathrm{d}}{\mathrm{d}X}, \tag{4.19}$$

especially in statistical applications. (Recall the comment above about the use by different authors of different structures for derivatives.)

Tables 4.2 and 4.3 list some formulas for the matrix derivatives of some common expressions. The derivatives shown in those tables can be obtained by using equation (4.17) or (4.18), possibly also using the chain rule.

Table 4.2. Formulas for some matrix derivatives

General X

$f(X)$	$\partial f/\partial X$
$a^{\mathrm{T}}Xb$	ab^{T}
$\mathrm{tr}(AX)$	A^{T}
$\mathrm{tr}(X^{\mathrm{T}}X)$	$2X^{\mathrm{T}}$
BX	$I_n \otimes B$
XC	$C^{\mathrm{T}} \otimes I_m$
BXC	$C^{\mathrm{T}} \otimes B$

In this table, X is an $n \times m$ matrix, a is a constant n-vector, b is a constant m-vector, A is a constant $m \times n$ matrix, B is a constant $p \times n$ matrix, and C is a constant $m \times q$ matrix.

Table 4.3. Formulas for some matrix derivatives

Square and possibly invertible X

$f(X)$	$\partial f/\partial X$
$\mathrm{tr}(X)$	I_n
$\mathrm{tr}(X^k)$	kX^{k-1}
$\mathrm{tr}(BX^{-1}C)$	$-(X^{-1}CBX^{-1})^{\mathrm{T}}$
$\det(X)$	$\det(X)X^{-\mathrm{T}}$
$\log(\det(X))$	$X^{-\mathrm{T}}$
$(\det(X))^k$	$k(\det(X))^k X^{-\mathrm{T}}$
$BX^{-1}C$	$-(X^{-1}C)^{\mathrm{T}} \otimes BX^{-1}$

In this table, X is an $n \times n$ matrix, B is a constant $p \times n$ matrix, and C is a constant $n \times q$ matrix.

4.4 Optimization of Scalar-Valued Functions

A common problem in statistics, as well as in science generally, is to find the minimum or maximum of some scalar-valued function. In statistical applications, for example, we may seek to fit a model to data by determining the values of model parameters that yield the *minimum* of the sum of squares of residuals. Alternatively, we may seek to determine the values of the parameters in a likelihood function that yield the *maximum* of the likelihood function for a given set of data.

In optimization problems, we refer to the function of interest as the *objective function*. In the least squares fitting problem, the objective function is the sum of squares, which, for given data, is a function of the parameters in the model. In the maximum likelihood estimation problem, the objective

function is the likelihood function, which, for given data, is a function of the parameters in the probability model.

Because the function may have many ups and downs, we often use the phrases *local minimum* and *global minimum* (or global maximum or global optimum), with obvious meanings. Our discussion focuses on local optima, and we will not consider the problem of finding a global optimum of a function with multiple local optima.

Since maximizing a scalar function $f(x)$ is equivalent to minimizing its negative, $-f(x)$, we can always just consider the basic problem to be minimization of a function. Thus, we can use terminology for the problem of finding a minimum of the objective function, and write the general problem as

$$\min_{x \in D} f(x). \tag{4.20}$$

Often, D in the expression above is just the full domain of f, and often that is $\mathbb{R}^m$ for some m. In the next few subsections, we will consider D to be the full domain of f. Problems in which D is the full domain of f are called *unconstrained optimization* problems. We consider them first, and then in Sect. 4.4.5 we will briefly discuss *constrained optimization* problems.

Methods to approach the optimization problem obviously depend on the nature of the objective function. Whether or not the function is convex and whether or not it is differentiable are important distinctions.

In this section we consider only scalar-valued objective functions that are twice-differentiable. For a scalar-valued objective function $f(x)$ of m variables, we will denote the m-vector of first derivatives, that is, the gradient, as $\mathrm{g}_f(x)$, and the $m \times m$ matrix of second derivatives, the Hessian, as $\mathrm{H}_f(x)$, as above. For a twice-differentiable function, the Hessian is nonnegative (positive) definite everywhere on a given domain if and only if the function is (strictly) convex on that domain.

Because a derivative measures the rate of change of a function, a point at which the first derivative is equal to 0 is a stationary point, which may be a maximum or a minimum of the function. Differentiation is therefore a very useful tool for finding the optima of functions, and so, for a given function $f(x)$, the gradient vector function, $\mathrm{g}_f(x)$, and the Hessian matrix function, $\mathrm{H}_f(x)$, play important roles in optimization methods.

Because except in the very simplest of cases, determining a solution to the equation defining a stationary point, $\mathrm{g}_f(x) = 0$ cannot be done in closed form, the optimization method itself must be iterative, moving through a sequence of points, $x^{(0)}, x^{(1)}, x^{(2)}, \ldots$, that approaches the optimum point arbitrarily closely. At the point $x^{(k)}$, the direction of *steepest descent* is clearly $-\mathrm{g}_f(x^{(k)})$, but because this direction may be continuously changing, the steepest descent direction may not be the best direction in which to seek the next point, $x^{(k+1)}$. In most cases, however, we move in a downward path in the general direction of the gradient. We call any such method a *gradient-descent* method.

4.4.1 Stationary Points of Functions

The optimum point is a stationary point (assuming that it occurs at an interior point of the domain), but a stationary point is not necessarily an optimum point.

The first derivative of the objective function helps only in finding a stationary point. The matrix of second derivatives, the Hessian, provides information about the nature of the stationary point, which may be a local minimum or maximum, a saddlepoint, or only an inflection point.

The so-called second-order optimality conditions are the following (see a general text on optimization, such as Griva, Nash, and Sofer 2009, for the proofs).

- If (but not only if) the stationary point is a local minimum, then the Hessian is nonnegative definite at the stationary point.
- If the Hessian is positive definite at the stationary point, then the stationary point is a local minimum.
- Likewise, if the stationary point is a local maximum, then the Hessian is nonpositive definite, and if the Hessian is negative definite, then the stationary point is a local maximum.
- If the Hessian has both positive and negative eigenvalues at the stationary point, then the stationary point is a saddlepoint.

4.4.2 Newton's Method

We consider a twice-differentiable scalar-valued function of a vector argument, $f(x)$. By a Taylor series expansion about a stationary point x_*, truncated after the second-order term

$$f(x) \approx f(x_*) + (x - x_*)^{\mathrm{T}} \mathrm{g}_f(x_*) + \frac{1}{2}(x - x_*)^{\mathrm{T}} \mathrm{H}_f(x_*)(x - x_*), \tag{4.21}$$

because $\mathrm{g}_f(x_*) = 0$, we have a general method of finding a stationary point for the function $f(\cdot)$, called Newton's method. If x is an m-vector, $\mathrm{g}_f(x)$ is an m-vector and $\mathrm{H}_f(x)$ is an $m \times m$ matrix.

Newton's method is to choose a starting point $x^{(0)}$, then, for $k = 0, 1, \ldots$, to solve the linear systems

$$\mathrm{H}_f(x^{(k)})p^{(k)} = -\mathrm{g}_f(x^{(k)}) \tag{4.22}$$

for $p^{(k)}$, and then to update the point in the domain of $f(\cdot)$ by

$$x^{(k+1)} = x^{(k)} + \alpha^{(k)} p^{(k)}, \tag{4.23}$$

where $\alpha^{(k)}$ is a scalar such that $f(x^{(k+1)}) < f(x^{(k)})$. The two steps are repeated until there is essentially no change from one iteration to the next. These two steps have a very simple form for a function of one variable (see Exercise 4.6a).

We can stop the iterative progression based on either the change in the domain or the amount of change in the value of the function. For specified constants ϵ_1 and ϵ_2, we can stop when

$$\left\| x^{(k+1)} - x^{(k)} \right\| < \epsilon_1 \tag{4.24}$$

or when

$$\left| f\left(x^{(k+1)}\right) - f\left(x^{(k)}\right) \right| < \epsilon_2. \tag{4.25}$$

If $f(\cdot)$ is a quadratic function, the solution using Newton's method is not iterative; it is obtained in one step because equation (4.21) is exact.

4.4.2.1 Quasi-Newton Methods

All gradient-descent methods determine the path $p^{(k)}$ to take toward the next point by a system of equations of the form

$$R^{(k)} p^{(k)} = -\mathrm{g}_f\left(x^{(k)}\right).$$

In the *steepest-descent* method, $R^{(k)}$ is the identity, I, in these equations. For functions with eccentric contours, the steepest-descent method traverses a zigzag path to the minimum. In Newton's method (equation (4.22)), $R^{(k)}$ is the Hessian evaluated at the previous point, $\mathrm{H}_f\left(x^{(k)}\right)$, which results in a more direct path to the minimum than the steepest descent follows.

Aside from the issue of consistency of the resulting equation, a major disadvantage of Newton's method is the computational burden of computing the Hessian, which requires $\mathrm{O}(m^2)$ function evaluations, and solving the system, which requires $\mathrm{O}(m^3)$ arithmetic operations, at each iteration.

Instead of using the Hessian at each iteration, we may use an approximation, $B^{(k)}$. We may choose approximations that are simpler to update and/or that allow the equations for the step to be solved more easily. Methods using such approximations are called *quasi-Newton* methods or *variable metric* methods.

One simple approximation to the Hessian is based on the fact that

$$\mathrm{H}_f\left(x^{(k)}\right)\left(x^{(k)} - x^{(k-1)}\right) \approx \mathrm{g}_f\left(x^{(k)}\right) - \mathrm{g}_f\left(x^{(k-1)}\right);$$

hence, we choose $B^{(k)}$ so that

$$B^{(k)}\left(x^{(k)} - x^{(k-1)}\right) = \mathrm{g}_f\left(x^{(k)}\right) - \mathrm{g}_f\left(x^{(k-1)}\right). \tag{4.26}$$

This is called the *secant condition*.

We express the secant condition as

$$B^{(k)} s^{(k)} = y^{(k)}, \tag{4.27}$$

where

$$s^{(k)} = x^{(k)} - x^{(k-1)}$$

and

$$y^{(k)} = \mathrm{g}_f(x^{(k)}) - \mathrm{g}_f(x^{(k-1)}),$$

as above.

The system of equations in (4.27) does not fully determine $B^{(k)}$ of course. Because $B^{(k)}$ should approximate the Hessian, we may require that it be symmetric and positive definite.

The most common approach in quasi-Newton methods is first to choose a reasonable starting matrix $B^{(0)}$ and then to choose subsequent matrices by additive updates,

$$B^{(k+1)} = B^{(k)} + B_a^{(k)}, \tag{4.28}$$

subject to preservation of symmetry and positive definiteness. An approximate Hessian $B^{(k)}$ may be used for several iterations before it is updated; that is, $B_a^{(k)}$ may be taken as 0 for several successive iterations.

4.4.3 Least Squares

One of the most important applications that involve minimization is the fitting of a model to data. In this problem, we have a function s that relates one variable, say y, to other variables, say the m-vector x. The function involves some unknown parameters, say the d-vector θ, so the model is

$$y \approx s(x;\theta). \tag{4.29}$$

The data consists of n observations on the variables y and x.

Fitting the model is usually done by minimizing some norm of the vector of residuals

$$r_i(\theta) = y_i - s(x_i;\theta). \tag{4.30}$$

The decision variables are the parameters θ. The optimal values of θ, often denoted as $\widehat{\theta}$, are called "estimates".

Because the data are observed and therefore are constants, the residuals are functions of θ only. The vector-valued function $r(\theta)$ maps $\mathbb{R}^d$ into $\mathbb{R}^n$.

The most common norm to minimize to obtain a fit is the L_2 or Euclidean norm. The scalar-valued objective function then is

$$\begin{aligned} f(\theta) &= \sum_{i=1}^{n} \big(y_i - s(x_i;\theta)\big)^2 \\ &= \sum_{i=1}^{n} \big(r_i(\theta)\big)^2 \\ &= \big(r(\theta)\big)^{\mathrm{T}} r(\theta). \end{aligned} \tag{4.31}$$

This problem is called *least squares regression*. If the function s is nonlinear in θ, the functions r_i are also nonlinear in θ, and the problem is called *nonlinear least squares regression*.

4.4.3.1 Linear Least Squares

A very common form of the model (4.29) is the linear regression model

$$y \approx x^{\mathrm{T}}\theta.$$

In this form of the model, we usually use β in place of θ, and we write the model as an equality with an additive "error" term instead of the approximation. In this case also, the order of β is the same as that of x, which we will continue to denote as m.

In statistical applications, we generally have n observations of pairs of y and x. We form an n-vector of the y observations, and an $n \times m$ matrix X of the x observations. Thus, we form the linear model of the data

$$y = X\beta + \epsilon, \tag{4.32}$$

where y is an n-vector, X is an $n \times m$ matrix, β is an m-vector, and ϵ is an n-vector.

For any value of β, say b, we have the residual vector

$$r = y - Xb. \tag{4.33}$$

For a least squares fit of the regression model, we minimize its Euclidean norm,

$$f(b) = r^{\mathrm{T}}r, \tag{4.34}$$

with respect to the variable b. We can solve this optimization problem by taking the derivative of this sum of squares and equating it to zero. Doing this, we get

$$\begin{aligned}\frac{\mathrm{d}(y - Xb)^{\mathrm{T}}(y - Xb)}{\mathrm{d}b} &= \frac{\mathrm{d}(y^{\mathrm{T}}y - 2b^{\mathrm{T}}X^{\mathrm{T}}y + b^{\mathrm{T}}X^{\mathrm{T}}Xb)}{\mathrm{d}b} \\ &= -2X^{\mathrm{T}}y + 2X^{\mathrm{T}}Xb \\ &= 0,\end{aligned}$$

which yields the normal equations

$$X^{\mathrm{T}}Xb = X^{\mathrm{T}}y. \tag{4.35}$$

The solution to the normal equations is a stationary point of the function (4.34). The Hessian of $(y - Xb)^{\mathrm{T}}(y - Xb)$ with respect to b is $2X^{\mathrm{T}}X$ and

$$X^{\mathrm{T}}X \succeq 0.$$

Because the matrix of second derivatives is nonnegative definite, the value of b that solves the system of equations arising from the first derivatives is a local

minimum of equation (4.34). We discuss these equations further in Sects. 6.6 and 9.3.2.

The normal equations also imply one of the most important properties of a linear least squares solution. The residuals are orthogonal to the column space of X:

$$X^{\mathrm{T}}(y - Xb) = 0. \tag{4.36}$$

Notice, of course, that this is just the gradient of the objective function, and the equation just states that the gradient is 0 at the optimum.

4.4.3.2 Nonlinear Least Squares: The Gauss-Newton Method

If the function in the model (4.29) is not linear, the problem may be quite different, both for solving the least squares problem and for issues involved in statistical inference. We will not consider the inference problems here, but rather, just discuss methods for obtaining the least squares fit.

The gradient and the Hessian for a least squares problem have special structures that involve the Jacobian of the residuals, which is a vector function of the parameters. The gradient of $f(\theta)$ is

$$\mathrm{g}_f(\theta) = \big(\mathrm{J}_r(\theta)\big)^{\mathrm{T}} r(\theta).$$

The Jacobian of r is also part of the Hessian of f:

$$\mathrm{H}_f(\theta) = \big(\mathrm{J}_r(\theta)\big)^{\mathrm{T}} \mathrm{J}_r(\theta) + \sum_{i=1}^{n} r_i(\theta) \mathrm{H}_{r_i}(\theta). \tag{4.37}$$

In this maze of notation the reader should pause to remember the shapes of these arrays, and their meanings in the context of fitting a model to data. Notice, in particular, that the dimension of the space of the optimization problem is d, instead of m as in the previous problems. We purposely chose a different letter to represent the dimension so as to emphasize that the decision variables may have a different dimension from that of the independent (observable) variables. The space of an observation has dimension $m + 1$ (the m elements of x, plus the response y); and the space of the observations as points y_i and corresponding model values $s(x_i, \theta)$ has dimension n.

- x_i is an m-vector. In the modeling context, these are the independent variables.
- y is an n-vector, and it together with the n x_i vectors are constants in the optimization problem. In the modeling context, these are observations.
- θ is a d-vector. This is the vector of parameters.
- $r(\cdot)$ is an n-vector. This is the vector of residuals.
- $\mathrm{J}_r(\cdot)$ is an $n \times d$ matrix.
- $\mathrm{H}_{r_i}(\cdot)$ is a $d \times d$ matrix.

- $f(\cdot)$ is a scalar. This is the objective function for the data-fitting criterion.
- $\mathrm{g}_f(\cdot)$ is a d-vector.
- $\mathrm{H}_f(\cdot)$ is a $d \times d$ matrix.

In the vicinity of the solution θ_*, the residuals $r_i(\theta)$ should be small, and $\mathrm{H}_f(\theta)$ may be approximated by neglecting the second term in equation (4.37). Using this approximation and the gradient descent equation, we have

$$\left(\mathrm{J}_r(\theta^{(k)})\right)^{\mathrm{T}} \mathrm{J}_r(\theta^{(k)})\, p^{(k)} \;=\; -\left(\mathrm{J}_r(\theta^{(k)})\right)^{\mathrm{T}} r(\theta^{(k)}). \tag{4.38}$$

It is clear that the solution $p^{(k)}$ is a descent direction and so $\mathrm{g}_f(\theta^{(k)}) \neq 0$, and

$$\begin{aligned}(p^{(k)})^{\mathrm{T}} \mathrm{g}_f(\theta^{(k)}) &= -\left(\left(\mathrm{J}_r(\theta^{(k)})\right)^{\mathrm{T}} p^{(k)}\right)^{\mathrm{T}} \left(\mathrm{J}_r(\theta^{(k)})\right)^{\mathrm{T}} p^{(k)} \\ &< 0.\end{aligned}$$

The update step is determined by a line search in the direction of the solution of equation (4.38):

$$x^{(k+1)} - x^{(k)} \;=\; \alpha^{(k)} p^{(k)}.$$

This method is called the *Gauss-Newton algorithm*. Because years ago the step was often taken simply as $p^{(k)}$, a method that uses a variable step length factor $\alpha^{(k)}$ is sometimes called a "modified Gauss-Newton algorithm". It is the *only* kind to use, so we just call it the "Gauss-Newton algorithm".

If the residuals are small and if the Jacobian is nonsingular, the Gauss-Newton method behaves much like Newton's method near the solution. The major advantage is that second derivatives are not computed.

If the residuals are not small or if $\mathrm{J}_r(\theta^{(k)})$ is poorly conditioned, the Gauss-Newton method can perform very poorly. If $\mathrm{J}_r(\theta^{(k)})$ is not of full rank, we could choose the solution corresponding to the Moore-Penrose inverse:

$$p^{(k)} = \left(\left(\mathrm{J}_r(\theta^{(k)})\right)^{\mathrm{T}}\right)^{+} r(\theta^{(k)}). \tag{4.39}$$

If the matrix is nonsingular, the Moore-Penrose inverse is the usual inverse. (We consider the linear case more thoroughly in Sect. 6.6.3, beginning on page 293, and show that the Moore-Penrose yields the solution that has the shortest Euclidean length.)

In the case of a linear model, the data, consisting of n observations on y and the m-vector x, result in an n-vector of residuals,

$$r = y - Xb,$$

where X is the $n \times m$ matrix whose rows are the observed x^{T}s (and b is used in place of θ). The Gauss-Newton algorithm, which is the same as the ordinary Newton method for this linear least squares problem, yields the solution in one step.

4.4.3.3 Levenberg-Marquardt Method

Another possibility is to add a conditioning matrix to $\left(\mathrm{J}_r(\theta^{(k)})\right)^{\mathrm{T}}\mathrm{J}_r(\theta^{(k)})$ on the left side of equation (4.38). A simple choice is $\tau^{(k)}I_d$ for some scalar $\tau^{(k)}$, and the equation for the update direction becomes

$$\left(\left(\mathrm{J}_r(\theta^{(k)})\right)^{\mathrm{T}}\mathrm{J}_r(\theta^{(k)}) + \tau^{(k)}I_d\right)p^{(k)} = -\left(\mathrm{J}_r(\theta^{(k)})\right)^{\mathrm{T}}r(\theta^{(k)}).$$

A better choice may be a scaling matrix, $S^{(k)}$, that takes into account the variability in the columns of $\mathrm{J}_r(\theta^{(k)})$; hence we have for the update

$$\left(\left(\mathrm{J}_r(\theta^{(k)})\right)^{\mathrm{T}}\mathrm{J}_r(\theta^{(k)}) + \lambda^{(k)}\left(S^{(k)}\right)^{\mathrm{T}}S^{(k)}\right)p^{(k)} = -\left(\mathrm{J}_r(\theta^{(k)})\right)^{\mathrm{T}}r(\theta^{(k)}). \quad (4.40)$$

The basic requirement for the matrix $\left(S^{(k)}\right)^{\mathrm{T}}S^{(k)}$ is that it improve the condition of the coefficient matrix. There are various ways of choosing this matrix. One is to transform the matrix $\left(\mathrm{J}_r(\theta^{(k)})\right)^{\mathrm{T}}\mathrm{J}_r(\theta^{(k)})$ so it has 1's along the diagonal (this is equivalent to forming a correlation matrix from a variance-covariance matrix), and to use the scaling vector to form $S^{(k)}$. The nonnegative factor $\lambda^{(k)}$ can be chosen to control the extent of the adjustment. The sequence $\lambda^{(k)}$ obviously must go to 0 for the solution sequence to converge to the optimum.

Use of an adjustment such as in equation (4.40) is called the *Levenberg-Marquardt method*. This is probably the most widely used method for nonlinear least squares.

The Levenberg-Marquardt adjustment is similar to the regularization done in ridge regression that we will discuss from time to time (see, for example, equation (8.59) on page 364). Just as in ridge regression the computations for equation (4.40) can be performed efficiently by recognizing that the system is the normal equations for the least squares fit of

$$\begin{pmatrix} r(\theta^{(k)}) \\ 0 \end{pmatrix} \approx \begin{bmatrix} \mathrm{J}_r(\theta^{(k)}) \\ \sqrt{\lambda^{(k)}}\left(S^{(k)}\right) \end{bmatrix} p.$$

(See page 432 and Exercise 9.11.)

Equation (4.40), as indeed regularized solutions such as ridge regression generally, can be thought of as a Lagrange multiplier formulation of a constrained problem, as we will discuss in Sect. 4.4.5, beginning on page 208.

4.4.4 Maximum Likelihood

For a sample $y = (y_1, \ldots, y_n)$ from a probability distribution with probability density function $p(\cdot; \theta)$, the *likelihood function* is

$$L(\theta;\, y) = \prod_{i=1}^{n} p(y_i;\, \theta), \tag{4.41}$$

and the *log-likelihood function* is $l(\theta;\, y) = \log(L(\theta;\, y))$. It is often easier to work with the log-likelihood function.

The log-likelihood is an important quantity in information theory and in unbiased estimation. If Y is a random variable having the given probability density function with the r-vector parameter θ, the *Fisher information* matrix that Y contains about θ is the $r \times r$ matrix

$$I(\theta) = \mathrm{Cov}_\theta \left(\frac{\partial l(t, Y)}{\partial t_i}, \frac{\partial l(t, Y)}{\partial t_j} \right), \tag{4.42}$$

where Cov_θ represents the variance-covariance matrix of the functions of Y formed by taking expectations for the given θ. (I use different symbols here because the derivatives are taken with respect to a *variable*, but the θ in Cov_θ cannot be the variable of the differentiation. This distinction is somewhat pedantic, and sometimes I follow the more common practice of using the same symbol in an expression that involves both Cov_θ and $\partial l(\theta, Y)/\partial \theta_i$.)

For example, if the distribution of Y is the d-variate normal distribution with mean d-vector μ and $d \times d$ positive definite variance-covariance matrix Σ, from the multivariate normal PDF, equation (4.73), the likelihood, equation (4.41), is

$$L(\mu, \Sigma;\, y) = \frac{1}{\left((2\pi)^{d/2} |\Sigma|^{1/2}\right)^n} \exp\left(-\frac{1}{2} \sum_{i=1}^{n} (y_i - \mu)^{\mathrm{T}} \Sigma^{-1} (y_i - \mu) \right).$$

(Note that $|\Sigma|^{1/2} = |\Sigma^{\frac{1}{2}}|$, where here I am using $|\cdot|$ to represent the determinant. The square root matrix $\Sigma^{\frac{1}{2}}$ is often useful in transformations of variables.)

Anytime we have a quadratic form that we need to simplify, we should recall equation (3.90): $x^{\mathrm{T}} A x = \mathrm{tr}(A x x^{\mathrm{T}})$. Using this, and because the log-likelihood is easier to work with here, we write

$$l(\mu, \Sigma;\, y) = c - \frac{n}{2} \log |\Sigma| - \frac{1}{2} \mathrm{tr}\left(\Sigma^{-1} \sum_{i=1}^{n} (y_i - \mu)(y_i - \mu)^{\mathrm{T}} \right), \tag{4.43}$$

where we have used c to represent the constant portion. Next, we use the Pythagorean equation (2.70) or equation (3.92) on the outer product to get

$$\begin{aligned} l(\mu, \Sigma;\, y) = c - \frac{n}{2} \log |\Sigma| - \frac{1}{2} \mathrm{tr}\left(\Sigma^{-1} \sum_{i=1}^{n} (y_i - \bar{y})(y_i - \bar{y})^{\mathrm{T}} \right) \\ - \frac{n}{2} \mathrm{tr}\left(\Sigma^{-1} (\bar{y} - \mu)(\bar{y} - \mu)^{\mathrm{T}} \right). \end{aligned} \tag{4.44}$$

In maximum likelihood estimation, we seek the maximum of the likelihood function (4.41) with respect to θ while we consider y to be fixed. If the

maximum occurs within an open set and if the likelihood is differentiable, we might be able to find the maximum likelihood estimates by differentiation. In the log-likelihood for the d-variate normal distribution, we consider the parameters μ and Σ to be variables. To emphasize that perspective, we replace the parameters μ and Σ by the variables $\hat{\mu}$ and $\widehat{\Sigma}$. Now, to determine the maximum, we could take derivatives with respect to $\hat{\mu}$ and $\widehat{\Sigma}$, set them equal to 0, and solve for the maximum likelihood estimates. Some subtle problems arise that depend on the fact that for any constant vector a and scalar b, $\Pr(a^{\mathrm{T}}X = b) = 0$, but we do not interpret the likelihood as a probability. In Exercise 4.7b you are asked to determine the values of $\hat{\mu}$ and $\widehat{\Sigma}$ using properties of traces and positive definite matrices without resorting to differentiation. (This approach does not avoid the subtle problems, however.)

4.4.5 Optimization of Functions with Constraints

Instead of the problem shown in expression (4.20) in which we seek a minimum point anywhere in the domain of the objective function, which often is $\mathbb{R}^m$, we may constrain the search for the minimum point to some subset of the domain. The problem is

$$\min_{x \in C} f(x), \tag{4.45}$$

where $C \subset D$, the domain of f. This is a *constrained optimization* problem. To emphasize the constraints or to make them more explicit, we often write the problem as

$$\begin{aligned} &\min_{x} f(x) \\ &\text{s.t. } x \in C \end{aligned} \tag{4.46}$$

Optimization is a large area. It is our intent here only to cover some of the subareas most closely related to our main subject of matrix algebra. Hence, we will consider only a simple type of equality-constrained optimization problem.

4.4.5.1 Equality-Constrained Linear Least Squares Problems

Instead of the simple least squares problem of determining a value of b that minimizes the sum of squares, we may have some restrictions that b must satisfy; for example, we may have the requirement that the elements of b sum to 1. More generally, consider the least squares problem for the linear model (4.32) with the requirement that b satisfy some set of linear restrictions, $Ab = c$, where A is a full-rank $k \times m$ matrix (with $k \leq m$). (The rank of A must be less than m or else the constraints completely determine the solution to the problem. If the rank of A is less than k, however, some rows of A and some elements of b could be combined into a smaller number of constraints. We can therefore assume A is of full row rank. Furthermore, we assume the linear system is consistent (that is, $\mathrm{rank}([A|c]) = k$) for otherwise there could be no solution.) We call any point b that satisfies $Ab = c$ a *feasible point*.

We write the equality-constrained least squares optimization problem as

$$\begin{aligned} &\min_b f(b) = (y - Xb)^{\mathrm{T}}(y - Xb) \\ &\text{s.t. } Ab = c. \end{aligned} \tag{4.47}$$

If b_c is any feasible point (that is, $Ab_c = c$), then any other feasible point can be represented as $b_c + p$, where p is any vector in the null space of A, $\mathcal{N}(A)$. From our discussion in Sect. 3.5.2, we know that the dimension of $\mathcal{N}(A)$ is $m - k$, and its order is m. If N is an $m \times (m - k)$ matrix whose columns form a basis for $\mathcal{N}(A)$, all feasible points can be generated by $b_c + Nz$, where $z \in \mathbb{R}^{m-k}$. Hence, we need only consider the restricted variables

$$b = b_c + Nz$$

and the "reduced" function

$$h(z) = f(b_c + Nz).$$

The argument of this function is a vector with only $m - k$ elements instead of m elements as in the unconstrained problem. It is clear, however, that an unconstrained minimum point, z_*, of $h(z)$ yields a solution, $b_* = b_c + Nz_*$, of the original constrained minimization problem.

4.4.5.2 The Reduced Gradient and Reduced Hessian

If we assume differentiability, the gradient and Hessian of the reduced function can be expressed in terms of the original function:

$$\begin{aligned} \mathrm{g}_h(z) &= N^{\mathrm{T}}\mathrm{g}_f(b_c + Nz) \\ &= N^{\mathrm{T}}\mathrm{g}_f(b) \end{aligned} \tag{4.48}$$

and

$$\begin{aligned} \mathrm{H}_h(z) &= N^{\mathrm{T}}\mathrm{H}_f(b_c + Nz)N \\ &= N^{\mathrm{T}}\mathrm{H}_f(b)N. \end{aligned} \tag{4.49}$$

In equation (4.48), $N^{\mathrm{T}}\mathrm{g}_f(b)$ is called the *reduced gradient* or *projected gradient*, and $N^{\mathrm{T}}\mathrm{H}_f(b)N$ in equation (4.49) is called the *reduced Hessian* or *projected Hessian*.

The properties of stationary points referred to above are the conditions that determine a minimum of this reduced objective function; that is, b_* is a minimum if and only if

$$N^{\mathrm{T}}\mathrm{g}_f(b_*) = 0, \tag{4.50}$$

$$N^{\mathrm{T}}\mathrm{H}_f(b_*)N \succ 0, \tag{4.51}$$

and

$$Ab_* = c. \tag{4.52}$$

These relationships then provide the basis for the solution of the optimization problem.

4.4.5.3 Lagrange Multipliers

Because the $m \times m$ matrix $[N|A^{\mathrm{T}}]$ spans $\mathbb{R}^m$, we can represent the vector $\mathrm{g}_f(b_*)$ as a linear combination of the columns of N and A^{T}, that is,

$$\begin{aligned}\mathrm{g}_f(b_*) &= [N|A^{\mathrm{T}}] \begin{pmatrix} z_* \\ \lambda_* \end{pmatrix} \\ &= \begin{pmatrix} Nz_* \\ A^{\mathrm{T}}\lambda_* \end{pmatrix},\end{aligned}$$

where z_* is an $(m-k)$-vector and λ_* is a k-vector. Because $\mathrm{g}_h(z_*) = 0$, Nz_* must also vanish (that is, $Nz_* = 0$), and thus, at the optimum, the nonzero elements of the gradient of the objective function are linear combinations of the rows of the constraint matrix, $A^{\mathrm{T}}\lambda_*$.

The k elements of the linear combination vector λ_* are called *Lagrange multipliers*.

4.4.5.4 The Lagrangian

Let us now consider a simple generalization of the constrained problem above and an abstraction of the results above so as to develop a general method. We consider the problem

$$\begin{aligned}&\min_x f(x) \\ &\text{s.t. } c(x) = 0,\end{aligned} \tag{4.53}$$

where f is a scalar-valued function of an m-vector variable and c is a k-vector-valued function of the variable. There are some issues concerning the equation $c(x) = 0$ that we will not go into here. Obviously, we have the same concerns as before; that is, whether $c(x) = 0$ is consistent and whether the individual equations $c_i(x) = 0$ are independent. Let us just assume they are, and proceed. (Again, we refer the interested reader to a more general text on optimization.)

Motivated by the results above, we form a function that incorporates a dot product of Lagrange multipliers and the function $c(x)$:

$$F(x) = f(x) + \lambda^{\mathrm{T}} c(x). \tag{4.54}$$

This function is called the *Lagrangian*. The solution, (x_*, λ_*), of the optimization problem occurs at a stationary point of the Lagrangian,

$$\mathrm{g}_f(x_*) = \begin{pmatrix} 0 \\ \mathrm{J}_c(x_*)^{\mathrm{T}}\lambda_* \end{pmatrix}. \tag{4.55}$$

Thus, at the optimum, the gradient of the objective function is a linear combination of the columns of the Jacobian of the constraints.

4.4.5.5 Another Example: The Rayleigh Quotient

The important equation (3.265) on page 156 can also be derived by using differentiation. This equation involves maximization of the Rayleigh quotient (equation (3.266)),

$$x^{\mathrm{T}}Ax/x^{\mathrm{T}}x$$

under the constraint that $x \neq 0$. In this function, this constraint is equivalent to the constraint that $x^{\mathrm{T}}x$ equal a fixed nonzero constant, which is canceled in the numerator and denominator. We can arbitrarily require that $x^{\mathrm{T}}x = 1$, and the problem is now to determine the maximum of $x^{\mathrm{T}}Ax$ subject to the constraint $x^{\mathrm{T}}x = 1$. We now formulate the Lagrangian

$$x^{\mathrm{T}}Ax - \lambda(x^{\mathrm{T}}x - 1), \tag{4.56}$$

differentiate, and set it equal to 0, yielding

$$Ax - \lambda x = 0.$$

This implies that a stationary point of the Lagrangian occurs at an eigenvector and that the value of $x^{\mathrm{T}}Ax$ is an eigenvalue. This leads to the conclusion that the maximum of the ratio is the maximum eigenvalue. We also see that the second order necessary condition for a local maximum is satisfied; $A - \lambda I$ is nonpositive definite when λ is the maximum eigenvalue. (We can see this using the spectral decomposition of A and then subtracting λI.) Note that we do not have the sufficient condition that $A - \lambda I$ is negative definite ($A - \lambda I$ is obviously singular), but the fact that it is a maximum is established by inspection of the finite set of stationary points.

4.4.5.6 Optimization of Functions with Inequality Constraints

Inequality constraints are much more difficult to deal with than equality constraints. The feasible region, instead of being a manifold of dimension $m - 1$ or less, may be a subset of with dimension m.

4.4.5.7 Inequality-Constrained Linear Least Squares Problems

Similarly to the equality-constrained least squares optimization problem (4.47), an inequality-constrained least squares optimization problem can be written as

$$\begin{aligned} &\min_{b} f(b) = (y - Xb)^{\mathrm{T}}(y - Xb) \\ &\text{s.t. } Ab \leq c, \end{aligned} \tag{4.57}$$

where A is a $k \times m$ matrix. There are no restrictions on k or on the rank of A as before, except as required to make the feasible region convex. (If the feasible region is not convex or it it is not connected, we may need to decompose

the optimization problem into a set of optimization problems.) In the most common cases the region is a convex cone.

The analysis leading to a reduced gradient and a reduced Hessian and to the conditions given on page 209 that characterize a solution does not apply. In an equality-constrained problem, all of the constraints are *active*, in the sense that a feasible point can change only in limited ways and still satisfy the constraint. Geometrically, the feasible region lies in a hyperplane. (Of course, it may be restricted even further within the hyperplane.) In an inequality-constrained problem, however, the feasible region defined by $Ab \leq c$ consists of the intersection of halfspaces defined by the rows of A. For the i^{th} row of A, a_i, the halfspace consists of all b satisfying $a_i^{\mathrm{T}} b \leq c_i$. The outward-pointing normal to the hyperplane defining the halfspace is a_i. For any point $b^{(0)}$ in the feasible region, a particular constraint, say $a_i^{\mathrm{T}} b \leq c_i$, is "active" if $a_i^{\mathrm{T}} b^{(0)} = c_i$; otherwise, that particular constraint is not active, and points in a neighborhood of $b^{(0)}$ are also feasible.

If b_* is a solution to the constrained optimization problem (4.57), then the gradient at b_*, $X^{\mathrm{T}}(y - Xb_*)$, would be zero except for the active constraints. The negative of the gradient, therefore can be represented as a linear combination of the normals to the hyperplanes defining the constraints that are active. Now, define a k-vector e such that $e_i \geq 0$ if $a_i^{\mathrm{T}} b_* = c_i$ and $e_i = 0$ if $a_i^{\mathrm{T}} b_* < c_i$. Let S (for "slack") represent the set of all i such that $e_i = 0$, and let E (for "equal") represent the set of all i such that $e_i \geq 0$. Then there are values of e_i as defined above such that

$$\sum_i e_i a_i = X^{\mathrm{T}}(y - Xb_*). \tag{4.58}$$

Now consider a perturbation of the optimal b_*, δb_*. In order for $b_* + \delta b_*$ to remain feasible, the perturbation must satisfy $\delta b_*^{\mathrm{T}} a_i \leq 0$ for all $i \in E$. Multiplying both sides of equation (4.58) by δb_*^{T}, because $e_i \geq 0$, we have $\delta b_*^{\mathrm{T}} X^{\mathrm{T}}(y - Xb_*) \leq 0$. Because

$$\begin{aligned}(y - X(b_* + \delta b_*))^{\mathrm{T}}(y - X(b_* + \delta b_*)) = \\ (y - Xb_*)^{\mathrm{T}}(y - Xb_*) - 2\delta b_*^{\mathrm{T}} X^{\mathrm{T}}(y - Xb_*) + (X\delta b_*)^{\mathrm{T}} X\delta b_*,\end{aligned}$$

no feasible solution will further decrease the value of the objective function $(y - Xb)^{\mathrm{T}}(y - Xb)$; that is, b_* is optimal.

We can summarize this discussion by the statement that b_* is a solution for the problem (4.57) if and only if there exists a k-vector e such that $e_i \geq 0$ if $a_i^{\mathrm{T}} b_* = c_i$ and $e_i = 0$ if $a_i^{\mathrm{T}} b_* < c_i$, and

$$A^{\mathrm{T}} e = X^{\mathrm{T}}(y - Xb_*). \tag{4.59}$$

These conditions are special cases of the Kuhn-Tucker or Karush-Kuhn-Tucker conditions for constrained optimization problems (see Griva, Nash, and Sofer, 2009, for example).

The variables in e are called *dual* variables. The non-zero dual variables are *slack* variables.

4.4.5.8 Nonlinear Least Squares as an Inequality-Constrained Problem

The Levenberg-Marquardt adjustment in equation (4.40) can be thought of as a Lagrangian multiplier formulation of the constrained problem:

$$\begin{aligned} \min_x \ & \tfrac{1}{2}\left\|\mathrm{J}_r(\theta^{(k)})x + r(\theta^{(k)})\right\| \\ \text{s.t.} \quad & \left\|S^{(k)}x\right\| \leq \delta_k. \end{aligned} \tag{4.60}$$

See Exercise 4.8.

The Lagrange multiplier $\lambda^{(k)}$ is zero if $p^{(k)}$ from equation (4.39) satisfies $\|p^{(k)}\| \leq \delta_k$; otherwise, it is chosen so that $\left\|S^{(k)}p^{(k)}\right\| = \delta_k$.

4.4.6 Optimization Without Differentiation

Differentiation is often an effective method for solving an optimization problem. It may lead us to a stationary point, and then in optimization problems, we must establish that the stationary point is indeed a maximum or minimum, which we do by use of second-order derivatives or else by inspection.

If the objective function is not differentiable, then of course differentiation cannot be used. Also, even if the function is differentiable, if the optimum occurs on the boundary of the domain of the objective function, then differentiation may not find it. Before attempting to solve an optimization problem, we should do some preliminary analysis of the problem. Often in working out maximum likelihood estimates, for example, students immediately think of differentiating, setting to 0, and solving. This requires that the likelihood function be differentiable, that it be concave, and that the maximum occur at an interior point of the parameter space. Keeping in mind exactly what the problem is—one of finding a maximum—often leads to the correct solution more quickly.

Sometimes in optimization problems, even in the simple cases of differentiable objective functions, it may be easier to use other methods to determine the optimum; that is, methods that do not depend on derivatives. For example, the Rayleigh quotient for which we determined the maximum by differentiation above, could be determined easily without differentiation, as we did on page 156.

For another example, a constrained minimization problem we encounter occasionally is

$$\min_{X \succ 0} \left(\log|X| + \mathrm{tr}(X^{-1}A)\right) \tag{4.61}$$

for a given positive definite matrix A and subject to X being positive definite. (The canonical problem giving rise to this minimization problem is the use of likelihood methods with a multivariate normal distribution. Also recall that an alternate notation for $\det(X)$ is $|X|$, which I am using here, as I often do in working with the multivariate normal PDF, for example in equation (4.73) on page 219.)

The derivatives given in Tables 4.2 and 4.3 could be used to minimize the function in (4.61). The derivatives set equal to 0 immediately yield $X = A$. This means that $X = A$ is a stationary point, but whether or not it is a minimum would require further analysis. As is often the case with such problems, an alternate approach leaves no such pesky complications. Let A and X be $n \times n$ positive definite matrices, and let $c_1, \ldots, c_n$ be the eigenvalues of $X^{-1}A$. Now, by property 8 on page 136 these are also the eigenvalues of $X^{-1/2}AX^{-1/2}$, which is positive definite (see inequality (3.161) on page 114). Now, consider the expression (4.61) with general X minus the expression with $X = A$:

$$\begin{aligned}
\log|X| + \mathrm{tr}(X^{-1}A) - \log|A| - \mathrm{tr}(A^{-1}A) &= \log|XA^{-1}| + \mathrm{tr}(X^{-1}A) - \mathrm{tr}(I) \\
&= -\log|X^{-1}A| + \mathrm{tr}(X^{-1}A) - n \\
&= -\log\left(\prod_i c_i\right) + \sum_i c_i - n \\
&= \sum_i (-\log c_i + c_i - 1) \\
&\geq 0
\end{aligned}$$

because if $c > 0$, then $\log c \leq c - 1$, and the minimum occurs when each $c_i = 1$; that is, when $X^{-1}A = I$. Thus, the minimum of expression (4.61) occurs uniquely at $X = A$.

4.5 Integration and Expectation: Applications to Probability Distributions

Just as we can take derivatives with respect to vectors or matrices, we can also take antiderivatives or definite integrals with respect to vectors or matrices. Our interest is in integration of functions weighted by a multivariate probability density function, and for our purposes we will be interested only in definite integrals.

Again, there are three components:

- the differential (the variable of the operation) and its domain (the range of the integration),
- the integrand (the function), and
- the result of the operation (the integral).

In the simplest case, all three of these objects are of the same type; they are scalars. In the happy cases that we consider, each definite integral within the nested sequence exists, so convergence and order of integration are not issues. (The implication of these remarks is that while there is a much bigger field of mathematics here, we are concerned about the relatively simple cases that suffice for our purposes.)

In this section we first consider some general issues relating to integration with respect to a vector, and then we consider applications to probability, specifically, to the computation of moments of a vector-valued random variable. The moments of a scalar-values random variable X are defined as $\mathrm{E}(X^k)$ (where the expectation operator, $\mathrm{E}(\cdot)$, is an integral, as defined in equation (4.68)). Obviously the moments of vector-valued random variables must be defined slightly differently.

In some cases of interest involving vector-valued random variables, the differential is the vector representing the values of the random variable and the integrand has a scalar function (the probability density) as a factor. In one type of such an integral, the integrand is only the probability density function, and the integral evaluates to a probability, which of course is a scalar. In another type of such an integral, the integrand is a vector representing the values of the random variable times the probability density function. The integral in this case evaluates to a vector, namely the expectation of the random variable over the domain of the integration. Finally, in an example of a third type of such an integral, the integrand is an outer product with itself of a vector representing the values of the random variable minus its mean times the probability density function. The integral in this case evaluates to a variance-covariance matrix. In each of these cases, the integral is the same type of object as the integrand.

4.5.1 Multidimensional Integrals and Integrals Involving Vectors and Matrices

An integral of the form $\int f(v)\,\mathrm{d}v$, where v is a vector, can usually be evaluated as a multiple integral with respect to each differential $\mathrm{d}v_i$. Likewise, an integral of the form $\int f(M)\,\mathrm{d}M$, where M is a matrix can usually be evaluated by "unstacking" the columns of $\mathrm{d}M$, evaluating the integral as a multiple integral with respect to each differential $\mathrm{d}m_{ij}$, and then possibly "restacking" the result.

Probabilities and expectations in multivariate probability distributions are defined in terms of multivariate integrals. As with many well-known univariate integrals, such as $\Gamma(\cdot)$, that relate to univariate probability distributions, there are standard multivariate integrals, such as the multivariate gamma, $\Gamma_d(\cdot)$, that relate to multivariate probability distributions. Using standard integrals often facilitates the computations.

4.5.1.1 Change of Variables: Jacobians

When evaluating an integral of the form $\int f(x)\,\mathrm{d}x$, where x is a vector, for various reasons, we may form a one-to-one differentiable transformation of the variables of integration; that is, of x. We write x as a function of the new variables; that is, $x = g(y)$, and so $y = g^{-1}(x)$. A simple fact from elementary multivariable calculus is

$$\int_{R(x)} f(x)\,\mathrm{d}x = \int_{R(y)} f(g(y))\,|\det(\mathrm{J}_g(y))|\mathrm{d}y, \tag{4.62}$$

where $R(y)$ is the image of $R(x)$ under g^{-1} and $\mathrm{J}_g(y)$ is the Jacobian of g (see equation (4.12)). (This is essentially a chain rule result for $\mathrm{d}x = \mathrm{d}(g(y)) = \mathrm{J}_g \mathrm{d}y$ under the interpretation of $\mathrm{d}x$ and $\mathrm{d}y$ as positive differential elements and the interpretation of $|\det(\mathrm{J}_g)|$ as a volume element, as discussed on page 74.)

In the simple case of a full rank linear transformation of a vector, the Jacobian is constant, and so for $y = Ax$ with A a fixed matrix, we have

$$\int f(x)\,\mathrm{d}x = |\det(A)|^{-1} \int f(A^{-1}y)\,\mathrm{d}y. \tag{4.63}$$

(Note that we write $\det(A)$ instead of $|A|$ for the determinant if we are to take the absolute value of it because otherwise we would have $||A||$, which is a symbol for a norm. However, $|\det(A)|$ is not a norm; it lacks each of the properties listed on page 25.)

In the case of a full rank linear transformation of a matrix variable of integration, the Jacobian is somewhat more complicated, but the Jacobian is constant for a fixed transformation matrix. For a transformation $Y = AX$, we determine the Jacobian as above by considering the columns of X one by one. Hence, if X is an $n \times m$ matrix and A is a constant nonsingular matrix, we have

$$\int f(X)\,\mathrm{d}X = |\det(A)|^{-m} \int f(A^{-1}Y)\,\mathrm{d}Y. \tag{4.64}$$

For a transformation of the form $Z = XB$, we determine the Jacobian by considering the rows of X one by one.

4.5.2 Integration Combined with Other Operations

Integration and another finite linear operator can generally be performed in any order. For example, because the trace is a finite linear operator, integration and the trace can be performed in either order:

$$\int \mathrm{tr}(A(x))\mathrm{d}x = \mathrm{tr}\left(\int A(x)\mathrm{d}x\right). \tag{4.65}$$

For a scalar function of two vectors x and y, it is often of interest to perform differentiation with respect to one vector and integration with respect to the other vector. In such cases, it is of interest to know when these operations can be interchanged. The answer is given in the following theorem, which is a consequence of the Lebesgue dominated convergence theorem. Its proof can be found in any standard text on real analysis.

Let $\mathcal{X}$ be an open set, and let $f(x, y)$ and $\partial f/\partial x$ be scalar-valued functions that are continuous on $\mathcal{X} \times \mathcal{Y}$ for some set $\mathcal{Y}$. Now suppose there are scalar functions $g_0(y)$ and $g_1(y)$ such that

$$\left.\begin{array}{c} |f(x,y)| \le g_0(y) \\ \\ \|\frac{\partial}{\partial x} f(x,y)\| \le g_1(y) \end{array}\right\} \quad \text{for all } (x,y) \in \mathcal{X} \times \mathcal{Y},$$

$$\int_{\mathcal{Y}} g_0(y)\,\mathrm{d}y < \infty,$$

and

$$\int_{\mathcal{Y}} g_1(y)\,\mathrm{d}y < \infty.$$

Then

$$\frac{\partial}{\partial x}\int_{\mathcal{Y}} f(x,y)\,\mathrm{d}y = \int_{\mathcal{Y}} \frac{\partial}{\partial x} f(x,y)\,\mathrm{d}y. \tag{4.66}$$

An important application of this interchange is in developing the information inequality for "regular" families of probability distributions.

4.5.3 Random Variables and Probability Distributions

Note on notation: *I generally use upper-case letters to represent matrices and lower-case letters to represent vectors and scalars. I also generally use upper-case letters to represent random variables and corresponding lower-case letters to represent their realizations. I will generally follow this convention in this section, but where the two conventions come in conflict, I will give precedence to the upper-case representation; that is, an upper-case letter may be a matrix, random or fixed, or it may be a random variable with any structure. I also generally use Greek letters to represent parameters of probability distributions, and for parameters, I also use my convention of upper-case for matrices and lower-case for vectors or scalars.*

Random variables are functions from a "sample space" into the reals. A d-vector random variable is a function from some sample space into $\mathbb{R}^d$, and an $n \times d$ matrix random variable is a function from a sample space into $\mathbb{R}^{n\times d}$. (Technically, in each case, the function is required to be *measurable* with respect to a *measure* defined in the context of the sample space and an appropriate collection of subsets of the sample space. A careful development of the theory is available at `http://mason.gmu.edu/~jgentle/books/MathStat.pdf`.)

Associated with each vector or matrix random variable is a distribution function, or cumulative distribution function (CDF), which represents the probability that the random variable takes values less than or equal to a specified value. The derivative of the distribution function with respect to an appropriate measure is nonnegative and integrates to 1 over the full space formed by $\mathbb{R}$ (that is, $\mathbb{R}$, $\mathbb{R}^d$, or $\mathbb{R}^{n\times d}$). This derivative is called the probability density function, or PDF. (There are some technical issues that arise in some cases, but I will ignore those issues here.)

If X is a random variable, and $p_X(x)$ is the PDF of X, we have

$$\int_{D(X)} p_X(x)\,\mathrm{d}x = 1, \tag{4.67}$$

where $D(X)$ is the set of the image of X in which $p_X(x) > 0$ and is called the support of the distribution.

The expected value of a function f of the random variable X is

$$\mathrm{E}(f(X)) = \int_{D(X)} f(x) p_X(x)\,\mathrm{d}x, \tag{4.68}$$

if the integral exists.

We see immediately that the expected value operator is linear; that is, if a is independent of X and consists of real elements of the appropriate structure that $af(X)$ makes sense (and exists), and if $g(X)$ has the structure such that $af(X) + g(X)$ makes sense (and exists), then

$$\mathrm{E}\left(af(X) + g(X)\right) = a\mathrm{E}(f(X)) + \mathrm{E}(g(X)). \tag{4.69}$$

Some special forms of f in equation (4.68) yield the (raw) moments of the random variable. In the case of a scalar random variable, these are just the power functions, and $\mathrm{E}(X^k)$ is the k^{th} moment of X.

4.5.3.1 Vector Random Variables

If the random variable X, and consequently, x in equation (4.68), is a vector, then if we interpret $\int_{D(X)} \mathrm{d}x$ as a nest of univariate integrals, the result of the integration of the vector $f(x)p_X(x)$ is clearly the same type of mathematical object as $f(x)$. For example, if $f(x) = x$, the expectation is the mean, which is a vector.

The first consideration, for a vector-valued random variable X, is how should the second moment be represented. (This is not a question of what are the individual elements that must comprise this moment; those are obvious. It is merely a question of representation.) Kollo and von Rosen (2005), page 173, suggest three obvious ways. The most commonly used method, and the only one I will consider here, is the expectation of the outer product,

$$\mathrm{E}(XX^{\mathrm{T}}), \tag{4.70}$$

which of course is a matrix.

From (4.70) we get the *variance-covariance matrix* for the vector random variable X,

$$\mathrm{V}(X) = \mathrm{E}\left((X - \mathrm{E}(X))(X - \mathrm{E}(X))^{\mathrm{T}}\right). \tag{4.71}$$

We also have, from the linearity of the expectation operator, if A is a fixed matrix not dependent on X, with a number of columns equal to the order of X,

$$\mathrm{E}(AX) = A\mathrm{E}(X) \qquad \text{and} \qquad \mathrm{V}(AX) = A\mathrm{V}(X)A^{\mathrm{T}}. \tag{4.72}$$

4.5.3.2 The Multivariate Normal Distribution

The most important vector random variables are those whose distributions are in the multivariate normal family. In the case of the multivariate normal distribution, the variances and covariances together with the means completely characterize the distribution. The PDF of the multivariate normal with mean μ and variance-covariance matrix Σ is

$$p_X(x) = (2\pi)^{-d/2}|\Sigma|^{-1/2}\mathrm{e}^{-(x-\mu)^{\mathrm{T}}\Sigma^{-1}(x-\mu)/2}, \tag{4.73}$$

where $|\Sigma| = \det(\Sigma)$. (I prefer the notation "$\det(\cdot)$" for a determinant, but in this section, I will mix the two notations.) In this notation, we are obviously assuming that the variance-covariance matrix Σ is nonsingular, and we will assume this anytime we refer to the "multivariate normal distribution". There is a related normal distribution, called a "singular normal distribution" or a "curved normal distribution", in which this is not the case (and $|\Sigma|^{-1/2}$ does not exist), but we will not discuss that family of distributions.

For a vector of order d, we often denote the family of multivariate normal distributions by $\mathrm{N}_d(\mu, \Sigma)$. We write

$$X \sim \mathrm{N}_d(\mu, \Sigma). \tag{4.74}$$

Each member of this family of distributions corresponds to a specific μ and Σ.

As with any PDF, the function p_X in (4.73) integrates to 1. The fundamental integral associated with the d-variate normal distribution, sometimes called *Aitken's integral*, is

$$\int_{\mathbb{R}^d} \mathrm{e}^{-(x-\mu)^{\mathrm{T}}\Sigma^{-1}(x-\mu)/2}\,\mathrm{d}x. \tag{4.75}$$

The rank of the integral is the same as the rank of the integrand. ("Rank" is used here in the sense of "number of dimensions".) In this case, the integrand and the integral are scalars.

We can evaluate the integral (4.75) by evaluating the individual single integrals after making the change of variables $y_i = x_i - \mu_i$. It can also be seen by first noting that because Σ^{-1} is positive definite, as in equation (3.272), it can be written as $P^{\mathrm{T}}\Sigma^{-1}P = I$ for some nonsingular matrix P. Now, after the translation $y = x - \mu$, which leaves the integral unchanged, we make the linear change of variables $z = P^{-1}y$, with the associated Jacobian $|\det(P)|$, as in equation (4.62). From $P^{\mathrm{T}}\Sigma^{-1}P = I$, we have $|\det(P)| = (\det(\Sigma))^{1/2}$ because the determinant is positive. (Note that $(\det(\Sigma))^{1/2}$ is the same as $|\Sigma|^{-1/2}$ above.) Aitken's integral therefore is

$$\int_{\mathbb{R}^d} \mathrm{e}^{-y^{\mathrm{T}}\Sigma^{-1}y/2}\,\mathrm{d}y = \int_{\mathbb{R}^d} \mathrm{e}^{-(Pz)^{\mathrm{T}}\Sigma^{-1}Pz/2}\,(\det(\Sigma))^{1/2}\mathrm{d}z$$

$$= \int_{\mathbb{R}^d} \mathrm{e}^{-z^{\mathrm{T}}z/2}\,\mathrm{d}z\,(\det(\Sigma))^{1/2}$$
$$= (2\pi)^{d/2}(\det(\Sigma))^{1/2}. \tag{4.76}$$

We stated that this normal distribution has mean μ and variance-covariance matrix Σ. We can see this directly:

$$\begin{aligned}\mathrm{E}(X) &= (2\pi)^{-d/2}|\Sigma|^{-1/2}\int_{\mathbb{R}^d} x\mathrm{e}^{-(x-\mu)^{\mathrm{T}}\Sigma^{-1}(x-\mu)/2}\,\mathrm{d}x\\ &= \mu;\end{aligned}$$

and for the variance we have

$$\begin{aligned}\mathrm{V}(X) &= \mathrm{E}\left((X-\mathrm{E}(X))(X-\mathrm{E}(X))^{\mathrm{T}}\right)\\ &= (2\pi)^{-d/2}|\Sigma|^{-1/2}\int_{\mathbb{R}^d}\left((x-\mu)(x-\mu)^{\mathrm{T}}\right)\mathrm{e}^{-(x-\mu)^{\mathrm{T}}\Sigma^{-1}(x-\mu)/2}\,\mathrm{d}x\\ &= \Sigma.\end{aligned}$$

4.5.3.3 Matrix Random Variables

The relationships of the elements of a matrix-valued random variable to each other may be useful in modeling interrelated populations. In some cases, a matrix is the obvious structure for a random variable, such as a Wishart matrix for modeling the distribution of a sample variance-covariance matrix. Kollo and von Rosen (2005) take a matrix as the basic structure for a multivariate random variable, in particular, for a random variable with a multivariate normal distribution.

A matrix random variable is often useful in certain applications, such as when p n-variate random variables may be related. The columns of an $n \times p$ matrix-valued random variable X may correspond to these n-variate random variables. In applications, it may be reasonable to allow the columns to have different means, but to assume that they have a common variance-covariance matrix Σ. The mean of X is the matrix M, whose columns, $\mu_1, \ldots, \mu_p$, are the means of the columns of X.

The probability distributions governing random matrices determine various properties of the matrices, such as whether or not the matrices are positive definite, or the probability that the matrices are positive definite, for example. One fact that is fairly obvious is that if the individual elements of a matrix have continuous probability distributions, both marginal and all nondegenerate conditional distributions, then the probability that the matrix is full rank is 1. (You are asked to write a proof of this in Exercise 4.11.)

The matrix normal distribution can be defined from a set of s random r-vectors $Y_1, \ldots, Y_s$ that are independently and identically distributed as $\mathrm{N}_r(0, I_r)$. For a fixed $n \times r$ matrix A and fixed n-vector μ_i, the vector $\widetilde{Y}_i = \mu_i + AY_i$ is distributed as $\mathrm{N}_n(\mu_i, AA^{\mathrm{T}})$. Now consider the s-vector Y_{j*}

consisting of the j^{th} element of each Y_i. It is clear that $Y_{j*} \sim \mathrm{N}_r(0, I_s)$, and hence for a given $d \times s$ matrix B, $BY_{j*} \sim \mathrm{N}_d(0, BB^{\mathrm{T}})$. Now, forming a matrix Y with the Y_is as the columns and a matrix M with the μ_is as the columns, and putting these two transformations together, we have a matrix normal random variable:

$$X = M + AYB^{\mathrm{T}}. \tag{4.77}$$

Let $\Sigma = AA^{\mathrm{T}}$ and $\Psi = BB^{\mathrm{T}}$ denote the variance-covariance matrices in the multivariate distributions above. These are parameters in the distribution of X. We denote the distribution of the random matrix X with three parameters as

$$X \sim \mathrm{N}_{n,d}(M, \Sigma, \Psi). \tag{4.78}$$

We may also wish to require that Σ and Ψ be positive definite. This requirement would be satisfied if A and B are each of full row rank.

There are of course various ways that relationships among the individual multivariate random variables (that is, the columns of X) could be modeled. A useful model is a multivariate normal distribution for $\mathrm{vec}(X)$ as in expression (4.74), instead of the matrix normal distribution for X as above:

$$\mathrm{vec}(X) \sim \mathrm{N}_{nd}\left(\mathrm{vec}(M), \Psi \otimes \Sigma\right). \tag{4.79}$$

This distribution is easy to work with because it is just an ordinary multivariate (vector) normal distribution. The variance-covariance matrix has a special structure, called a *Kronecker structure*, that incorporates the homoscedasticity of the columns as well as their interdependence.

An interesting and useful matrix distribution is that of a random variance-covariance matrix formed from n independent d-variate normal random variables distributed as $\mathrm{N}_d(0, \Sigma)$. This is the Wishart distribution, see Exercise 4.12.

A common matrix integral that arises in the Wishart and related distributions is the complete d-variate gamma function, denoted by $\Gamma_d(x)$ and defined as

$$\Gamma_d(x) = \int_D \mathrm{e}^{-\mathrm{tr}(A)} |A|^{x-(d+1)/2} \, \mathrm{d}A, \tag{4.80}$$

where D is the set of all $d \times d$ positive definite matrices, $A \in D$, and $x > (d-1)/2$. A multivariate gamma distribution can be defined in terms of the integrand. (There are different definitions of a multivariate gamma distribution.) The multivariate gamma function also appears in the probability density function for a Wishart random variable (see Muirhead 1982, Carmeli 1983, or Kollo and von Rosen 2005, for example).

Another distribution for random matrices is one in which the individual elements have identical and independent normal distributions. This distribution of matrices was named the BMvN distribution by Birkhoff and Gulati (1979) (from the last names of three mathematicians who used such random matrices in numerical studies). Birkhoff and Gulati (1979) showed that if the elements

of the $n \times n$ matrix X are i.i.d. $\mathrm{N}(0, \sigma^2)$, and if Q is an orthogonal matrix and R is an upper triangular matrix with positive elements on the diagonal such that $QR = X$, then Q has the *Haar distribution*. (The factorization $X = QR$ is called the QR decomposition and is discussed on page 251. If X is a random matrix as described, this factorization exists with probability 1.) The Haar(n) distribution is uniform over the space of $n \times n$ orthogonal matrices.

The measure

$$\mu(D) = \int_D H^{\mathrm{T}} \, \mathrm{d}H, \tag{4.81}$$

where D is a subset of the orthogonal group $\mathcal{O}(n)$ (see page 133), is called the *Haar measure*. This measure is used to define a kind of "uniform" probability distribution for orthogonal factors of random matrices. For any $Q \in \mathcal{O}(n)$, let QD represent the subset of $\mathcal{O}(n)$ consisting of the matrices $\tilde{H} = QH$ for $H \in D$ and DQ represent the subset of matrices formed as HQ. From the integral, we see that

$$\mu(QD) = \mu(DQ) = \mu(D),$$

so the Haar measure is invariant to multiplication within the group. The measure is therefore also called the *Haar invariant measure* over the orthogonal group. (See Muirhead 1982, for more properties of this measure.)

Exercises

4.1. Use equation (4.6), which defines the derivative of a matrix with respect to a scalar, to show the product rule equation (4.3) directly:

$$\frac{\partial YW}{\partial x} = \frac{\partial Y}{\partial x} W + Y \frac{\partial W}{\partial x}.$$

4.2. For the n-vector x, compute the gradient $\mathrm{g}_{\mathrm{V}}(x)$, where $\mathrm{V}(x)$ is the variance of x, as given in equation (2.76).
Hint: Use the chain rule.

4.3. Prove $\nabla \times (\nabla f) = 0$.
(This is equation (4.13) expressed in different notation).

4.4. Prove $\nabla \cdot (\nabla \times f) = 0$.
(This is equation (4.14) expressed in different notation).

4.5. For the square, nonsingular matrix Y, derive equation (4.15):

$$\frac{\partial Y^{-1}}{\partial x} = -Y^{-1} \frac{\partial Y}{\partial x} Y^{-1}.$$

Hint: Differentiate $YY^{-1} = I$.

4.6. Newton's method.
You should not, of course, just blindly pick a starting point and begin iterating. How can you be sure that your solution is a local optimum?

Can you be sure that your solution is a global optimum? It is often a good idea to make some plots of the function. In the case of a function of a single variable, you may want to make plots in different scales. For functions of more than one variable, profile plots may be useful (that is, plots of the function in one variable with all the other variables held constant).

a) Use Newton's method to determine the maximum of the function $f(x) = \sin(4x) - x^4/12$.

b) Use Newton's method to determine the minimum of

$$f(x_1, x_2) = 2x_1^4 + 3x_1^3 + 2x_1^2 + x_2^2 - 4x_1x_2.$$

What is the Hessian at the minimum?

4.7. Consider the log-likelihood $l(\mu, \Sigma; y)$ for the d-variate normal distribution, equation (4.43). Be aware of the subtle issue referred to in the text. It has to do with whether $\sum_{i=1}^n (y_i - \bar{y})(y_i - \bar{y})^{\mathrm{T}}$ is positive definite.

a) Replace the parameters μ and Σ by the variables $\hat{\mu}$ and $\widehat{\Sigma}$, take derivatives with respect to $\hat{\mu}$ and $\widehat{\Sigma}$, set them equal to 0, and solve for the maximum likelihood estimates. What assumptions do you have to make about n and d?

b) Another approach to maximizing the expression in equation (4.43) is to maximize the last term with respect to $\hat{\mu}$ (this is the only term involving μ) and then, with the maximizing value substituted, to maximize

$$-\frac{n}{2}\log|\Sigma| - \frac{1}{2}\mathrm{tr}\left(\Sigma^{-1}\sum_{i=1}^n (y_i - \bar{y})(y_i - \bar{y})^{\mathrm{T}}\right).$$

Use this approach to determine the maximum likelihood estimates $\hat{\mu}$ and $\widehat{\Sigma}$.

4.8. Compare the constrained optimization problem (4.60) with the Levenberg-Marquardt regularization in equation (4.40). Carefully identify the corresponding terms in the two different expressions.

4.9. Let

$$D = \left\{ \begin{bmatrix} c & -s \\ s & c \end{bmatrix} : -1 \le c \le 1, c^2 + s^2 = 1 \right\}.$$

Evaluate the Haar measure $\mu(D)$. (This is the class of 2×2 rotation matrices; see equation (5.3), page 232.)

4.10. Write a program (use R, Matlab, Python, Fortran, C) to generate $n \times n$ random orthogonal matrices with the Haar uniform distribution. Use the following method due to Heiberger (1978), which was modified by Stewart (1980). (See also Tanner and Thisted 1982.)

a) Generate $n-1$ independent i-vectors, $x_2, x_3, \ldots, x_n$, from $N_i(0, I_i)$. (x_i is of length i.)

b) Let $r_i = \|x_i\|_2$, and let $\widetilde{H}_i$ be the $i \times i$ reflection matrix that transforms x_i into the i-vector $(r_i, 0, 0, \ldots, 0)$.
c) Let H_i be the $n \times n$ matrix

$$\begin{bmatrix} I_{n-i} & 0 \\ 0 & \widetilde{H}_i \end{bmatrix},$$

and form the diagonal matrix,

$$J = \mathrm{diag}\big((-1)^{b_1}, (-1)^{b_2}, \ldots, (-1)^{b_n}\big),$$

where the b_i are independent realizations of a Bernoulli random variable.
d) Deliver the orthogonal matrix $Q = JH_1H_2 \cdots H_n$.

The matrix Q generated in this way is orthogonal and has a Haar distribution.
Can you think of any way to test the goodness-of-fit of samples from this algorithm? Generate a sample of 1,000 2×2 random orthogonal matrices, and assess how well the sample follows a Haar uniform distribution.

4.11. Let A be an $n \times m$ random matrix whose elements all have continuous marginal distributions. (By "continuous distribution" we mean that any set of zero Lebesgue measure has zero probability.) In addition, assume that all nondegenerate conditional distributions of all elements are continuous (that is, assume that the joint distribution is nonsingular). Show that $\Pr(A \text{ is full rank}) = 1$.
Hint: Let $r = \min(n, m)$. If $r = 1$ it is proven. Otherwise, without loss of generality, assume $r = m$, and show that for any two a_{i*} and a_{j*} and any fixed $c_i, c_j \neq 0$, $\Pr(c_i a_{i*} + c_j a_{j*} = 0) = 0$. Then extend this to all sets of columns.

4.12. The Wishart distribution.
The Wishart distribution is a useful distribution of random matrices of sums of squares and cross products. It can be thought of as a multivariate generalization of the chi-squared distribution.
There are various ways of defining the Wishart distribution, each resulting in the same distribution. A natural way to define it is in terms of transformations of a multivariate normal distribution. Let

$$X_1, \ldots, X_n \overset{\text{iid}}{\sim} \mathrm{N}_d(\mu, \Sigma),$$

and let $X = [X_1 \ \ldots \ X_n]$. Now, let $W = XX^{\mathrm{T}}$.
First, assume $\mu = 0$. Then W has a central Wishart distribution with parameters d, n, where $d < n$, and Σ. We denote this as

$$W \sim \mathrm{W}_d(\Sigma, n),$$

and we often just refer to it as the Wishart distribution. (There are related distributions in which $d \geq n$, which we call "pseudo-Wishart" distributions.)

In the general case, we denote the distribution as

$$W \sim \mathrm{W}_d(\Sigma, n, \Delta),$$

where $\Delta = \mu\mu^{\mathrm{T}}$. If $\Delta \neq 0$; that is, if $\mu \neq 0$, we call the distribution the noncentral Wishart distribution.
The importance of the Wishart distribution is that the Wishart matrix can be used to make inferences on the variance-covariance matrix Σ of an underlying multivariate normal distribution.

a) Show that the probability density function for the (central) Wishart distribution is proportional to

$$\mathrm{e}^{-\mathrm{tr}(\Sigma^{-1}W/2)}|W|^{(n-d-1)/2},$$

and determine the constant of proportionality.

b) Let $W_1 \sim \mathrm{W}_d(\Sigma, n, \Delta_1)$ and let $W_2 \sim \mathrm{W}_d(\Sigma, m, \Delta_2)$ and be independent of W_1. Show that

$$W_1 + W_2 \sim \mathrm{W}_d(\Sigma, n + m, \Delta_1 + \Delta_2).$$

Hint: Using the definition of the Wishart distribution, you can assume that there are $n + m$ independent d-variate normal random variables.

c) Let $W \sim \mathrm{W}_d(\Sigma, n, \Delta)$ and let A be a fixed $q \times d$ matrix. Show that

$$AWA^{\mathrm{T}} \sim \mathrm{W}_q(A\Sigma A^{\mathrm{T}}, n, A\Delta A^{\mathrm{T}}).$$

d) Let $W \sim \mathrm{W}_d(\sigma^2 I, n, \Delta)$. Show that

$$\frac{1}{\sigma^2} W_{ii} \sim \chi^2_n(\Delta_{ii}).$$

(The PDF of the noncentral chi-squared distribution is given in equation (9.3) on page 402.)

5

Matrix Transformations and Factorizations

In most applications of linear algebra, problems are solved by transformations of matrices. A given matrix (which represents some transformation of a vector) is itself transformed. The simplest example of this is in solving the linear system $Ax = b$, where the matrix A represents a transformation of the vector x to the vector b. The matrix A is transformed through a succession of linear operations until x is determined easily by the transformed A and the transformed b. Each operation in the transformation of A is a pre- or postmultiplication by some other matrix. Each matrix formed as a product must be *equivalent* to A; therefore, in order to ensure this in general, each transformation matrix must be of full rank. In eigenproblems, we likewise perform a sequence of pre- or postmultiplications. In this case, each matrix formed as a product must be *similar* to A; therefore each transformation matrix must be orthogonal. We develop transformations of matrices by transformations on the individual rows or columns.

5.1 Factorizations

Given a matrix A, it is often useful to decompose A into the product of other matrices; that is, to form a factorization $A = BC$, where B and C are matrices. We refer to this as "matrix factorization", or sometimes as "matrix decomposition", although this latter term includes more general representations of the matrix, such as the spectral decomposition (page 154).

Most methods for eigenanalysis and for solving linear systems proceed by factoring the matrix, as we see in Chaps. 6 and 7.

In Chap. 3, we discussed some factorizations including

- the full rank factorization (equation (3.150)) of a general matrix,
- the equivalent canonical factorization (equation (3.155)) of a general matrix,

J.E. Gentle, *Matrix Algebra*, Springer Texts in Statistics,
DOI 10.1007/978-3-319-64867-5_5

- the Schur factorization (equation (3.244)) of a square matrix,
- the similar canonical factorization (equation (3.247)) or "diagonal factorization" of a diagonalizable matrix (which is necessarily square),
- the orthogonally similar canonical factorization (equation (3.252)) of a symmetric matrix (which is necessarily diagonalizable),
- the square root (equation (3.273)) of a nonnegative definite matrix (which is necessarily symmetric), and
- the singular value factorization (equation (3.276)) of a general matrix.

In this chapter we describe three additional factorizations:

- the LU (and LR and LDU) factorization of a general matrix,
- the QR factorization of a general matrix, and
- the Cholesky factorization of a nonnegative definite matrix.

These factorizations are useful both in theory and in practice.

5.2 Computational Methods: Direct and Iterative

In our previous discussions of matrix factorizations and other operations, we have shown derivations that may indicate computational methods, but we have not specified the computational details. There are many important computational issues, some of which we will discuss in Part III. At this point, again without getting into the details, we want to note a fundamental difference in the types of computational methods.

The developments of the full rank factorization (equation (3.150)) and the equivalent canonical factorization (equation (3.155)) were constructive, and indeed those factorizations could be computed following those constructions. On the other hand, our developments of the diagonalizing transformations, such as the orthogonally similar canonical factorization (equation (3.252)), and other factorizations related to eigenvalues, such as the singular value factorization (equation (3.276)), were not constructive.

As it turns out, the factorizations involving eigenvalues or singular values cannot in general be computed using a finite set of arithmetic operations. If they could be, then the characteristic polynomial equation could be solved in a finite set of arithmetic operations, and the Abel-Ruffini theorem states that such a solution does not exist for polynomials of degree five or higher. For factorizations of this type we must use *iterative methods*, at least to get the eigenvalues or singular values. We will describe some of those methods in Chap. 7. (In this chapter I do one factorization based on an eigendecomposition. It is the square root, described in Sect. 5.9.1. This is because it so naturally goes with a Cholesky factorization of a nonnegative definite matrix.)

In this chapter we first discuss some transformations and important factorizations that can be carried out in a finite number of arithmetic steps; that is, by the use of *direct* methods. The factorizations themselves can be used iteratively; indeed, as we will discuss in Chap. 7, the QR is the most important factorization used iteratively to obtain eigenvalues or singular values.

A factorization of a given matrix A is generally effected by a series of pre- or postmultiplications by transformation matrices with simple and desirable properties. One such transformation matrix is the Gaussian matrix, G_{ij} of equation (3.63) or $\widetilde{G}_{ij}$ of equation (3.65) on page 84. Another important class of transformation matrices are orthogonal matrices. Orthogonal transformation matrices have some desirable properties. In this chapter, before discussing factorizations, we first consider some general properties of various types of transformations, and then we describe two specific types of orthogonal transformations, Householder reflections (Sect. 5.4) and Givens rotations (Sect. 5.5). As we will see, the Householder reflections are very similar to the Gram-Schmidt transformations that we discussed beginning on page 38.

5.3 Linear Geometric Transformations

In many important applications of linear algebra, a vector represents a point in space, with each element of the vector corresponding to an element of a coordinate system, usually a Cartesian system. A set of vectors describes a geometric object, such as a polyhedron or a Lorentz cone, as on page 44. Algebraic operations can be thought of as geometric transformations that rotate, deform, or translate the object. While these transformations are often used in the two or three dimensions that correspond to the easily perceived physical space, they have similar applications in higher dimensions. Thinking about operations in linear algebra in terms of the associated geometric operations often provides useful intuition.

A linear transformation of a vector x is effected by multiplication by a matrix A. Any $n \times m$ matrix A is a function or transformation from $\mathcal{V}_1$ to $\mathcal{V}_2$, where $\mathcal{V}_1$ is a vector space of order m and $\mathcal{V}_2$ is a vector space of order n.

5.3.1 Invariance Properties of Linear Transformations

An important characteristic of a transformation is what it leaves *unchanged*; that is, its *invariance properties* (see Table 5.1). All of the transformations we will discuss are *linear transformations* because they preserve straight lines. A set of points that constitute a straight line is transformed into a set of points that constitute a straight line.

As mentioned above, reflections and rotations are orthogonal transformations, and we have seen that an orthogonal transformation preserves lengths of vectors (equation (3.286)). We will also see that an orthogonal transformation preserves angles between vectors (equation (5.1)). A transformation that preserves lengths and angles is called an *isometric transformation*. Such a transformation also preserves areas and volumes.

Another isometric transformation is a *translation*, which is essentially the addition of another vector (see Sect. 5.3.5).

Table 5.1. Invariance properties of transformations

Transformation	Preserves
Linear	Lines
Projective	Lines
Affine	Lines, Collinearity
Shearing	Lines, Collinearity
Scaling	Lines, Angles (and, hence, Collinearity)
Rotation	Lines, Angles, Lengths
Reflection	Lines, Angles, Lengths
Translation	Lines, Angles, Lengths

A transformation that preserves angles is called an *isotropic transformation*. An example of an isotropic transformation that is not isometric is a uniform scaling or dilation transformation, $\tilde{x} = ax$, where a is a scalar.

The transformation $\tilde{x} = Ax$, where A is a diagonal matrix with not all elements the same, does not preserve angles; it is an *anisotropic* scaling. Another anisotropic transformation is a *shearing transformation*, $\tilde{x} = Ax$, where A is the same as an identity matrix, except for a single row or column that has a one on the diagonal but nonzero, possibly constant, elements in the other positions; for example,

$$\begin{bmatrix} 1 & 0 & a_1 \\ 0 & 1 & a_1 \\ 0 & 0 & 1 \end{bmatrix}.$$

Although they do not preserve angles, both anisotropic scaling and shearing transformations preserve parallel lines. A transformation that preserves parallel lines is called an *affine transformation*. Preservation of parallel lines is equivalent to preservation of collinearity, and so an alternative characterization of an affine transformation is one that preserves collinearity. More generally, we can combine nontrivial scaling and shearing transformations to see that the transformation Ax for any nonsingular matrix A is affine. It is easy to see that addition of a constant vector to all vectors in a set preserves collinearity within the set, so a more general affine transformation is $\tilde{x} = Ax + t$ for a nonsingular matrix A and a vector t.

A *projective transformation*, which uses the homogeneous coordinate system of the projective plane (see Sect. 5.3.5), preserves straight lines, but does not preserve parallel lines. Projective transformations are very useful in computer graphics. In those applications we do not always want parallel lines to project onto the display plane as parallel lines.

5.3.2 Transformations by Orthogonal Matrices

We defined orthogonal matrices and considered some basic properties on page 132. Orthogonal matrices are not necessarily square; they may have

more rows than columns or may have fewer. In the following, we will consider only orthogonal matrices with at least as many rows as columns; that is, if Q is an orthogonal transformation matrix, then $Q^{\mathrm{T}}Q = I$. This means that an orthogonal matrix is of full rank. Of course, many useful orthogonal matrices are square (and, obviously, nonsingular). There are many types of orthogonal transformation matrices. As noted previously, permutation matrices are square orthogonal matrices, and we have used them extensively in rearranging the columns and/or rows of matrices.

As we stated, transformations by orthogonal matrices preserve lengths of vectors. Orthogonal transformations also preserve angles between vectors, as we can easily see. If Q is an orthogonal matrix, then, for vectors x and y, we have

$$\langle Qx, Qy\rangle = (Qx)^{\mathrm{T}}(Qy) = x^{\mathrm{T}}Q^{\mathrm{T}}Qy = x^{\mathrm{T}}y = \langle x, y\rangle,$$

and hence,

$$\arccos\left(\frac{\langle Qx, Qy\rangle}{\|Qx\|_2\,\|Qy\|_2}\right) = \arccos\left(\frac{\langle x, y\rangle}{\|x\|_2\ \|y\|_2}\right). \tag{5.1}$$

Thus, orthogonal transformations preserve angles.

We have seen that if Q is an orthogonal matrix and

$$B = Q^{\mathrm{T}}AQ,$$

then A and B have the same eigenvalues (and A and B are said to be orthogonally similar). By forming the transpose, we see immediately that the transformation $Q^{\mathrm{T}}AQ$ preserves symmetry; that is, if A is symmetric, then B is symmetric.

From equation (3.287), we see that $\|Q^{-1}\|_2 = 1$. This has important implications for the accuracy of numerical computations. (Using computations with orthogonal matrices will not make problems more "ill-conditioned".)

We often use orthogonal transformations that preserve lengths and angles while rotating $\mathbb{R}^n$ or reflecting regions of $\mathbb{R}^n$. The transformations are appropriately called rotators and reflectors, respectively.

5.3.3 Rotations

The simplest rotation of a vector can be thought of as the rotation of a plane defined by two coordinates about the other principal axes. Such a rotation changes two elements of all vectors in that plane and leaves all the other elements, representing the other coordinates, unchanged. This rotation can be described in a two-dimensional space defined by the coordinates being changed, without reference to the other coordinates.

Consider the rotation of the vector x through the angle θ into the vector $\tilde{x}$. The length is preserved, so we have $\|\tilde{x}\| = \|x\|$. Referring to Fig. 5.1, we can write

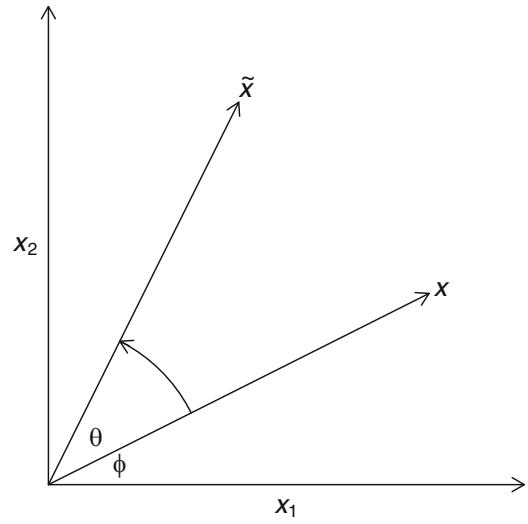


Figure 5.1. Rotation of x

$$\begin{aligned}\tilde{x}_1 &= \|x\| \cos(\phi + \theta),\\ \tilde{x}_2 &= \|x\| \sin(\phi + \theta).\end{aligned}$$

Now, from elementary trigonometry, we know

$$\begin{aligned}\cos(\phi + \theta) &= \cos\phi\cos\theta - \sin\phi\sin\theta,\\ \sin(\phi + \theta) &= \sin\phi\cos\theta + \cos\phi\sin\theta.\end{aligned}$$

Because $\cos\phi = x_1/\|x\|$ and $\sin\phi = x_2/\|x\|$, we can combine these equations to get

$$\begin{aligned}\tilde{x}_1 &= x_1\cos\theta - x_2\sin\theta,\\ \tilde{x}_2 &= x_1\sin\theta + x_2\cos\theta.\end{aligned} \tag{5.2}$$

Hence, multiplying x by the orthogonal matrix

$$\begin{bmatrix}\cos\theta & -\sin\theta\\ \sin\theta & \cos\theta\end{bmatrix} \tag{5.3}$$

performs the rotation of x.

This idea easily extends to the rotation of a plane formed by two coordinates about all of the other (orthogonal) principal axes. By convention, we assume clockwise rotations for axes that increase in the direction from which the system is viewed. For example, if there were an x_3 axis in Fig. 5.1, it would point toward the viewer. (This is called a "right-hand" coordinate system, because if the viewer's right-hand fingers point in the direction of the rotation, the thumb points toward the viewer.)

The rotation matrix about principal axes is the same as an identity matrix with two diagonal elements changed to $\cos\theta$ and the corresponding off-diagonal elements changed to $\sin\theta$ and $-\sin\theta$.

To rotate a 3-vector, x, about the x_2 axis in a right-hand coordinate system, we would use the rotation matrix

$$\begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}.$$

A rotation of any hyperplane in n-space can be formed by n successive rotations of hyperplanes formed by two principal axes. (In 3-space, this fact is known as *Euler's rotation theorem*. We can see this to be the case, in 3-space or in general, by construction.)

A rotation of an arbitrary plane can be defined in terms of the direction cosines of a vector in the plane before and after the rotation. In a coordinate geometry, rotation of a plane can be viewed equivalently as a rotation of the coordinate system in the opposite direction. This is accomplished by rotating the unit vectors e_i into $\tilde{e}_i$.

A special type of transformation that rotates a vector to be perpendicular to a principal axis is called a Givens rotation. We discuss the use of this type of transformation in Sect. 5.5 on page 238. Another special rotation is the "reflection" of a vector about another vector. We discuss this kind of rotation next.

5.3.4 Reflections

Let u and v be orthonormal vectors, and let x be a vector in the space spanned by u and v, so

$$x = c_1 u + c_2 v$$

for some scalars c_1 and c_2. The vector

$$\tilde{x} = -c_1 u + c_2 v \tag{5.4}$$

is a *reflection* of x through the line defined by the vector v, or $u^{\perp}$. This reflection is a rotation in the plane defined by u and v through an angle of twice the size of the angle between x and v.

The form of $\tilde{x}$ of course depends on the vector v and its relationship to x. In a common application of reflections in linear algebraic computations, we wish to rotate a given vector into a vector collinear with a coordinate axis; that is, we seek a reflection that transforms a vector

$$x = (x_1, x_2, \ldots, x_n)$$

into a vector collinear with a unit vector,

$$\begin{aligned}\tilde{x} &= (0, \ldots, 0, \tilde{x}_i, 0, \ldots, 0) \\ &= \pm\|x\|_2 e_i. \end{aligned} \tag{5.5}$$

Geometrically, in two dimensions we have the picture shown in Fig. 5.2, where $i = 1$. Which vector that x is rotated through (that is, which is u and which is v) depends on the choice of the sign in $\pm\|x\|_2$. The choice that was made yields the $\tilde{x}$ shown in the figure, and from the figure, this can be seen to be correct. Note that

$$v = \frac{1}{|2c_2|}(x + \tilde{x})$$

If the opposite choice is made, we get the $\tilde{\tilde{x}}$ shown. In the simple two-dimensional case, this is equivalent to reversing our choice of u and v.

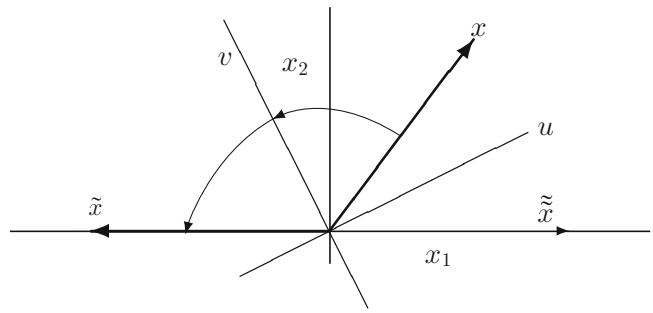


Figure 5.2. Reflections of x about v (or $u^{\perp}$) and about u

To accomplish this special rotation of course, we first choose an appropriate vector about which to reflect our given vector, and then perform the rotation. We will describe this process in Sect. 5.4 below.

5.3.5 Translations: Homogeneous Coordinates

A translation of a vector is a relatively simple transformation in which the vector is transformed into a parallel vector. It involves a type of addition of vectors. Rotations, as we have seen, and other geometric transformations such as shearing, as we have indicated, involve multiplication by an appropriate matrix. In applications where several geometric transformations are to be made, it would be convenient if translations could also be performed by matrix multiplication. This can be done by using *homogeneous coordinates*.

Homogeneous coordinates, which form the natural coordinate system for projective geometry, have a very simple relationship to Cartesian coordinates.

The point with Cartesian coordinates $(x_1, x_2, \ldots, x_d)$ is represented in homogeneous coordinates as $(x_0^{\mathrm{h}}, x_1^{\mathrm{h}}, \ldots, x_d^{\mathrm{h}})$, where, for arbitrary x_0^{h} not equal to zero, $x_1^{\mathrm{h}} = x_0^{\mathrm{h}} x_1$, and so on. Because the point is the same, the two different symbols represent the same thing, and we have

$$(x_1, \ldots, x_d) = (x_0^{\mathrm{h}}, x_1^{\mathrm{h}}, \ldots, x_d^{\mathrm{h}}). \tag{5.6a}$$

Alternatively, the hyperplane coordinate may be added at the end, and we have

$$(x_1, \ldots, x_d) = (x_1^{\mathrm{h}}, \ldots, x_d^{\mathrm{h}}, x_0^{\mathrm{h}}). \tag{5.6b}$$

Each value of x_0^{h} corresponds to a hyperplane in the ordinary Cartesian coordinate system. The most common choice is $x_0^{\mathrm{h}} = 1$, and so $x_i^{\mathrm{h}} = x_i$. The special plane $x_0^{\mathrm{h}} = 0$ does not have a meaning in the Cartesian system, but in projective geometry it corresponds to a hyperplane at infinity.

We can easily effect the translation $\tilde{x} = x + t$ by first representing the point x as $(1, x_1, \ldots, x_d)$ and then multiplying by the $(d+1) \times (d+1)$ matrix

$$T = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ t_1 & 1 & \cdots & 0 \\ & \cdots & & \\ t_d & 0 & \cdots & 1 \end{bmatrix}.$$

We will use the symbol x^{h} to represent the vector of corresponding homogeneous coordinates:

$$x^{\mathrm{h}} = (1, x_1, \ldots, x_d).$$

We must be careful to distinguish the point x from the vector that represents the point. In Cartesian coordinates, there is a natural correspondence and the symbol x representing a point may also represent the vector $(x_1, \ldots, x_d)$. The vector of homogeneous coordinates of the result Tx^{h} corresponds to the Cartesian coordinates of $\tilde{x}$, $(x_1 + t_1, \ldots, x_d + t_d)$, which is the desired result.

Homogeneous coordinates are used extensively in computer graphics not only for the ordinary geometric transformations but also for projective transformations, which model visual properties. Hill and Kelley (2006) describe many of these applications. See Exercise 5.2 for a simple example.

5.4 Householder Transformations (Reflections)

We have briefly discussed geometric transformations that reflect a vector through another vector. We now consider some properties and uses of these transformations.

Consider the problem of reflecting x through the vector v. As before, we assume that u and v are orthonormal vectors and that x lies in a space spanned by u and v, and $x = c_1 u + c_2 v$. Form the matrix

$$H = I - 2uu^{\mathrm{T}}, \tag{5.7}$$

and note that

$$\begin{aligned} Hx &= c_1 u + c_2 v - 2c_1 uu^{\mathrm{T}} u - 2c_2 uu^{\mathrm{T}} v \\ &= c_1 u + c_2 v - 2c_1 u^{\mathrm{T}} uu - 2c_2 u^{\mathrm{T}} vu \\ &= -c_1 u + c_2 v \\ &= \tilde{x}, \end{aligned}$$

as in equation (5.4). The matrix H is a *reflector*; it has transformed x into its reflection $\tilde{x}$ about v.

A reflection is also called a Householder reflection or a Householder transformation, and the matrix H is called a Householder matrix or a Householder reflector. The following properties of H are immediate:

- $Hu = -u$.
- $Hv = v$ for any v orthogonal to u.
- $H = H^{\mathrm{T}}$ (symmetric).
- $H^{\mathrm{T}} = H^{-1}$ (orthogonal).

Because H is orthogonal, if $Hx = \tilde{x}$, then $\|x\|_2 = \|\tilde{x}\|_2$ (see equation (3.286)), so $\tilde{x}_1 = \pm\|x\|_2$.

The matrix uu^{T} is symmetric, idempotent, and of rank 1. A transformation by a matrix of the form $A - vw^{\mathrm{T}}$ is often called a "rank-one" update, because vw^{T} is of rank 1. Thus, a Householder reflection is a special rank-one update.

5.4.1 Zeroing All Elements But One in a Vector

The usefulness of Householder reflections results from the fact that it is easy to construct a reflection that will transform a vector x into a vector $\tilde{x}$ that has zeros in all but one position, as in equation (5.5). To construct the reflector of x into $\tilde{x}$, we first need to determine a vector v as in Fig. 5.2 about which to reflect x. That vector is merely

$$x + \tilde{x}.$$

Because $\|\tilde{x}\|_2 = \|x\|_2$, we know $\tilde{x}$ to within the sign; that is,

$$\tilde{x} = (0, \ldots, 0, \pm\|x\|_2, 0, \ldots, 0).$$

We choose the sign so as not to add quantities of different signs and possibly similar magnitudes. (See the discussions of catastrophic cancellation below and beginning on page 488, in Chap. 10.) Hence, we have

$$q = (x_1, \ \ldots, \ x_{i-1}, \ x_i + \mathrm{sign}(x_i)\|x\|_2, \ x_{i+1}, \ \ldots, \ x_n). \tag{5.8}$$

We normalize this to obtain

$$u = q/\|q\|_2, \tag{5.9}$$

and finally form

$$H = I - 2uu^{\mathrm{T}}. \tag{5.10}$$

Consider, for example, the vector

$$x = (3, 1, 2, 1, 1),$$

which we wish to transform into

$$\tilde{x} = (\tilde{x}_1, 0, 0, 0, 0).$$

We have

$$\|x\| = 4,$$

so we form the vector

$$u = \frac{1}{\sqrt{56}}(7, 1, 2, 1, 1)$$

and the Householder reflector

$$H = I - 2uu^{\mathrm{T}}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} - \frac{1}{28} \begin{bmatrix} 49 & 7 & 14 & 7 & 7 \\ 7 & 1 & 2 & 1 & 1 \\ 14 & 2 & 4 & 2 & 2 \\ 7 & 1 & 2 & 1 & 1 \\ 7 & 1 & 2 & 1 & 1 \end{bmatrix}$$

$$= \frac{1}{28} \begin{bmatrix} -21 & -7 & -14 & -7 & -7 \\ -7 & 27 & -2 & -1 & -1 \\ -14 & -2 & 24 & -2 & -2 \\ -7 & -1 & -2 & 27 & -1 \\ -7 & -1 & -2 & -1 & 27 \end{bmatrix}$$

to yield $Hx = (-4, 0, 0, 0, 0)$.

The procedure described by equations (5.8), (5.9), and (5.10), zeroes out all but the i^{th} element of the given vector. We could of course modify the reflector matrix so that certain elements of the reflected vector are unchanged.

We will consider these reflections further in Sect. 5.8.8, beginning on page 252.

5.4.2 Computational Considerations

Notice that if we had chosen $\tilde{x}$ as $(-4, 0, 0, 0, 0)$, then u would have been $(-1, 1, 2, 1, 1)/\sqrt{8}$, and Hx would have been $(4, 0, 0, 0, 0)$, and our objective would also have been achieved. In this case, there would have been no numerical rounding problems. If, however, x were such that $x_1 \approx -\|x\|_2$ in

the addition $x_1 + \|x\|_2$, "catastrophic cancellation" would occur. For example, if $x_1 = -3$, and $\|x\|$ is computed as 3.0000002, the computation of $x_1 + \|x\|_2$ would lose seven significant digits. Of course, it can be the case that $x_1 \approx -\|x\|_2$, only if $x_2 \approx x_3 \approx \cdots \approx 0$. Nevertheless, we should perform computations in such a way as to protect against the worst cases, especially if it is easy to do so.

Standard Householder computations are performed generally as indicated above, but there may be minor variations in the order of performing the computations that take advantage of specific computer architectures. There are variants of the Householder transformations that are more efficient by taking advantage of such architectures as a cache memory or a bank of floating-point registers whose contents are immediately available to the computational unit.

5.5 Givens Transformations (Rotations)

We have briefly discussed geometric transformations that rotate a vector in such a way that a specified element becomes 0 and only one other element in the vector is changed. Such a method may be particularly useful if only part of the matrix to be transformed is available. These transformations are called *Givens transformations*, or *Givens rotations*, or sometimes *Jacobi transformations*.

The basic idea of the rotation, which is a special case of the rotations discussed on page 231, can be seen in the case of a vector of length 2. Given the vector $x = (x_1, x_2)$, we wish to rotate it to $\tilde{x} = (\tilde{x}_1, 0)$. As with a reflection, in the rotation we also have $\tilde{x}_1 = \|x\|$. Geometrically, we have the picture shown in Fig. 5.3.

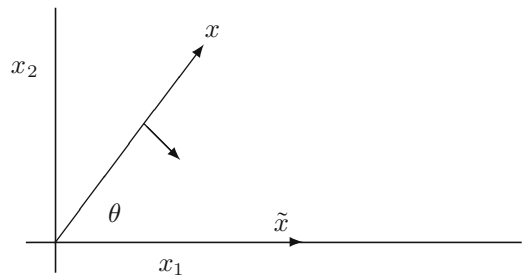


Figure 5.3. Rotation of x onto a coordinate axis

It is easy to see that the orthogonal matrix

$$Q = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \tag{5.11}$$

will perform this rotation of x if $\cos\theta = x_1/r$ and $\sin\theta = x_2/r$, where $r = \|x\| = \sqrt{x_1^2 + x_2^2}$. (This is the same matrix as in equation (5.3), except that

the rotation is in the opposite direction.) Notice that θ is not relevant; we only need real numbers c and s such that $c^2 + s^2 = 1$.

We have

$$\begin{aligned}\tilde{x}_1 &= \frac{x_1^2}{r} + \frac{x_2^2}{r} \\ &= \|x\|,\end{aligned}$$

$$\begin{aligned}\tilde{x}_2 &= -\frac{x_2 x_1}{r} + \frac{x_1 x_2}{r} \\ &= 0;\end{aligned}$$

that is,

$$Q \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} \|x\| \\ 0 \end{pmatrix}.$$

5.5.1 Zeroing One Element in a Vector

As with the Householder reflection that transforms a vector

$$x = (x_1, x_2, x_3, \ldots, x_n)$$

into a vector

$$\tilde{x}_H = (\tilde{x}_{H1}, 0, 0, \ldots, 0),$$

it is easy to construct a Givens rotation that transforms x into

$$\tilde{x}_G = (\tilde{x}_{G1}, 0, x_3, \ldots, x_n).$$

We can construct an orthogonal matrix G_{pq} similar to that shown in equation (5.11) that will transform the vector

$$x = (x_1, \ldots, x_p, \ldots, x_q, \ldots, x_n)$$

into

$$\tilde{x} = (x_1, \ldots, \tilde{x}_p, \ldots, 0, \ldots, x_n).$$

The orthogonal matrix that will do this is

$$G_{pq}(\theta) = \begin{bmatrix}
1 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\
0 & 1 & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\
 & & \ddots & & & & & & & & & \\
0 & 0 & \cdots & 1 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\
0 & 0 & \cdots & 0 & c & 0 & \cdots & 0 & s & 0 & \cdots & 0 \\
0 & 0 & \cdots & 0 & 0 & 1 & \cdots & 0 & 0 & 0 & \cdots & 0 \\
 & & & & & & \ddots & & & & & \\
0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 1 & 0 & 0 & \cdots & 0 \\
0 & 0 & \cdots & 0 & -s & 0 & \cdots & 0 & c & 0 & \cdots & 0 \\
0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 & 1 & \cdots & 0 \\
 & & & & & & & & & & \ddots & \\
0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 1
\end{bmatrix}, \tag{5.12}$$

where the entries in the p^{th} and q^{th} rows and columns are

$$c = \frac{x_p}{r}$$

and

$$s = \frac{x_q}{r},$$

where $r = \sqrt{x_p^2 + x_q^2}$. A rotation matrix is the same as an identity matrix with four elements changed.

Considering x to be the p^{th} column in a matrix X, we can easily see that $G_{pq}X$ results in a matrix with a zero as the q^{th} element of the p^{th} column, and all except the p^{th} and q^{th} rows and columns of $G_{pq}X$ are the same as those of X.

5.5.2 Givens Rotations That Preserve Symmetry

If X is a symmetric matrix, we can preserve the symmetry by a transformation of the form $Q^{\mathrm{T}}XQ$, where Q is any orthogonal matrix. The elements of a Givens rotation matrix that is used in this way and with the objective of forming zeros in two positions in X simultaneously would be determined in the same way as above, but the elements themselves would not be the same. We illustrate that below, while at the same time considering the problem of transforming a value into something other than zero.

5.5.3 Givens Rotations to Transform to Other Values

Consider a symmetric matrix X that we wish to transform to the symmetric matrix $\widetilde{X}$ that has all rows and columns except the p^{th} and q^{th} the same as those in X, and we want a specified value in the $(p,p)^{\text{th}}$ position of $\widetilde{X}$, say $\widetilde{x}_{pp} = a$. We seek a rotation matrix G such that $\widetilde{X} = G^{\mathrm{T}}XG$. We have

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} x_{pp} & x_{pq} \\ x_{pq} & x_{qq} \end{bmatrix} \begin{bmatrix} c & s \\ -s & c \end{bmatrix} = \begin{bmatrix} a & \tilde{x}_{pq} \\ \tilde{x}_{pq} & \tilde{x}_{qq} \end{bmatrix} \tag{5.13}$$

and

$$c^2 + s^2 = 1.$$

Hence

$$a = c^2 x_{pp} - 2csx_{pq} + s^2 x_{qq}. \tag{5.14}$$

Writing $t = s/c$ (the tangent), we have the quadratic

$$(x_{qq} - a)t^2 - 2x_{pq}t + x_{pp} - a = 0 \tag{5.15}$$

with roots

$$t = \frac{x_{pq} \pm \sqrt{x_{pq}^2 - (x_{pp} - a)(x_{qq} - a)}}{(x_{qq} - a)}. \tag{5.16}$$

The roots are real if and only if

$$x_{pq}^2 \geq (x_{pp} - a)(x_{qq} - a).$$

If the roots in equation (5.16) are real, we choose the nonnegative one. (See the discussion of equation (10.3) on page 488.) We then form

$$c = \frac{1}{\sqrt{1 + t^2}} \tag{5.17}$$

and

$$s = ct. \tag{5.18}$$

The rotation matrix G formed from c and s will transform X into $\widetilde{X}$.

5.5.4 Fast Givens Rotations

Often in applications we need to perform a succession of Givens transformations. The overall number of computations can be reduced using a succession of "fast Givens rotations". We write the matrix Q in equation (5.11) as CT,

$$\begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ 0 & \cos\theta \end{bmatrix} \begin{bmatrix} 1 & \tan\theta \\ -\tan\theta & 1 \end{bmatrix}, \tag{5.19}$$

and instead of working with matrices such as Q, which require four multiplications and two additions, we work with matrices such as T, involving the tangents, which require only two multiplications and two additions. After a number of computations with such matrices, the diagonal matrices of the form of C are accumulated and multiplied together.

The diagonal elements in the accumulated C matrices in the fast Givens rotations can become widely different in absolute values, so to avoid excessive loss of accuracy, it is usually necessary to rescale the elements periodically.

5.6 Factorization of Matrices

It is often useful to represent a matrix A in a factored form,

$$A = BC,$$

where B and C have some specified desirable properties, such as being orthogonal or being triangular. We generally seek B and C such that B and C have useful properties for some particular aspect of the problem being addressed.

Most direct methods of solving linear systems (discussed in Chap. 6) are based on factorizations (or, equivalently, "decompositions") of the matrix of coefficients. Matrix factorizations are also performed for reasons other than to solve a linear system, such as in eigenanalysis (discussed in Chap. 7).

Notice an indeterminacy in the factorization $A = BC$; if B and C are factors of A, then so are $-B$ and $-C$. This indeterminacy includes not only the negatives of the matrices themselves, but also the negatives of various rows and columns properly chosen. More generally, if D and E are matrices such that $DE = I_m$, where m is the number of columns in B and rows in C, then $A = BDEC$, and so A can be factored as the product of BD and EC. Hence, in general, a factorization is not unique. If restrictions are placed on certain properties of the factors, however, then under those restrictions, the factorizations may be unique. Also, if one factor is given, the other factor may be unique. (For example, in the case of nonsingular matrices, we can see this by taking the inverse.)

Invertible transformations result in a factorization of a matrix. For an $n \times k$ matrix B, if D is a $k \times n$ matrix such that $BD = I_n$, then a given $n \times m$ matrix A can be factorized as $A = BDA = BC$, where $C = DA$.

Some important matrix factorizations were listed at the beginning of this chapter. Of those, we have already discussed the full rank and the diagonal canonical factorizations in Chap. 3. Also in Chap. 3, we have briefly described the orthogonally similar canonical factorization and the SVD. We will discuss these factorizations, which require iterative methods, further in Chap. 7. In the next few sections we will introduce the LU, LDU, QR, and Cholesky factorizations. We will also describe the square root factorization, even though it uses eigenvalues, which require the iterative methods.

Matrix factorizations are generally performed by a sequence of full-rank transformations and their inverses.

5.7 LU and LDU Factorizations

For any matrix (whether square or not) that can be expressed as LU, where L is lower triangular (or lower trapezoidal) and U is upper triangular (or upper trapezoidal), the product LU is called the *LU factorization*. We also generally restrict either L or U to have 0s or 1s on the diagonal. If an LU factorization exists, it is clear that either L or U (but not necessarily both) can be made to have only 1s and 0s on its diagonal.

If an LU factorization exists, both the lower triangular matrix, L, and the upper triangular matrix, U, can be made to have only 1s or 0s on their diagonals (that is, be made to be unit lower triangular or unit upper triangular) by putting the products of any non-unit diagonal elements into a diagonal matrix D and then writing the factorization as LDU, where now L and U are unit triangular matrices (that is, matrices with 1s on the diagonal). This is called the *LDU factorization*.

If a matrix is not square, or if the matrix is not of full rank, in its LU decomposition, L and/or U may have zero diagonal elements or will be of trapezoidal form. An example of a singular matrix and its LU factorization is

$$\begin{aligned} A &= \begin{bmatrix} 0 & 3 & 2 \\ 0 & 0 & 0 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 3 & 2 \\ 0 & 0 & 0 \end{bmatrix} \\ &= LU. \end{aligned} \tag{5.20}$$

In this case, U is an upper trapezoidal matrix.

5.7.1 Properties: Existence

Existence and uniqueness of an LU factorization (or LDU factorization) are interesting questions. It is neither necessary nor sufficient that a matrix be nonsingular for it to have an LU factorization. The example above shows the LU factorization for a matrix not of full rank. Furthermore, a full rank matrix does not necessarily have an LU factorization, as we see next.

An example of a nonsingular matrix that does not have an LU factorization is an identity matrix with permuted rows or columns:

$$B = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}. \tag{5.21}$$

The conditions for the existence of an LU factorization are not so easy to state (see Harville 1997, for example), but in practice, as we will see, the question is not very relevant. First, however, we will consider a matrix that is guaranteed to have an LU factorization, and show one method of obtaining it.

A sufficient condition for an $n \times m$ matrix A to have an LU factorization is that for $k = 1, 2, \ldots, \min(n, m)$, each $k \times k$ principal submatrix of A be nonsingular.

The proof is by construction. We assume that all principal submatrices are nonsingular. This means that $a_{11} \neq 0$, and so the Gaussian matrix G_{11} exists. (See equation (3.63) on page 84, where we also set the notation.) We multiply A by G_{11}, obtaining

$$G_{11}A = A^{(1)},$$

in which $a_{i1}^{(1)} = 0$ for $i = 2, \ldots, n$ and $a_{22}^{(1)} \neq 0$ (otherwise the 2×2 principal submatrix would be singular, which by assumption it is not).

Since $a_{22}^{(1)} \neq 0$, the Gaussian matrix G_{22} exists, and now we multiply $G_{11}A$ by G_{22}, obtaining

$$G_{22}G_{11}A = A^{(2)},$$

in which $a_{i2}^{(2)} = 0$ for $i = 3, \ldots, n$ and $a_{33}^{(2)} \neq 0$ as before. (All $a_{i1}^{(1)}$ are unchanged.)

We continue in this way for $k = \min(n, m)$ steps, to obtain

$$G_{kk} \cdots G_{22} G_{11} A = A^{(k)}, \tag{5.22}$$

in which $A^{(k)}$ is upper triangular, and the matrix $G_{kk} \cdots G_{22} G_{11}$ is lower triangular because each matrix in the product is lower triangular. (Note that if $m > n$, $G_{kk} = E_n(1/a_{nn}^{(n-1)})$.)

Furthermore each matrix in the product $G_{kk} \cdots G_{22} G_{11}$ is nonsingular, and the matrix $G_{11}^{-1} G_{22}^{-1} \cdots G_{kk}^{-1}$ is lower triangular (see equation (3.64) on page 84). We complete the factorization by multiplying both sides of equation (5.22) by $G_{11}^{-1} G_{22}^{-1} \cdots G_{kk}^{-1}$:

$$\begin{aligned} A &= G_{11}^{-1} G_{22}^{-1} \cdots G_{kk}^{-1} A^{(k)} \\ &= L A^{(k)} \\ &= LU. \end{aligned} \tag{5.23}$$

Hence, we see that an $n \times m$ matrix A has an LU factorization if for $k = 1, 2, \ldots, \min(n, m)$, each $k \times k$ principal submatrix of A is nonsingular.

The elements on the diagonal of $A^{(k)}$, that is, U, are all 1s; hence, we note a useful property for square matrices. If the matrix is square ($k = n$), then

$$\det(A) = \det(L)\det(U) = l_{11} l_{22} \cdots l_{nn}. \tag{5.24}$$

An alternate construction leaves 1s on the diagonal of L, and then the determinant of A is the product of the diagonal elements of U. One way of achieving this factorization, again in the case in which the principal submatrices are all nonsingular, is to form the matrices

$$L_j = E_{n,j}\left(-a_{n,j}^{(j-1)} / a_{jj}^{(j-1)}\right) \cdots E_{j+1,j}\left(-a_{j+1,j}^{(j-1)} / a_{jj}^{(j-1)}\right); \tag{5.25}$$

that is,

$$L_j = \begin{bmatrix} 1 \cdots & 0 & 0 \cdots 0 \\ \ddots & & \\ 0 \cdots & 1 & 0 \cdots 0 \\ 0 \cdots & -\frac{a_{j+1,j}^{(j-1)}}{a_{jj}^{(j-1)}} & 1 \cdots 0 \\ & & \ddots \\ 0 \cdots & -\frac{a_{nj}^{(j-1)}}{a_{jj}^{(j-1)}} & 0 \cdots 1 \end{bmatrix}. \tag{5.26}$$

(See page 86 for the notation "$E_{pq}(a)$", representing the elementary axpy matrix.)

Each L_j is nonsingular, with a determinant of 1. The whole process of forward reduction can be expressed as a matrix product,

$$U = L_{k-1}L_{k-2}\dots L_2L_1A, \tag{5.27}$$

and by the way we have performed the forward reduction, U is an upper triangular matrix. The matrix $L_{k-1}L_{k-2}\dots L_2L_1$ is nonsingular and is unit lower triangular (all 1s on the diagonal). Its inverse therefore is also unit lower triangular. Call its inverse L; that is,

$$L = (L_{k-1}L_{k-2}\dots L_2L_1)^{-1}. \tag{5.28}$$

Thus, the forward reduction is equivalent to expressing A as LU,

$$A = LU. \tag{5.29}$$

In this case, the diagonal elements of the lower triangular matrix L in the LU factorization are all 1s by the method of construction, and if A is square,

$$\det(A) = u_{11}u_{22}\cdots u_{nn}.$$

Even if the principal submatrices of a matrix are not nonsingular, the matrix may have an LU decomposition, and it may be computable using a sequence of Gaussian matrices as we did above. Consider, for example,

$$C = \begin{bmatrix} 2 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}, \tag{5.30}$$

whose 0 in the $(2,2)$ position violates the nonsingularity condition. After three Gaussian steps as above, we have

$$\begin{bmatrix} \frac{1}{2} & 0 & 0 \\ 1 & -2 & 0 \\ -1 & 1 & 1 \end{bmatrix} C = \begin{bmatrix} 1 & \frac{1}{2} & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

from which we get

$$C = \begin{bmatrix} 2 & 0 & 0 \\ 1 & -\frac{1}{2} & 0 \\ 1 & \frac{1}{2} & 1 \end{bmatrix} \begin{bmatrix} 1 & \frac{1}{2} & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = LU. \tag{5.31}$$

We also note that

$$\det(C) = 2 \cdot \left(-\frac{1}{2}\right) \cdot 1 = -1.$$

The method of constructing the LU factorization described above is guaranteed to work in the case of matrices with all nonsingular principal submatrices and in some other cases, such as for C in equation (5.30). The fact that C is nonsingular is not sufficient to ensure that the process works or even that the factorization exists. (As we indicated above, the sufficient conditions are rather complicated, but not very important in practice.)

If the matrix is not of full rank, as the Gaussian matrices are being formed, at some point the diagonal element $a_{ii}^{(k)}$ will be zero, so the matrix G_{kk} cannot be formed. For such cases, we merely form a row of zeros in the lower triangular matrix, and proceed to the next diagonal element. Even if the matrix is of full rank, but not all principal submatrices are of full rank, we would encounter this same kind of problem. In applications, we may address these two problems similarly, using a technique of pivoting.

5.7.2 Pivoting

As we have seen, the sufficient condition of nonsingularity of all principal submatrices is not a necessary requirement for the existence of an LU factorization. We have also seen that, at least in some cases, if it exists, the factorization, can be performed using Gaussian steps. There are matrices such as B in equation (5.21), however, for which no LU factorization exists.

Does this matter? Obviously, it depends on the application; that is, it depends on the purpose of using an LU factorization.

One of the most common applications of an LU factorization is to solve a system of linear equations $Ax = b$. In such applications, interchange of rows or columns does not change the problem, so long as the interchanges are made appropriately over the entire system. Such an interchange is called *pivoting*.

Pivoting is often done not just to yield a matrix with an LU decomposition, it is routinely done in computations to improve the numerical accuracy. Pivoting is generally effected by premultiplication by an elementary permutation matrix E_{pq} (see equation (3.66) on page 85 for notation and definitions). Hence, instead of factoring A, we factor an equivalent matrix $E_{(\pi)}A$:

$$E_{(\pi)}A = LU,$$

where L and U are lower triangular or trapezoidal and upper triangular or trapezoidal respectively, and satisfying other restrictions we wish to impose on the LU factorization. In an LDU decomposition, we often choose a permutation matrix so that the diagonal elements of D are nonincreasing. Depending on the shape and other considerations of numerical computations, we may also permute the columns of the matrix, by postmultiplying by a permutation matrix. In its most general form, we may express the LDU decomposition of the matrix A as

$$E_{(\pi_1)}AE_{(\pi_2)} = LDU. \tag{5.32}$$

We will discuss pivoting in more detail on page 277 in Chap. 6.

As we mentioned in Chap. 3, in actual computations, we do not form the elementary transformation matrices or the Gaussian matrices explicitly, but their formulation in the text allows us to discuss the operations in a systematic way and better understand the properties of the operations.

This is an instance of a principle that we will encounter repeatedly: *the form of a mathematical expression and the way the expression should be evaluated in actual practice may be quite different.*

5.7.3 Use of Inner Products

The use of the elementary matrices described above is effectively a series of outer products (columns of the elementary matrices with rows of the matrices being operated on).

The LU factorization can also be performed by using inner products. From equation (5.29), we see

$$a_{ij} = \sum_{k=1}^{i-1} l_{ik}u_{kj} + u_{ij},$$

so

$$l_{ij} = \frac{a_{ij} - \sum_{k=1}^{j-1} l_{ik}u_{kj}}{u_{jj}} \quad \text{for } i = j+1, j+2, \ldots, n. \tag{5.33}$$

The use of computations implied by equation (5.33) is called the Doolittle method or the Crout method. (There is a slight difference between the Doolittle method and the Crout method: the Crout method yields a decomposition in which the 1s are on the diagonal of the U matrix rather than the L matrix.) Whichever method is used to form the LU decomposition, $n^3/3$ multiplications and additions are required.

5.7.4 Properties: Uniqueness

There are clearly many ways indeterminacies can occur in L, D, or U in an LU or LDU factorization in general. (Recall the simple replacement of L and U by $-L$ and $-U$.) In some cases, indeterminacies can be eliminated or reduced by putting restrictions on the factors, but any uniqueness of an LU factorization is rather limited.

If a nonsingular matrix has an LU factorization, the factorization itself in general is not unique, but given either L or U, the other factor is unique, as we can see by use of inverses. (Recall the form of the inverse of a triangular matrix, page 120.)

For the LDU factorization of a general square matrix, if L and U are restricted to be unit triangular matrices, then D is unique. To see this, let A be an $n \times n$ matrix for which an LDU factorization, and let $A = LDU$, with L a lower unit triangular matrix and U an upper unit triangular matrix and D a diagonal matrix. (All matrices are $n \times n$.) Now, suppose $A = \widetilde{L}\widetilde{D}\widetilde{U}$, where $\widetilde{L}$, $\widetilde{D}$, and $\widetilde{U}$ have the same patterns as L, D, and U. All of these unit triangular matrices have inverses of the same type (see page 120). Now, since $LDU = \widetilde{L}\widetilde{D}\widetilde{U}$, premultiplying by $\widetilde{L}^{-1}$ and postmultiplying by U^{-1}, we have

$$\widetilde{L}^{-1}LD = \widetilde{D}\widetilde{U}U^{-1};$$

that is, the diagonal elements of $\widetilde{L}^{-1}LD$ and $\widetilde{D}\widetilde{U}U^{-1}$ are the same. By the properties of unit triangular matrices given on page 78, we see that those diagonal elements are the diagonal elements of D and $\widetilde{D}$. Since, therefore, $D = \widetilde{D}$, D in the LDU factorization of a general square matrix is unique.

5.7.5 Properties of the LDU Factorization of a Square Matrix

The uniqueness of D in the LDU factorization of a general square matrix is an important fact. A related useful fact about the LDU factorization of a general square matrix A is

$$\det(A) = \prod d_{ii}, \tag{5.34}$$

which we see from equations (3.81) and (3.37) on pages 88 and 70 respectively.

There are other useful properties of the LDU factorization of square matrices with special properties, such as positive definiteness, but we will not pursue them here.

5.8 QR Factorization

A very useful factorization of a matrix is the product of an orthogonal matrix and an upper triangular matrix with nonnegative diagonal elements. Depending on the shape of A, the shapes of the factors may vary, and even the definition of the factors themselves may be stated differently.

Let A be an $n \times m$ matrix and suppose

$$A = QR, \tag{5.35}$$

where Q is an orthogonal matrix and R is an upper triangular or trapezoidal matrix with nonnegative diagonal elements. This is called the QR factorization of A. In most applications, $n \geq m$, but if this is not the case, we still have a factorization into similar matrices.

The QR factorization is useful for many tasks in linear algebra. It can be used to determine the rank of a matrix (see page 252 below), to extract eigenvalues and eigenvectors (see page 318), to form the singular value decomposition (see page 322), and to show various theoretical properties of matrices (see, for example, Exercise 5.5 on page 262). The QR factorization is particularly useful in computations for overdetermined systems, as we will see in Sect. 6.6 on page 289, and in other computations involving nonsquare matrices.

If A is square, both factors are square, but when A is not square, there are some variations in the form of the factorization. I will consider only the case in which the number of rows in A is at least as great as the number of columns. The other case is logically similar.

If $n > m$, there are two different forms of the QR factorization. In one form Q is an $n \times n$ matrix and R is an $n \times m$ upper trapezoidal matrix with

with zeroes in the lower rows. In the other form, Q is an $n \times m$ matrix with orthonormal columns and R is an $m \times m$ upper triangular matrix. This latter form is sometimes called a "skinny" QR factorization. When $n > m$, the skinny QR factorization is more commonly used than one with a square Q.

The two factorizations are essentially the same. If R_1 is the matrix in the skinny factorization and R is the matrix in the full form, they are related as

$$R = \begin{bmatrix} R_1 \\ 0 \end{bmatrix}. \tag{5.36}$$

Likewise the square Q can be partitioned as $[Q_1 \,|\, Q_2]$, and the skinny factorization written as

$$A = Q_1 R_1. \tag{5.37}$$

In the full form $Q^{\mathrm{T}}Q = I_n$ and in the skinny form, $Q_1^{\mathrm{T}}Q_1 = I_m$.

The existence of the QR factorization can be shown by construction using, for example, Householder reflections, as in Sect. 5.8.8 below.

5.8.1 Related Matrix Factorizations

For the $n \times m$ matrix, similar to the factorization in equation (5.35), we may have

$$A = RQ,$$

where Q is an orthogonal matrix and R is an upper triangular or trapezoidal matrix with nonnegative diagonal elements. This is called the RQ factorization of A.

Two other related factorizations, with obvious names, are

$$A = QL,$$

and

$$A = LQ,$$

where Q is an orthogonal matrix and L is an lower triangular or trapezoidal matrix with nonnegative diagonal elements.

5.8.2 Matrices of Full Column Rank

If the matrix A is of full column rank (meaning that there are at least as many rows as columns and the columns are linearly independent), as in many applications in statistics, the R matrix in the QR factorization is full rank. Furthermore, in the skinny QR factorization $A = Q_1 R_1$, R_1 is nonsingular and the factorization is unique. (Recall that the diagonal elements are required to be nonnegative.)

We see that the factorization is unique by forming $A = Q_1 R_1$ and then letting $\widetilde{Q}_1$ and $\widetilde{R}_1$ be the skinny QR factorization

$$A = \widetilde{Q}_1 \widetilde{R}_1,$$

and showing that $\widetilde{Q}_1 = Q_1$ and $\widetilde{R}_1 = R_1$. Since $Q_1 R_1 = \widetilde{Q}_1 \widetilde{R}_1$, we have $Q_1 = \widetilde{Q}_1 \widetilde{R}_1 R_1^{-1}$ and so $\widetilde{R}_1 R_1^{-1} = \widetilde{Q}_1^{\mathrm{T}} Q_1$. As we saw on page 120, since R_1 is upper triangular, R_1^{-1} is upper triangular; and as we saw on page 78, since $\widetilde{R}_1$ is upper triangular, $\widetilde{R}_1 R_1^{-1}$ is upper triangular. Let T be this upper triangular matrix,

$$T = \widetilde{R}_1 R_1^{-1}.$$

Now consider $T^{\mathrm{T}} T$. (This is a Cholesky factorization; see Sect. 5.9.2.)

Since $\widetilde{Q}_1^{\mathrm{T}} \widetilde{Q}_1 = I_m$, we have $T^{\mathrm{T}} T = T^{\mathrm{T}} \widetilde{Q}_1^{\mathrm{T}} \widetilde{Q}_1 T$. Now, because $Q_1 = \widetilde{Q}_1 T$, we have

$$T^{\mathrm{T}} T = Q_1^{\mathrm{T}} Q_1 = I_m.$$

The only upper triangular matrix T such that $T^{\mathrm{T}} T = I$ is the identity I itself. (This is from the definition of matrix multiplication. First, we see that $t_{11} = 1$, and since all off-diagonal elements in the first row of I are 0, all off-diagonal elements in the first row of T^{T} must be 0. Continuing in this way, we see that $T = I$.) Hence,

$$\widetilde{R}_1 R_1^{-1} = I,$$

and so

$$\widetilde{R}_1 = R_1.$$

Now,

$$\widetilde{Q}_1 = \widetilde{Q}_1 T = Q_1;$$

hence, the factorization is unique.

5.8.3 Relation to the Moore-Penrose Inverse for Matrices of Full Column Rank

If the matrix A is of full column rank, the Moore-Penrose inverse of A is immediately available from the QR factorization:

$$A^+ = \begin{bmatrix} R_1^{-1} & 0 \end{bmatrix} Q^{\mathrm{T}}. \tag{5.38}$$

(The four properties of a Moore-Penrose inverse listed on page 128 are easily verified, and you are asked to do so in Exercise 5.8.)

5.8.4 Nonfull Rank Matrices

If A is square but not of full rank, R has the form

$$\begin{bmatrix} \mathtt{X} & \mathtt{X} & \mathtt{X} \\ \mathtt{0} & \mathtt{X} & \mathtt{X} \\ \mathtt{0} & \mathtt{0} & \mathtt{0} \end{bmatrix}. \tag{5.39}$$

In the common case in which A has more rows than columns, if A is not of full (column) rank, R_1 in equation (5.36) will have the form shown in matrix (5.39).

If A is not of full rank, we apply permutations to the columns of A by multiplying on the right by a permutation matrix. The permutations can be taken out by a second multiplication on the right. If A is of rank r ($\leq m$), the resulting decomposition consists of three matrices: an orthogonal Q, a T with an $r \times r$ upper triangular submatrix, and a permutation matrix $E_{(\pi)}^{\mathrm{T}}$,

$$A = QTE_{(\pi)}^{\mathrm{T}}. \tag{5.40}$$

The matrix T has the form

$$T = \begin{bmatrix} T_1 & T_2 \\ 0 & 0 \end{bmatrix}, \tag{5.41}$$

where T_1 is upper triangular and is $r \times r$. The decomposition in equation (5.40) is not unique because of the permutation matrix. The choice of the permutation matrix is the same as the pivoting that we discussed in connection with Gaussian elimination. A generalized inverse of A is immediately available from equation (5.40):

$$A^{-} = P \begin{bmatrix} T_1^{-1} & 0 \\ 0 & 0 \end{bmatrix} Q^{\mathrm{T}}. \tag{5.42}$$

Additional orthogonal transformations can be applied from the right-hand side of the $n \times m$ matrix A in the form of equation (5.40) to yield

$$A = QRU^{\mathrm{T}}, \tag{5.43}$$

where R has the form

$$R = \begin{bmatrix} R_1 & 0 \\ 0 & 0 \end{bmatrix}, \tag{5.44}$$

where R_1 is $r \times r$ upper triangular, Q is $n \times n$ and as in equation (5.40), and U^{T} is $n \times m$ and orthogonal. (The permutation matrix in equation (5.40) is also orthogonal, of course.)

5.8.5 Relation to the Moore-Penrose Inverse

The decomposition (5.43) is unique, and it provides the unique Moore-Penrose generalized inverse of A:

$$A^{+} = U \begin{bmatrix} R_1^{-1} & 0 \\ 0 & 0 \end{bmatrix} Q^{\mathrm{T}}. \tag{5.45}$$

5.8.6 Determining the Rank of a Matrix

It is often of interest to know the rank of a matrix. Given a decomposition of the form of equation (5.40), the rank is obvious, and in practice, this QR decomposition with pivoting is a good way to determine the rank of a matrix. The QR decomposition is said to be "rank-revealing".

For many matrices, the computations are quite sensitive to rounding. Pivoting is often required, and even so, the pivoting must be done with some care (see Hong and Pan 1992; Section 2.7.3 of Björck 1996; and Bischof and Quintana-Ortí 1998a,b). (As we pointed out on page 121, the problem itself is ill-posed in Hadamard's sense because the rank is not a continuous function of any of the quantities that determine it. For a given matrix, the problem can also be ill-conditioned in the computational sense. Ill-conditioning is a major concern, and we will discuss it often in latter chapters of this book. We introduce some of the concepts of ill-conditioning formally in Sect. 6.1.)

5.8.7 Formation of the QR Factorization

There are three good methods for obtaining the QR factorization: Householder transformations or reflections; Givens transformations or rotations; and the (modified) Gram-Schmidt procedure. Different situations may make one of these procedures better than the two others. The Householder transformations described in the next section are probably the most commonly used. If the data are available only one row at a time, the Givens transformations discussed in Sect. 5.8.9 are very convenient. Whichever method is used to compute the QR decomposition, at least $2n^3/3$ multiplications and additions are required. The operation count is therefore about twice as great as that for an LU decomposition.

5.8.8 Householder Reflections to Form the QR Factorization

To use reflectors to compute a QR factorization, we form in sequence the reflector for the i^{th} column that will produce 0s below the (i, i) element.

For a convenient example, consider the matrix

$$A = \begin{bmatrix} 3 & -\frac{98}{28} & \mathtt{X} & \mathtt{X} & \mathtt{X} \\ 1 & \frac{122}{28} & \mathtt{X} & \mathtt{X} & \mathtt{X} \\ 2 & -\frac{8}{28} & \mathtt{X} & \mathtt{X} & \mathtt{X} \\ 1 & \frac{66}{28} & \mathtt{X} & \mathtt{X} & \mathtt{X} \\ 1 & \frac{10}{28} & \mathtt{X} & \mathtt{X} & \mathtt{X} \end{bmatrix}.$$

The first transformation would be determined so as to transform $(3, 1, 2, 1, 1)$ to $(\mathtt{X}, 0, 0, 0, 0)$. We use equations (5.8) through (5.10) to do this. Call this first Householder matrix P_1. We have

$$P_1 A = \begin{bmatrix} -4 & 1 & \mathtt{X} & \mathtt{X} & \mathtt{X} \\ 0 & 5 & \mathtt{X} & \mathtt{X} & \mathtt{X} \\ 0 & 1 & \mathtt{X} & \mathtt{X} & \mathtt{X} \\ 0 & 3 & \mathtt{X} & \mathtt{X} & \mathtt{X} \\ 0 & 1 & \mathtt{X} & \mathtt{X} & \mathtt{X} \end{bmatrix}.$$

We now choose a reflector to transform $(5, 1, 3, 1)$ to $(-6, 0, 0, 0)$. We do not want to disturb the first column in $P_1 A$ shown above, so we form P_2 as

$$P_2 = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & & & \\ \vdots & & H_2 & \\ 0 & & & \end{bmatrix}.$$

Forming the vector $(11, 1, 3, 1)/\sqrt{132}$ and proceeding as before, we get the reflector

$$\begin{aligned} H_2 &= I - \frac{1}{66}(11, 1, 3, 1)(11, 1, 3, 1)^{\mathrm{T}} \\ &= \frac{1}{66}\begin{bmatrix} -55 & -11 & -33 & -11 \\ -11 & 65 & -3 & -1 \\ -33 & -3 & 57 & -3 \\ -11 & -1 & -3 & 65 \end{bmatrix}. \end{aligned}$$

Now we have

$$P_2 P_1 A = \begin{bmatrix} -4 & 1 & \mathtt{X} & \mathtt{X} & \mathtt{X} \\ 0 & -6 & \mathtt{X} & \mathtt{X} & \mathtt{X} \\ 0 & 0 & \mathtt{X} & \mathtt{X} & \mathtt{X} \\ 0 & 0 & \mathtt{X} & \mathtt{X} & \mathtt{X} \\ 0 & 0 & \mathtt{X} & \mathtt{X} & \mathtt{X} \end{bmatrix}.$$

Continuing in this way for three more steps, we would have the QR decomposition of A with $Q^{\mathrm{T}} = P_5 P_4 P_3 P_2 P_1$.

The number of computations for the QR factorization of an $n \times n$ matrix using Householder reflectors is $2n^3/3$ multiplications and $2n^3/3$ additions.

5.8.9 Givens Rotations to Form the QR Factorization

Just as we built the QR factorization by applying a succession of Householder reflections, we can also apply a succession of Givens rotations to achieve the factorization. If the Givens rotations are applied directly, the number of computations is about twice as many as for the Householder reflections, but if

fast Givens rotations are used and accumulated cleverly, the number of computations for Givens rotations is not much greater than that for Householder reflections. As mentioned on page 241, it is necessary to monitor the differences in the magnitudes of the elements in the C matrix and often necessary to rescale the elements. This additional computational burden is excessive unless done carefully (see Bindel et al. 2002, for a description of an efficient method).

5.8.10 Gram-Schmidt Transformations to Form the QR Factorization

Gram-Schmidt transformations yield a set of orthonormal vectors that span the same space as a given set of linearly independent vectors, $\{x_1, x_2, \ldots, x_m\}$. Application of these transformations is called Gram-Schmidt orthogonalization. If the given linearly independent vectors are the columns of a matrix A, the Gram-Schmidt transformations ultimately yield the QR factorization of A. The basic Gram-Schmidt transformation is shown in equation (2.56) on page 38.

The Gram-Schmidt algorithm for forming the QR factorization is just a simple extension of equation (2.56); see Exercise 5.10 on page 263.

5.9 Factorizations of Nonnegative Definite Matrices

There are factorizations that may not exist except for nonnegative definite matrices, or may exist only for such matrices. The LU decomposition, for example, exists and is unique for a nonnegative definite matrix; but may not exist for general matrices (without permutations). In this section we discuss two important factorizations for nonnegative definite matrices, the square root and the Cholesky factorization.

5.9.1 Square Roots

On page 160, we defined the square root of a nonnegative definite matrix in the natural way and introduced the notation $A^{\frac{1}{2}}$ as the square root of the nonnegative definite $n \times n$ matrix A:

$$A = \left(A^{\frac{1}{2}}\right)^2. \tag{5.46}$$

Just as the computation of a square root of a general real number requires iterative methods, the computation of the square root of a matrix requires iterative methods. In this case, the iterative methods are required for the evaluation of the eigenvalues (as we will describe in Chap. 7). Once the eigenvalues are available, the computations are simple, as we describe below.

Because A is symmetric, it has a diagonal factorization, and because it is nonnegative definite, the elements of the diagonal matrix are nonnegative. In terms of the orthogonal diagonalization of A, as on page 160, we write $A^{\frac{1}{2}} = VC^{\frac{1}{2}}V^{\mathrm{T}}$.

We now show that this square root of a nonnegative definite matrix is unique among nonnegative definite matrices. Let A be a (symmetric) nonnegative definite matrix and $A = VCV^{\mathrm{T}}$, and let B be a symmetric nonnegative definite matrix such that $B^2 = A$. We want to show that $B = VC^{\frac{1}{2}}V^{\mathrm{T}}$ or that $B - VC^{\frac{1}{2}}V^{\mathrm{T}} = 0$. Form

$$\begin{aligned}\left(B - VC^{\frac{1}{2}}V^{\mathrm{T}}\right)\left(B - VC^{\frac{1}{2}}V^{\mathrm{T}}\right) &= B^2 - VC^{\frac{1}{2}}V^{\mathrm{T}}B - BVC^{\frac{1}{2}}V^{\mathrm{T}} + \left(VC^{\frac{1}{2}}V^{\mathrm{T}}\right)^2 \\ &= 2A - VC^{\frac{1}{2}}V^{\mathrm{T}}B - \left(VC^{\frac{1}{2}}V^{\mathrm{T}}B\right)^{\mathrm{T}}. \qquad (5.47)\end{aligned}$$

Now, we want to show that $VC^{\frac{1}{2}}V^{\mathrm{T}}B = A$. The argument below follows (Harville 1997). Because B is nonnegative definite, we can write $B = UDU^{\mathrm{T}}$ for an orthogonal $n \times n$ matrix U and a diagonal matrix D with nonnegative elements, $d_1, \ldots d_n$. We first want to show that $V^{\mathrm{T}}UD = C^{\frac{1}{2}}V^{\mathrm{T}}U$. We have

$$\begin{aligned}V^{\mathrm{T}}UD^2 &= V^{\mathrm{T}}UDU^{\mathrm{T}}UDU^{\mathrm{T}}U \\ &= V^{\mathrm{T}}B^2U \\ &= V^{\mathrm{T}}AU \\ &= V^{\mathrm{T}}(VC^{\frac{1}{2}}V^{\mathrm{T}})^2U \\ &= V^{\mathrm{T}}VC^{\frac{1}{2}}V^{\mathrm{T}}VC^{\frac{1}{2}}V^{\mathrm{T}}U \\ &= CV^{\mathrm{T}}U.\end{aligned}$$

Now consider the individual elements in these matrices. Let z_{ij} be the $(ij)^{\text{th}}$ element of $V^{\mathrm{T}}U$, and since D^2 and C are diagonal matrices, the $(ij)^{\text{th}}$ element of $V^{\mathrm{T}}UD^2$ is $d_j^2 z_{ij}$ and the corresponding element of $CV^{\mathrm{T}}U$ is $c_i z_{ij}$, and these two elements are equal, so $d_j z_{ij} = \sqrt{c_i} z_{ij}$. These, however, are the $(ij)^{\text{th}}$ elements of $V^{\mathrm{T}}UD$ and $C^{\frac{1}{2}}V^{\mathrm{T}}U$, respectively; hence $V^{\mathrm{T}}UD = C^{\frac{1}{2}}V^{\mathrm{T}}U$. We therefore have

$$VC^{\frac{1}{2}}V^{\mathrm{T}}B = VC^{\frac{1}{2}}V^{\mathrm{T}}UDU^{\mathrm{T}} = VC^{\frac{1}{2}}C^{\frac{1}{2}}V^{\mathrm{T}}UU^{\mathrm{T}} = VCV^{\mathrm{T}} = A.$$

We conclude that $VC^{\frac{1}{2}}V^{\mathrm{T}}$ is the unique square root of A.

If A is positive definite, it has an inverse, and the unique square root of the inverse is denoted as $A^{-\frac{1}{2}}$.

5.9.2 Cholesky Factorization

If the matrix A is symmetric and nonnegative definite, another important factorization is the *Cholesky decomposition*. In this factorization,

$$A = T^{\mathrm{T}}T, \tag{5.48}$$

where T is an upper triangular matrix with nonnegative diagonal elements. We occasionally denote the Cholesky factor of A (that is, T in the expression above) as A_{C}. (Notice on page 48 and later on page 366 that we use a lowercase c subscript to represent a centered vector or matrix.)

The factor T in the Cholesky decomposition is sometimes called the square root, but we have defined a different matrix as the square root, $A^{\frac{1}{2}}$ (page 160 and Sect. 5.9.1). The Cholesky factor is more useful in practice, but the square root has more applications in the development of the theory.

We first consider the Cholesky decomposition of a positive definite matrix A. In that case, a factor of the form of T in equation (5.48) is unique up to the sign, just as a square root is. To make the Cholesky factor unique, we require that the diagonal elements be positive. The elements along the diagonal of T will be square roots. Notice, for example, that t_{11} is $\sqrt{a_{11}}$.

Algorithm 5.1 is a method for constructing the Cholesky factorization of a positive definite matrix A. The algorithm serves as the basis for a constructive proof of the existence and uniqueness of the Cholesky factorization (see Exercise 5.6 on page 262). The uniqueness is seen by factoring the principal square submatrices.

Algorithm 5.1 Cholesky factorization of a positive definite matrix

1. Let $t_{11} = \sqrt{a_{11}}$.
2. For $j = 2, \ldots, n$, let $t_{1j} = a_{1j}/t_{11}$.
3. For $i = 2, \ldots, n$,
   {
   let $t_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} t_{ki}^2}$, and
   for $j = i + 1, \ldots, n$,
   {
   let $t_{ij} = (a_{ij} - \sum_{k=1}^{i-1} t_{ki}t_{kj})/t_{ii}$.
   }
   }

∎

There are other algorithms for computing the Cholesky decomposition. The method given in Algorithm 5.1 is sometimes called the inner product formulation because the sums in step 3 are inner products. The algorithms for computing the Cholesky decomposition are numerically stable. Although the order of the number of computations is the same, there are only about half as many computations in the Cholesky factorization as in the LU factorization. Another advantage of the Cholesky factorization is that there are only $n(n+1)/2$ unique elements as opposed to $n^2 + n$ in the LU factorization.

The Cholesky decomposition can also be formed as $\widetilde{T}^{\mathrm{T}}D\widetilde{T}$, where D is a diagonal matrix that allows the diagonal elements of $\widetilde{T}$ to be computed without taking square roots. This modification is sometimes called a *Banachiewicz*

factorization or *root-free Cholesky*. The Banachiewicz factorization can be formed in essentially the same way as the Cholesky factorization shown in Algorithm 5.1: just put 1s along the diagonal of T and store the squared quantities in a vector d.

5.9.2.1 Cholesky Decomposition of Singular Nonnegative Definite Matrices

Any symmetric nonnegative definite matrix has a decomposition similar to the Cholesky decomposition for a positive definite matrix. If A is $n \times n$ with rank r, there exists a unique matrix T such that $A = T^{\mathrm{T}}T$, where T is an upper triangular matrix with r positive diagonal elements and $n - r$ rows containing all zeros. The algorithm is the same as Algorithm 5.1, except that in step 3 if $t_{ii} = 0$, the entire row is set to zero. The algorithm serves as a constructive proof of the existence and uniqueness.

5.9.2.2 Relations to Other Factorizations

For a symmetric matrix, the LDU factorization is $U^{\mathrm{T}}DU$; hence, we have for the Cholesky factor

$$T = D^{\frac{1}{2}}U,$$

where $D^{\frac{1}{2}}$ is the matrix whose elements are the square roots of the corresponding elements of D. (This is consistent with our notation above for Cholesky factors; $D^{\frac{1}{2}}$ is the Cholesky factor of D, and it is symmetric.)

The LU and Cholesky decompositions generally are applied to square matrices. However, many of the linear systems that occur in scientific applications are *overdetermined*; that is, there are more equations than there are variables, resulting in a nonsquare coefficient matrix.

For the $n \times m$ matrix A with $n \geq m$, we can write

$$\begin{aligned} A^{\mathrm{T}}A &= R^{\mathrm{T}}Q^{\mathrm{T}}QR \\ &= R^{\mathrm{T}}R, \end{aligned} \tag{5.49}$$

so we see that the matrix R in the QR factorization is (or at least can be) the same as the matrix T in the Cholesky factorization of $A^{\mathrm{T}}A$. There is some ambiguity in the Q and R matrices, but if the diagonal entries of R are required to be nonnegative, the ambiguity disappears and the matrices in the QR decomposition are unique.

An overdetermined system may be written as

$$Ax \approx b,$$

where A is $n \times m$ ($n \geq m$), or it may be written as

$$Ax = b + e,$$

where e is an n-vector of possibly arbitrary "errors". Because not all equations can be satisfied simultaneously, we must define a meaningful "solution". A useful solution is an x such that e has a small norm. The most common definition is an x such that e has the least Euclidean norm; that is, such that the sum of squares of the e_is is minimized.

It is easy to show that such an x satisfies the square system $A^{\mathrm{T}}Ax = A^{\mathrm{T}}b$, the "normal equations". This expression is important and allows us to analyze the overdetermined system (not just to solve for the x but to gain some better understanding of the system). It is easy to show that if A is of full rank (i.e., of rank m, all of its columns are linearly independent, or, redundantly, "full column rank"), then $A^{\mathrm{T}}A$ is positive definite. Therefore, we could apply either Gaussian elimination or the Cholesky decomposition to obtain the solution.

As we have emphasized many times before, however, *useful conceptual expressions are not necessarily useful as computational formulations.* That is sometimes true in this case also. In Sect. 6.1, we will discuss issues relating to the expected accuracy in the solutions of linear systems. There we will define a "condition number". Larger values of the condition number indicate that the expected accuracy is less. We will see that the condition number of $A^{\mathrm{T}}A$ is the square of the condition number of A. Given these facts, we conclude that it may be better to work directly on A rather than on $A^{\mathrm{T}}A$, which appears in the normal equations. We discuss solutions of overdetermined systems in Sect. 6.6, beginning on page 289, and in Sect. 6.7, beginning on page 296. Overdetermined systems are also a main focus of the statistical applications in Chap. 9.

5.9.3 Factorizations of a Gramian Matrix

The sums of squares and cross products matrix, the Gramian matrix $X^{\mathrm{T}}X$, formed from a given matrix X, arises often in linear algebra. We discuss properties of the sums of squares and cross products matrix beginning on page 359. Now we consider some additional properties relating to various factorizations.

First we observe that $X^{\mathrm{T}}X$ is symmetric and hence has an orthogonally similar canonical factorization,

$$X^{\mathrm{T}}X = VCV^{\mathrm{T}}.$$

We have already observed that $X^{\mathrm{T}}X$ is nonnegative definite, and so it has the LU factorization

$$X^{\mathrm{T}}X = LU,$$

with L lower triangular and U upper triangular, and it has the Cholesky factorization

$$X^{\mathrm{T}}X = T^{\mathrm{T}}T$$

with T upper triangular. With $L = T^{\mathrm{T}}$ and $U = T$, both factorizations are the same. In the LU factorization, the diagonal elements of either L or U

are often constrained to be 1, and hence the two factorizations are usually different.

It is instructive to relate the factors of the $m \times m$ matrix $X^{\mathrm{T}}X$ to the factors of the $n \times m$ matrix X. Consider the QR factorization

$$X = QR,$$

where R is upper triangular. Then $X^{\mathrm{T}}X = (QR)^{\mathrm{T}}QR = R^{\mathrm{T}}R$, so R is the Cholesky factor T because the factorizations are unique (again, subject to the restrictions that the diagonal elements be nonnegative).

Consider the SVD factorization

$$X = UDV^{\mathrm{T}}.$$

We have $X^{\mathrm{T}}X = (UDV^{\mathrm{T}})^{\mathrm{T}}UDV^{\mathrm{T}} = VD^2V^{\mathrm{T}}$, which is the orthogonally similar canonical factorization of $X^{\mathrm{T}}X$. The eigenvalues of $X^{\mathrm{T}}X$ are the squares of the singular values of X, and the condition number of $X^{\mathrm{T}}X$ (which we define in Sect. 6.1) is the square of the condition number of X.

5.10 Approximate Matrix Factorization

It is occasionally of interest to form a factorization that approximates a given matrix. For a given matrix A, we may have factors B and C, where

$$\widetilde{A} = BC,$$

and $\widetilde{A}$ is an approximation to A. (See Sect. 3.10, beginning on page 175, for discussions of approximation of matrices.)

The approximate factorization

$$A \approx BC$$

may be useful for various reasons. The computational burden of an exact factorization may be excessive. Alternatively, the matrices B and C may have desirable properties that no exact factors of A possess.

In this section we discuss two kinds of approximate factorizations, one motivated by the properties of the matrices, and the other merely an incomplete factorization, which may be motivated by computational expediency or by other considerations.

5.10.1 Nonnegative Matrix Factorization

If A in an $n \times m$ matrix all of whose elements are nonnegative, it may be of interest to approximate A as

$$A \approx WH,$$

where W is $n \times r$ and H $r \times m$, and both W and H have only nonnegative elements. Such matrices are called nonnegative matrices. If all elements of a matrix are positive, the matrix is called a positive matrix. Nonnegative and positive matrices arise often in applications and have a number of interesting properties. These kinds of matrices are the subject of Sect. 8.7, beginning on page 372.

Clearly, if $r \geq \min(n, m)$, the factorization $A = WH$ exists exactly, for if $r = \min(n, m)$ then $A = I_r W$, which is not unique since for $a > 0$, $A = (\frac{1}{a})I_r(aH)$. If, however, $r < \min(n, m)$, the factorization may not exist.

A *nonnegative matrix factorization* (NMF) of the nonnegative matrix A for a given r is the expression WH, where the $n \times r$ matrix W and the $r \times m$ matrix H are nonnegative, and the difference $A - WH$ is minimum according to some criterion (see page 175); that is, given r, the NMF factorization of the $n \times m$ nonnegative matrix A are the matrices W and H satisfying

$$\begin{aligned} \min_{W \in R^{n \times r}, H \in R^{r \times m}} & \rho(A - WH) \\ \text{s.t. } & W, H \geq 0, \end{aligned} \tag{5.50}$$

where ρ is a measure of the size of $A - WH$. Interest in this factorization arose primarily in the problem of analysis of text documents (see page 339).

Most methods for solving the optimization problem (5.50) follow the alternating variables approach: for fixed $W^{(0)}$ determine an optimal $H^{(1)}$; then given optimal $H^{(k)}$ determine an optimal $W^{(k+1)}$ and for optimal $W^{(k+1)}$ determine an optimal $H^{(k+1)}$.

The ease of solving the optimization problem (5.50), whether or not the alternating variables approach is used, depends on the nature of ρ. Generally, ρ is a norm, but Lee and Seung (2001) considered ρ to be the Kullback-Leibler divergence (see page 176), and described a computational method for solving the optimization problem. Often ρ is chosen to be a Frobenius p norm, because that matrix norm can be expressed as a L_p vector norm, as shown in equation (3.299) on page 169. Furthermore, if the ordinary Frobenius norm is chosen (that is, $p = 2$), then each subproblem is just a constrained linear least squares problem (as discussed on page 211). Kim and Park (2008) described an alternating variables approach for nonnegative matrix factorization based on the ordinary Frobenius norm.

Often in applications, the matrix A to be factored is sparse. Computational methods for the factorization that take advantage of the sparsity can lead to improvements in the computational efficiency of orders of magnitude.

5.10.2 Incomplete Factorizations

Often instead of an exact factorization, an approximate or "incomplete" factorization may be more useful because of its computational efficiency. This may be the case in the context of an iterative algorithm in which a matrix is being successively transformed, and, although a factorization is used in

each step, the factors from a previous iteration are adequate approximations. Another common situation is in working with sparse matrices. Many exact operations on a sparse matrix yield a dense matrix; however, we may want to preserve the sparsity, even at the expense of losing exact equalities. When a zero position in a sparse matrix becomes nonzero, this is called "fill-in", and we want to avoid that.

For example, instead of an LU factorization of a sparse matrix A, we may seek lower and upper triangular factors $\widetilde{L}$ and $\widetilde{U}$, such that

$$A \approx \widetilde{L}\widetilde{U}, \tag{5.51}$$

and if $a_{ij} = 0$, then $\tilde{l}_{ij} = \tilde{u}_{ij} = 0$. This approximate factorization is easily accomplished by modifying the Gaussian elimination step that leads to the outer product algorithm of equations (5.27) and (5.28).

More generally, we may choose a set of indices $S = \{(p, q)\}$ and modify the elimination step, for $m \geq i$, to be

$$a_{ij}^{(k+1)} \leftarrow \begin{cases} a_{ij}^{(k)} - a_{mj}^{(k)} a_{ij}^{(k)} / a_{jj}^{(k)} & \text{if } (i,j) \in S \\ a_{ij} & \text{otherwise.} \end{cases} \tag{5.52}$$

Note that a_{ij} does not change unless (i, j) is in S. This allows us to preserve 0s in L and U corresponding to given positions in A.

Exercises

5.1. Consider the transformation of the 3-vector x that first rotates the vector 30° about the x_1 axis, then rotates the vector 45° about the x_2 axis, and then translates the vector by adding the 3-vector y. Find the matrix A that effects these transformations by a single multiplication. Use the vector x^{h} of homogeneous coordinates that corresponds to the vector x. (Thus, A is 4×4.)

5.2. Homogeneous coordinates are often used in mapping three-dimensional graphics to two dimensions. The perspective plot function `persp` in R, for example, produces a 4×4 matrix for projecting three-dimensional points represented in homogeneous coordinates onto two-dimensional points in the displayed graphic. R uses homogeneous coordinates in the form of equation (5.6b) rather than equation (5.6a). If the matrix produced is T and if a^{h} is the representation of a point (x_a, y_a, z_a) in homogeneous coordinates, in the form of equation (5.6b), then $a^{\mathrm{h}}T$ yields transformed homogeneous coordinates that correspond to the projection onto the two-dimensional coordinate system of the graphical display. Consider the two graphs in Fig. 5.4. The graph on the left in the unit cube was produced by the simple R statements

```
x<-c(0,1)
```

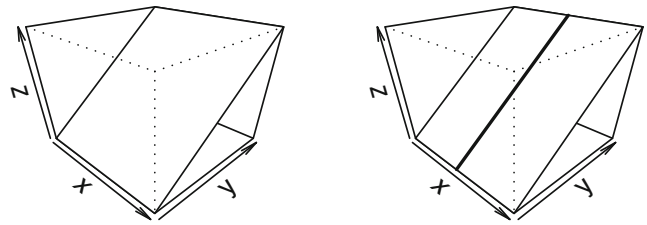


Figure 5.4. Illustration of the use of homogeneous coordinates to locate three-dimensional points on a two-dimensional graph

```
y<-c(0,1)
z<-matrix(c(0,0,1,1),nrow=2)
persp(x, y, z, theta = 45, phi = 30)
```

(The angles `theta` and `phi` are the azimuthal and latitudinal viewing angles, respectively, in degrees.) The graph on the right is the same with a heavy line going down the middle of the surface; that is, from the point $(0.5, 0, 0)$ to $(0.5, 1, 1)$. Obtain the transformation matrix necessary to identify the rotated points and produce the graph on the right.

5.3. Determine the rotation matrix that rotates 3-vectors through an angle of $30°$ in the plane $x_1 + x_2 + x_3 = 0$.

5.4. Let $A = LU$ be the LU decomposition of the $n \times n$ matrix A.

a) Suppose we multiply the j^{th} column of A by c_j, $j = 1, 2, \ldots n$, to form the matrix A_c. What is the LU decomposition of A_c? Try to express your answer in a compact form.

b) Suppose we multiply the i^{th} row of A by c_i, $i = 1, 2, \ldots n$, to form the matrix A_r. What is the LU decomposition of A_r? Try to express your answer in a compact form.

c) What application might these relationships have?

5.5. Use the QR decomposition to prove *Hadamard's inequality*:

$$|\det(A)| \leq \prod_{j=1}^{n} \|a_j\|_2,$$

where A is an $n \times n$ matrix, whose columns are the same as the vectors a_j. Equality holds if and only if either the a_j are mutually orthogonal or some a_j is zero.

5.6. Show that if A is positive definite, there exists a unique upper triangular matrix T with positive diagonal elements such that

$$A = T^{\mathrm{T}}T.$$

Hint: Show that $a_{ii} > 0$. Show that if A is partitioned into square submatrices A_{11} and A_{22},

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

that A_{11} and A_{22} are positive definite. Use Algorithm 5.1 (page 256) to show the existence of a T, and finally show that T is unique.

5.7. Let X_1, X_2, and X_3 be independent random variables identically distributed as standard normals.

a) Determine a matrix A such that the random vector

$$A \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix}$$

has a multivariate normal distribution with variance-covariance matrix

$$\begin{bmatrix} 4 & 2 & 8 \\ 2 & 10 & 7 \\ 8 & 7 & 21 \end{bmatrix}.$$

b) Is your solution unique? (The answer is no.) Determine a different solution.

5.8. Generalized inverses.

a) Prove equation (5.38) on page 250 (Moore-Penrose inverse of a full column rank matrix).

b) Prove equation (5.42) on page 251 (generalized inverse of a nonfull rank matrix).

c) Prove equation (5.45) on page 251, (Moore-Penrose inverse of a nonfull rank matrix).

5.9. Determine the Givens transformation matrix that will rotate the matrix

$$A = \begin{bmatrix} 3 & 5 & 6 \\ 6 & 1 & 2 \\ 8 & 6 & 7 \\ 2 & 3 & 1 \end{bmatrix}$$

so that the second column becomes $(5, \tilde{a}_{22}, 6, 0)$ (see also Exercise 12.5).

5.10. Gram-Schmidt transformations.

a) Use Gram-Schmidt transformations to determine an orthonormal basis for the space spanned by the vectors

$$\begin{aligned} v_1 &= (3, 6, 8, 2), \\ v_2 &= (5, 1, 6, 3), \\ v_3 &= (6, 2, 7, 1). \end{aligned}$$

b) Write out a formal algorithm for computing the QR factorization of the $n \times m$ full rank matrix A. Assume $n \geq m$.

c) Write a Fortran or C function to implement the algorithm you described.

6

Solution of Linear Systems

One of the most common problems in numerical computing is to solve the linear system

$$Ax = b;$$

that is, for given A and b, to find x such that the equation holds. The system is said to be *consistent* if there exists such an x, and in that case a solution x may be written as A^-b, where A^- is some inverse of A. If A is square and of full rank, we can write the solution as $A^{-1}b$.

It is important to distinguish the expression $A^{-1}b$ or A^-b, which represents the solution, from the method of computing the solution. We would never compute A^{-1} just so we could multiply it by b to form the solution $A^{-1}b$.

Two topics we discuss in this chapter have wider relevance in computations for scientific applications. The first is how to assess the expected numerical difficulties in solving a specific problem. A specific problem is comprised of a task and data, or input to the task. Some tasks are naturally harder than others; for example, it is generally more difficult to solve a constrained optimization problem than it is to solve a system of linear equations. Some constrained optimization problems, however, may be very "small" and even have a simple closed-form solution; while a system of linear equations may be very large and be subject to severe rounding errors (for reasons we will discuss). For any specific task, there may be ways of assessing the "difficulty" of a specific input. One measure of difficulty is a "condition number", and we discuss a condition number for the task of solving a linear system in Sect. 6.1.

Another topic that is illustrated by the techniques discussed in this chapter is the difference between direct methods and iterative methods. The methods for factoring a matrix discussed in Chap. 5 are direct methods; that is, there is a fixed sequence of operations (possibly with some pivoting along the way) that yields the solution. Iterative methods do not have a fixed, predetermined number of steps; rather, they must have a stopping rule that depends on

J.E. Gentle, *Matrix Algebra*, Springer Texts in Statistics,
DOI 10.1007/978-3-319-64867-5_6

the steps themselves. Most optimization methods discussed in Sect. 4.4 are examples of iterative methods. In Chap. 7, we will see that all methods for obtaining eigenvalues (for general square matrices larger than 4×4) are necessarily iterative.

An iterative method is characterized by a sequence of partial or approximate solutions, which I index by a superscript enclosed in parentheses: $s^{(0)}, s^{(1)}, s^{(2)}, \ldots$. A starting point is necessary. I usually index it as "$^{(0)}$". The iterative algorithm is essentially a rule for computing $s^{(k)}$ given $s^{(k-1)}$.

A stopping criterion is also necessary. The partial solution at any point in the sequence often consists of multiple parts; for example, in an optimization problem, there are two parts, the value of the decision variable and the value of the objective function. The stopping rule may be based on the magnitude of the change in one or more parts of the solution, as shown in the two criteria (4.24) and (4.25) on page 201. Of course, a stopping criterion based simply on the number of iterations is also possible, and, in fact, is usually a good idea in addition to any other stopping criterion.

We discuss direct methods for solving systems of linear equations in Sect. 6.2. A direct method uses a fixed number of computations that would in exact arithmetic lead to the solution. We may also refer to the steps in a direct method as "iterations"; but there is a fixed limit on the number of them that depends only on the size of the input.

We discuss iterative methods for solving systems of linear equations in Sect. 6.3. Iterative methods often work well for very large and/or sparse matrices.

Another general principle in numerical computations is that once an approximate solution has been obtained (in numerical computing, almost all solutions are approximate), it is often useful to see if the solution can be improved. This principle is well-illustrated by "iterative refinement" of a solution to a linear system, which we discuss in Sect. 6.4. Iterative refinement is an iterative method, but the method yielding the starting approximation may be a direct method.

6.1 Condition of Matrices

Data are said to be "ill-conditioned" for a particular problem or computation if the data are likely to cause difficulties in the computations, such as severe loss of precision. More generally, the term "ill-conditioned" is applied to a problem in which small changes to the input result in large changes in the output. In the case of a linear system

$$Ax = b,$$

the problem of solving the system is ill-conditioned if small changes to some elements of A or b will cause large changes in the solution x. The concept

of ill-conditionedness is a heuristic generalization of Hadamard's property of ill-posedness (see page 121).

Consider, for example, the system of equations

$$\begin{aligned} 1.000x_1 + 0.500x_2 &= 1.500, \\ 0.667x_1 + 0.333x_2 &= 1.000. \end{aligned} \tag{6.1}$$

The solution is easily seen to be $x_1 = 1.000$ and $x_2 = 1.000$.

Now consider a small change in the right-hand side:

$$\begin{aligned} 1.000x_1 + 0.500x_2 &= 1.500, \\ 0.667x_1 + 0.333x_2 &= 0.999. \end{aligned} \tag{6.2}$$

This system has solution $x_1 = 0.000$ and $x_2 = 3.000$. That is a relatively large change.

Alternatively, consider a small change in one of the elements of the coefficient matrix:

$$\begin{aligned} 1.000x_1 + 0.500x_2 &= 1.500, \\ 0.667x_1 + 0.334x_2 &= 1.000. \end{aligned} \tag{6.3}$$

The solution now is $x_1 = 2.000$ and $x_2 = -1.000$; again, a relatively large change.

In both cases, small changes of the order of 10^{-3} in the input (the elements of the coefficient matrix or the right-hand side) result in relatively large changes (of the order of 1) in the output (the solution). Solving the system (either one of them) is an ill-conditioned problem when our relevant scales are of order 3.

The nature of the data that cause ill-conditioning depends on the type of problem. The case above can be considered as a geometric problem of determining the point of intersection of two lines. The problem is that the lines represented by the equations are almost parallel, as seen in Fig. 6.1, and so their point of intersection is very sensitive to slight changes in the coefficients defining the lines.

The problem can also be described in terms of the angle between the lines. When the angle is small, but not necessarily 0, we refer to the condition as "collinearity". (This term is somewhat misleading because, strictly speaking, it should indicate that the angle is exactly 0.) In this example, the cosine of the angle between the lines, from equation (2.54), is $1 - 2 \times 10^{-7}$. In general, collinearity (or "multicollinearity") exists whenever the angle between any line (that is, vector) and the subspace spanned by any other set of vectors is small.

6.1.1 Condition Number

For a specific problem such as solving a system of equations, we may quantify the condition of the matrix by a *condition number*. To develop this quantification for the problem of solving linear equations, consider a linear system

$Ax = b$, with A nonsingular and $b \neq 0$, as above. Now perturb the system slightly by adding a small amount, δb, to b, and let $\tilde{b} = b + \delta b$. The system

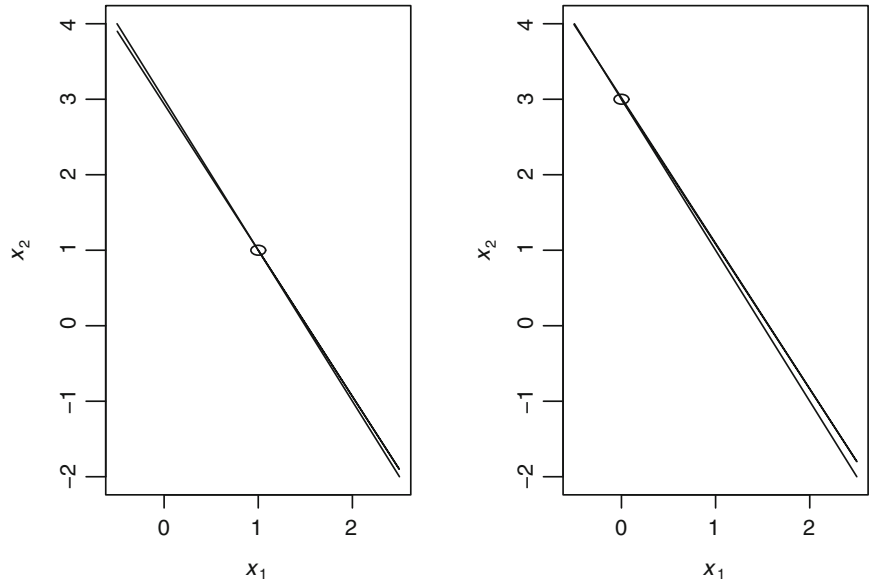


Figure 6.1. Almost parallel lines: ill-conditioned coefficient matrices, equations (6.1) and (6.2)

$$A\tilde{x} = \tilde{b}$$

has a solution $\tilde{x} = \delta x + x = A^{-1}\tilde{b}$. (Notice that δb and δx do not necessarily represent scalar multiples of the respective vectors.) If the system is well-conditioned, for any reasonable norm, if $\|\delta b\|/\|b\|$ is small, then $\|\delta x\|/\|x\|$ is likewise small.

From $\delta x = A^{-1}\delta b$ and the inequality (3.280) (page 165), for an induced norm on A, we have

$$\|\delta x\| \leq \|A^{-1}\| \, \|\delta b\|. \tag{6.4}$$

Likewise, because $b = Ax$, we have

$$\frac{1}{\|x\|} \leq \|A\| \frac{1}{\|b\|}, \tag{6.5}$$

and equations (6.4) and (6.5) together imply

$$\frac{\|\delta x\|}{\|x\|} \leq \|A\| \, \|A^{-1}\| \frac{\|\delta b\|}{\|b\|}. \tag{6.6}$$

This provides a bound on the change in the solution $\|\delta x\|/\|x\|$ in terms of the perturbation $\|\delta b\|/\|b\|$.

The bound in equation (6.6) motivates us to define the *condition number with respect to inversion*, denoted by $\kappa(\cdot)$, as

$$\kappa(A) = \|A\| \, \|A^{-1}\| \tag{6.7}$$

for nonsingular A. In the context of linear algebra, the condition number with respect to inversion is so dominant in importance that we generally just refer to it as the "condition number". This condition number also provides a bound on changes in eigenvalues due to perturbations of the matrix, as we will see in inequality (7.1). (That particular bound is of more theoretical interest than of practical value.)

A condition number is a useful measure of the condition of A for the problem of solving a linear system of equations. There are other condition numbers useful in numerical analysis, however, such as the condition number for computing the sample variance (see equation (10.3.2.7) on page 504) or a condition number for a root of a function.

We can write equation (6.6) as

$$\frac{\|\delta x\|}{\|x\|} \leq \kappa(A) \frac{\|\delta b\|}{\|b\|}, \tag{6.8}$$

and, following a development similar to that above, write

$$\frac{\|\delta b\|}{\|b\|} \leq \kappa(A) \frac{\|\delta x\|}{\|x\|}. \tag{6.9}$$

These inequalities, as well as the other ones we write in this section, are sharp, as we can see by letting $A = I$.

Because the condition number is an upper bound on a quantity that we would not want to be large, a large condition number is "bad".

Notice that our definition of the condition number does not specify the norm; it only requires that the norm be an induced norm. (An equivalent definition does not rely on the norm being an induced norm.) We sometimes specify a condition number with regard to a particular norm, and just as we sometimes denote a specific norm by a special symbol, we may use a special symbol to denote a specific condition number. For example, $\kappa_p(A)$ may denote the condition number of A in terms of an L_p norm. Most of the properties of condition numbers (but not their actual values) are independent of the norm used.

The coefficient matrix in equations (6.1) and (6.2) is

$$A = \begin{bmatrix} 1.000 & 0.500 \\ 0.667 & 0.333 \end{bmatrix},$$

and its inverse is

$$A^{-1} = \begin{bmatrix} -666 & 1000 \\ 1344 & -2000 \end{bmatrix}.$$

It is easy to see that

$$\|A\|_1 = 1.667$$

and

$$\|A^{-1}\|_1 = 3000;$$

hence,

$$\kappa_1(A) = 5001.$$

Likewise,

$$\|A\|_\infty = 1.500$$

and

$$\|A^{-1}\|_\infty = 3344;$$

hence,

$$\kappa_\infty(A) = 5016.$$

Notice that the condition numbers are not exactly the same, but they are close. Notice also that the condition numbers are of the order of magnitude of the ratio of the output perturbation to the input perturbation in those equations.

Although we used this matrix in an example of ill-conditioning, these condition numbers, although large, are not so large as to cause undue concern for numerical computations. Indeed, solving the systems of equations (6.1), (6.2), and (6.3) would not cause problems for a computer program.

An interesting relationship for the L_2 condition number is

$$\kappa_2(A) = \frac{\max_{x \neq 0} \frac{\|Ax\|}{\|x\|}}{\min_{x \neq 0} \frac{\|Ax\|}{\|x\|}} \tag{6.10}$$

(see Exercise 6.1, page 305). The numerator and denominator in equation (6.10) look somewhat like the maximum and minimum eigenvalues, as we have suggested. Indeed, the L_2 condition number is just the ratio of the largest eigenvalue in absolute value to the smallest (see page 166). The L_2 condition number is also called the *spectral condition number*.

The eigenvalues of the coefficient matrix in equations (6.1) and (6.2) are 1.333375 and −0.0003750, and so

$$\kappa_2(A) = 3555.67,$$

which is the same order of magnitude as $\kappa_\infty(A)$ and $\kappa_1(A)$ computed above.

Some useful facts about condition numbers are:

- $\kappa(A) = \kappa(A^{-1})$,
- $\kappa(cA) = \kappa(A), \quad \text{for } c \neq 0$,
- $\kappa(A) \geq 1$,
- $\kappa_1(A) = \kappa_\infty(A^{\mathrm{T}})$,
- $\kappa_2(A^{\mathrm{T}}) = \kappa_2(A)$,
- $\kappa_2(A^{\mathrm{T}}A) = \kappa_2^2(A)$
  $\geq \kappa_2(A)$, and
- if A and B are orthogonally similar (equation (3.242)), then

$$\|A\|_2 = \|B\|_2$$

and

$$\kappa_2(A) = \kappa_2(B)$$

(see equation (3.286)).

Even though the condition number provides a very useful indication of the condition of the problem of solving a linear system of equations, it can be misleading at times. Consider, for example, the coefficient matrix

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

where $0 < \epsilon < 1$. The condition numbers are

$$\kappa_1(A) = \kappa_2(A) = \kappa_\infty(A) = \frac{1}{\epsilon},$$

and so if ϵ is small, the condition number is large. It is easy to see, however, that small changes to the elements of A or b in the system $Ax = b$ do not cause undue changes in the solution (our heuristic definition of ill-conditioning). In fact, the simple expedient of multiplying the second row of A by $1/\epsilon$ (that is, multiplying the second equation, $a_{21}x_1 + a_{22}x_2 = b_2$, by $1/\epsilon$) yields a linear system that is very well-conditioned.

This kind of apparent ill-conditioning is called *artificial ill-conditioning*. It is due to the different rows (or columns) of the matrix having a very different *scale*; the condition number can be changed just by scaling the rows or columns. This usually does not make a linear system any better or any worse conditioned.

In Sect. 6.1.3 we relate the condition number to bounds on the numerical accuracy of the solution of a linear system of equations.

The relationship between the size of the matrix and its condition number is interesting. In general, we would expect the condition number to increase as the size increases. This is the case, but the nature of the increase depends on the type of elements in the matrix. If the elements are randomly and independently distributed as normal or uniform with a mean of zero and variance of one, the increase in the condition number is approximately linear in the size of the matrix (see Exercise 10.23, page 521).

Our definition of condition number given above is for nonsingular matrices. We can formulate a useful alternate definition that extends to singular matrices and to nonsquare matrices: the *condition number* of a matrix is the ratio of the largest singular value to the smallest nonzero singular value, and of course this is the same as the definition for square nonsingular matrices. This is also called the *spectral condition number*.

The condition number, like the determinant, is not easy to compute (see page 535 in Sect. 11.4).

6.1.2 Improving the Condition Number

Sometimes the condition number reflects an essential characteristic of the problem being addressed. In other cases, it is only an artifact of the model formulation for the problem, as in the case of artificial ill-conditioning, in which a simple change of units removes the ill-conditioning.

In other cases, we change the problem slightly. For example, we can improve the condition number of a matrix by adding a diagonal matrix. Although this fact holds in general, we consider only one special case. Let A be positive definite, and let $d > 0$. Then, for the spectral condition number, we have

$$\kappa_2(A + dI) < \kappa_2(A). \tag{6.11}$$

To see this, let c_i be an eigenvalue of A (which is positive). We have

$$\frac{\max(c_i + d)}{\min(c_i + d)} < \frac{\max(c_i)}{\min(c_i)}$$

for $d > 0$.

6.1.2.1 Ridge Regression and the Condition Number

Ridge regression, which we will discuss briefly beginning on page 364, and again beginning on page 431, is a form of *regularization* of an inverse problem for which we have developed a set of approximate linear equations in the variable β, $y \approx X\beta$. One possible solution to the underlying problem is characterized by the equations $X^{\mathrm{T}}X\beta = X^{\mathrm{T}}y$. This solution satisfies a criterion that in many cases is not an essential aspect of the inverse problem itself. (I am using "inverse problem" here in its usual sense; here "inverse" does not refer to an inverse of a matrix.)

The matrix $X^{\mathrm{T}}X$ may be ill-conditioned, or even singular, but its eigenvalues are nonnegative. The matrix $X^{\mathrm{T}}X + \lambda I$ for $\lambda > 0$, however, is nonsingular and has a smaller condition number,

$$\kappa_2(X^{\mathrm{T}}X + \lambda I) < \kappa_2(X^{\mathrm{T}}X), \tag{6.12}$$

as we have seen above.

In applications, the ill-conditioning of $X^{\mathrm{T}}X$ may imply other issues relating to the statistical questions being addressed in the inverse problem.

6.1.3 Numerical Accuracy

The condition numbers we have defined are useful indicators of the accuracy we may expect when solving a linear system $Ax = b$. Suppose the entries of the matrix A and the vector b are accurate to approximately p decimal digits, so we have the system

$$(A + \delta A)(x + \delta x) = b + \delta b$$

with

$$\frac{\|\delta A\|}{\|A\|} \approx 10^{-p}$$

and

$$\frac{\|\delta b\|}{\|b\|} \approx 10^{-p}.$$

Assume A is nonsingular, and suppose that the condition number with respect to inversion, $\kappa(A)$, is approximately 10^t, so

$$\kappa(A)\frac{\|\delta A\|}{\|A\|} \approx 10^{t-p}.$$

Ignoring the approximation of b (that is, assuming $\delta b = 0$), we can write

$$\delta x = -A^{-1}\delta A(x + \delta x),$$

which, together with the triangular inequality and inequality (3.280) on page 165, yields the bound

$$\|\delta x\| \leq \|A^{-1}\|\,\|\delta A\|\,\big(\|x\| + \|\delta x\|\big).$$

Using equation (6.7) with this, we have

$$\|\delta x\| \leq \kappa(A)\frac{\|\delta A\|}{\|A\|}\big(\|x\| + \|\delta x\|\big)$$

or

$$\left(1 - \kappa(A)\frac{\|\delta A\|}{\|A\|}\right)\|\delta x\| \leq \kappa(A)\frac{\|\delta A\|}{\|A\|}\|x\|.$$

If the condition number is not too large relative to the precision (that is, if $10^{t-p} \ll 1$), then we have

$$\begin{aligned}\frac{\|\delta x\|}{\|x\|} &\approx \kappa(A)\frac{\|\delta A\|}{\|A\|} \\ &\approx 10^{t-p}.\end{aligned} \tag{6.13}$$

Expression (6.13) provides a rough bound on the accuracy of the solution in terms of the precision of the data and the condition number of the coefficient matrix. This result must be used with some care, however, and there are cases where the bound is violated or where it is not very tight. Rust (1994), among others, points out failures of the condition number for setting bounds on the accuracy of the solution.

Another consideration in the practical use of expression (6.13) is the fact that the condition number is usually not known, and methods for computing it suffer from the same rounding problems as the solution of the linear system itself. In Sect. 11.4, we describe ways of estimating the condition number, but as the discussion there indicates, these estimates are often not very reliable.

We would expect the norms in the expression (6.13) to be larger for larger size problems. The approach taken above addresses a type of "total" error. It may be appropriate to scale the norms to take into account the number of elements. Chaitin-Chatelin and Frayssé (1996) discuss error bounds for individual elements of the solution vector and condition measures for elementwise error.

Another approach to determining the accuracy of a solution is to use random perturbations of A and/or b and then to estimate the effects of the perturbations on x. Stewart (1990) discusses ways of doing this. Stewart's method estimates error measured by a norm, as in expression (6.13). Kenney and Laub (1994) and Kenney, Laub, and Reese (1998) describe an estimation method to address elementwise error.

Higher accuracy in computations for solving linear systems can be achieved in various ways: multiple precision, interval arithmetic, and residue arithmetic (see pages 495 and 493). Stallings and Boullion (1972) and Keller-McNulty and Kennedy (1986) describe ways of using residue arithmetic in some linear computations for statistical applications.

Another way of improving the accuracy is by using iterative refinement, which we discuss in Sect. 6.4.

6.2 Direct Methods for Consistent Systems

There are two general approaches to solving the linear system $Ax = b$, direct and iterative. In this section, we discuss direct methods, which usually proceed by a factorization of the coefficient matrix.

6.2.1 Gaussian Elimination and Matrix Factorizations

The most common direct method for the solution of linear systems is Gaussian elimination. The basic idea in this method is to form equivalent sets of equations, beginning with the system to be solved, $Ax = b$, or

$$a_{1*}^{\mathrm{T}} x = b_1$$

$$\begin{aligned} a_{2*}^{\mathrm{T}} x &= b_2 \\ \ldots &= \ldots \\ a_{n*}^{\mathrm{T}} x &= b_n, \end{aligned}$$

where a_{j*} is the j^{th} row of A. An equivalent set of equations can be formed by a sequence of *elementary operations* on the equations in the given set.

These elementary operations on equations are essentially the same as the elementary operations on the rows of matrices discussed in Sect. 3.2.3 and in Sect. 5.7. There are three kinds of elementary operations:

- an interchange of two equations,

$$\begin{aligned} a_{j*}^{\mathrm{T}} x = b_j &\leftarrow a_{k*}^{\mathrm{T}} x = b_k, \\ a_{k*}^{\mathrm{T}} x = b_k &\leftarrow a_{j*}^{\mathrm{T}} x = b_j, \end{aligned}$$

  which affects two equations simultaneously,
- a scalar multiplication of a given equation,

$$a_{j*}^{\mathrm{T}} x = b_j \quad \leftarrow \quad c a_{j*}^{\mathrm{T}} x = c b_j,$$

  and
- a replacement of a single equation by an axpy operation (a sum of it and a scalar multiple of another equation),

$$a_{j*}^{\mathrm{T}} x = b_j \quad \leftarrow \quad a_{j*}^{\mathrm{T}} x + c a_{k*}^{\mathrm{T}} x = b_j + c b_k.$$

The interchange operation can be accomplished by premultiplication by an elementary permutation matrix (see page 81):

$$E_{jk} A x = E_{jk} b.$$

The scalar multiplication can be performed by premultiplication by an elementary transformation matrix $E_j(c)$, and the axpy operation can be effected by premultiplication by an $E_{jk}(c)$ elementary transformation matrix.

The elementary operation on the equation

$$a_{2*}^{\mathrm{T}} x = b_2$$

in which the first equation is combined with it using $c_1 = -a_{21}/a_{11}$ and $c_2 = 1$ will yield an equation with a zero coefficient for x_1. Generalizing this, we perform elementary operations on the second through the n^{th} equations to yield a set of equivalent equations in which all but the first have zero coefficients for x_1.

Next, we perform elementary operations using the second equation with the third through the n^{th} equations, so that the new third through the n^{th} equations have zero coefficients for x_2. This is the kind of sequence of multiplications by elementary operator matrices shown in equation (3.62) on page 84 and grouped together as L_k in equation (5.25) on page 244.

The sequence of equivalent equations, beginning with $Ax = b$, is

$$
(0)\quad \begin{array}{rcccccc}
a_{11}x_1 & + a_{12}x_2 & + & \cdots & + & a_{1n}x_n & = & b_1 \\
a_{21}x_1 & + a_{22}x_2 & + & \cdots & + & a_{2n}x_n & = & b_2 \\
\vdots & + \quad \vdots & & & & \vdots & & \vdots \\
a_{n1}x_1 & + a_{n2}x_2 & + & \cdots & + & a_{nn}x_n & = & b_n
\end{array}, \tag{6.14}
$$

then $A^{(1)}x = b^{(1)}$, or $L_1Ax = L_1b$,

$$
(1)\quad \begin{array}{rcccccc}
a_{11}x_1 & + a_{12}x_2 & + & \cdots & + & a_{1n}x_n & = & b_1 \\
 & a_{22}^{(1)}x_2 & + & \cdots & + & a_{2n}^{(1)}x_n & = & b_2^{(1)} \\
 & \vdots \quad + & & \cdots & + & \vdots & & \vdots \\
 & a_{n2}^{(1)}x_2 & + & \cdots & + & a_{nn}^{(1)}x_n & = & b_n^{(1)}
\end{array}, \tag{6.15}
$$

$$\vdots \quad \vdots$$

and finally $A^{(n)}x = b^{(n)}$, or $L_{n-1}\cdots L_1Ax = L_{n-1}\cdots L_1b$, or $Ux = L_{n-1}\cdots L_1b$,

$$
(n-1)\quad \begin{array}{rcccccc}
a_{11}x_1 & + a_{12}x_2 & + & \cdots & + & a_{1n}x_n & = & b_1 \\
 & a_{22}^{(1)}x_2 & + & \cdots & + & a_{2n}^{(1)}x_n & = & b_2^{(1)} \\
 & & & & \vdots & \vdots & & \vdots \\
 & & & a_{n-1,n-1}^{(n-2)}x_{n-1} & + & a_{n-1,n}^{(n-2)}x_n & = & b_{n-1}^{(n-2)} \\
 & & & & & a_{nn}^{(n-1)}x_n & = & b_n^{(n-1)}
\end{array}. \tag{6.16}
$$

Recalling equation (5.28), we see that the last system is $Ux = L^{-1}b$. This system is easy to solve because the coefficient matrix is upper triangular. The last equation in the system yields

$$x_n = \frac{b_n^{(n-1)}}{a_{nn}^{(n-1)}}.$$

By back substitution, we get

$$x_{n-1} = \frac{(b_{n-1}^{(n-2)} - a_{n-1,n}^{(n-2)}x_n)}{a_{n-1,n-1}^{(n-2)}},$$

and we obtain the rest of the xs in a similar manner. This back substitution is equivalent to forming

$$x = U^{-1}L^{-1}b, \tag{6.17}$$

or $x = A^{-1}b$ with $A = LU$.

Gaussian elimination consists of two steps: the forward reduction, which is of order $\mathrm{O}(n^3)$, and the back substitution, which is of order $\mathrm{O}(n^2)$.

6.2.1.1 Pivoting

The only obvious problem with this method arises if some of the $a_{kk}^{(k-1)}$s used as divisors are zero (or very small in magnitude). These divisors are called "pivot elements".

Suppose, for example, we have the equations

$$\begin{aligned} 0.0001x_1 + x_2 &= 1, \\ x_1 + x_2 &= 2. \end{aligned}$$

The solution is $x_1 = 1.0001$ and $x_2 = 0.9999$. Suppose we are working with three digits of precision (so our solution is $x_1 = 1.00$ and $x_2 = 1.00$). After the first step in Gaussian elimination, we have

$$\begin{aligned} 0.0001x_1 + \qquad x_2 &= \qquad 1, \\ -10,000x_2 &= -10,000, \end{aligned}$$

and so the solution by back substitution is $x_2 = 1.00$ and $x_1 = 0.000$. The L_2 condition number of the coefficient matrix is 2.618, so even though the coefficients vary greatly in magnitude, we certainly would not expect any difficulty in solving these equations.

A simple solution to this potential problem is to interchange the equation having the small leading coefficient with an equation below it. Thus, in our example, we first form

$$\begin{aligned} x_1 + x_2 &= 2, \\ 0.0001x_1 + x_2 &= 1, \end{aligned}$$

so that after the first step we have

$$\begin{aligned} x_1 + x_2 &= 2, \\ x_2 &= 1, \end{aligned}$$

and the solution is $x_2 = 1.00$ and $x_1 = 1.00$, which is correct to three digits.

Another strategy would be to interchange the column having the small leading coefficient with a column to its right. Both the row interchange and the column interchange strategies could be used simultaneously, of course. These processes, which obviously do not change the solution, are called *pivoting*. The equation or column to move into the active position may be chosen in such a way that the magnitude of the new diagonal element is the largest possible.

Performing only row interchanges, so that at the k^{th} stage the equation with

$$\max_{i=k}^{n} |a_{ik}^{(k-1)}|$$

is moved into the k^{th} row, is called *partial pivoting*. Performing both row interchanges and column interchanges, so that

$$\max_{i=k:n;j=k:n} |a_{ij}^{(k-1)}|$$

is moved into the k^{th} diagonal position, is called *complete pivoting*. See Exercises 6.2a and 6.2b.

It is always important to distinguish descriptions of effects of actions from the actions that are actually carried out in the computer. Pivoting is "interchanging" rows or columns. We would usually do something like that in the computer only when we are finished and want to produce some output. In the computer, a row or a column is determined by the index identifying the row or column. All we do for pivoting is to keep track of the indices that we have permuted.

There are many more computations required in order to perform complete pivoting than are required to perform partial pivoting. Gaussian elimination with complete pivoting can be shown to be stable; that is, the algorithm yields an exact solution to a slightly perturbed system, $(A + \delta A)x = b$. (We discuss stability on page 502.) For Gaussian elimination with partial pivoting, there are examples that show that it is not stable. These examples are somewhat contrived, however, and experience over many years has indicated that Gaussian elimination with partial pivoting is stable for most problems occurring in practice. For this reason, together with the computational savings, Gaussian elimination with partial pivoting is one of the most commonly used methods for solving linear systems. See Golub and Van Loan (1996) for a further discussion of these issues.

There are two modifications of partial pivoting that result in stable algorithms. One is to add one step of iterative refinement (see Sect. 6.4, page 286) following each pivot. It can be shown that Gaussian elimination with partial pivoting together with one step of iterative refinement is unconditionally stable (Skeel 1980). Another modification is to consider two columns for possible interchange in addition to the rows to be interchanged. This does not require nearly as many computations as complete pivoting does. Higham (1997) shows that this method, suggested by Bunch and Kaufman (1977) and used in LINPACK and LAPACK (see page 558), is stable.

6.2.1.2 Nonfull Rank and Nonsquare Systems

The existence of an x that solves the linear system $Ax = b$ depends on that system being consistent; it does not depend on A being square or of full rank. The methods discussed above also apply in this case. (See the discussion of LU and QR factorizations for nonfull rank and nonsquare matrices on pages 243 and 251.) In applications, it is often annoying that many software developers do not provide capabilities for handling such systems. Many of the standard programs for solving systems provide solutions only if A is square and of full rank. This is a poor design decision.

6.2.2 Choice of Direct Method

Direct methods of solving linear systems all use some form of matrix factorization, as discussed in Chap. 5. The LU factorization is the most commonly used method to solve a linear system.

For certain patterned matrices, other direct methods may be more efficient. If a given matrix initially has a large number of zeros, it is important to preserve the zeros in the same positions (or in other known positions) in the matrices that result from operations on the given matrix. This helps to avoid unnecessary computations. The iterative methods discussed in the next section are often more useful for sparse matrices.

Another important consideration is how easily an algorithm lends itself to implementation on advanced computer architectures. Many of the algorithms for linear algebra can be vectorized easily. It is now becoming more important to be able to parallelize the algorithms. The iterative methods discussed in the next section can often be parallelized more easily.

6.3 Iterative Methods for Consistent Systems

In iterative methods for solving the linear system $Ax = b$, we begin with starting point $x^{(0)}$, which we consider to be an approximate solution, and then move through a sequence of successive approximations $x^{(1)}, x^{(2)}, \ldots$, that ultimately (it is hoped!) converge to a solution. The user must specify a convergence criterion to determine when the approximation is close enough to the solution. The criterion may be based on successive changes in the solution $x^{(k)} - x^{(k-1)}$ or on the difference $\|Ax^{(k)} - b\|$.

Iterative methods may be particularly useful for very large systems because it may not be necessary to have the entire A matrix available for computations in each step. These methods are also useful for sparse systems. Also, as mentioned above, the iterative algorithms can often be parallelized. Heath, Ng, and Peyton (1991) review various approaches to parallelizing iterative methods for solving sparse systems.

6.3.1 The Gauss-Seidel Method with Successive Overrelaxation

One of the simplest iterative procedures is the *Gauss-Seidel method*. In this method, first rearrange the equations if necessary so that no diagonal element of the coefficient matrix is 0. We then begin with an initial approximation to the solution, $x^{(0)}$. We next compute an update for the first element of x:

$$x_1^{(1)} = \frac{1}{a_{11}} \left(b_1 - \sum_{j=2}^{n} a_{1j} x_j^{(0)} \right).$$

(If a_{11} is zero or very small in absolute value, we first rearrange the equations; that is, we pivot.)

Continuing in this way for the other elements of x, we have for $i = 1, \ldots, n$

$$x_i^{(1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(0)} \right),$$

where no sums are performed if the upper limit is smaller than the lower limit. After getting the approximation $x^{(1)}$, we then continue this same kind of iteration for $x^{(2)}, x^{(3)}, \ldots$.

We continue the iterations until a convergence criterion is satisfied. As we discuss in Sect. 10.3.4, this criterion may be of the form

$$\Delta \left(x^{(k)}, x^{(k-1)} \right) \leq \epsilon,$$

where $\Delta \left(x^{(k)}, x^{(k-1)} \right)$ is a measure of the difference of $x^{(k)}$ and $x^{(k-1)}$, such as $\|x^{(k)} - x^{(k-1)}\|$. We may also base the convergence criterion on $\|r^{(k)} - r^{(k-1)}\|$, where $r^{(k)} = b - Ax^{(k)}$.

The Gauss-Seidel iterations can be thought of as beginning with a rearrangement of the original system of equations as

$$\begin{array}{rcccccccc}
a_{11}x_1 & & & & & = b_1 & - a_{12}x_2 & \cdots - & a_{1n}x_n \\
a_{21}x_1 & + & a_{22}x_2 & & & = b_2 & & \cdots - & a_{2n}x_n \\
\vdots & + & \vdots & & \vdots & \vdots & & & \\
a_{(n-1)1}x_1 & + & a_{(n-1)2}x_2 & + \cdots & & = b_{n-1} & & - & a_{nn}x_n \\
a_{n1}x_1 & + & a_{n2}x_2 & + \cdots + & a_{nn}x_n & = b_n & & &
\end{array} \tag{6.18}$$

In this form, we identify three matrices: a diagonal matrix D, a lower triangular L with 0s on the diagonal, and an upper triangular U with 0s on the diagonal:

$$(D + L)x = b - Ux.$$

We can write this entire sequence of Gauss-Seidel iterations in terms of these three fixed matrices:

$$x^{(k+1)} = (D + L)^{-1}\left(-Ux^{(k)} + b\right). \tag{6.19}$$

6.3.1.1 Convergence of the Gauss-Seidel Method

This method will converge for any arbitrary starting value $x^{(0)}$ if and only if

$$\rho((D + L)^{-1}U) < 1. \tag{6.20}$$

We see this by considering the difference $x^{(k)} - x$ and writing

$$x^{(k+1)} - x = (D + L)^{-1}U(x^{(k)} - x) = ((D + L)^{-1}U)^k(x^{(0)} - x).$$

From (3.314) on page 173, we see that this residual goes to 0 if and only if $\rho((D + L)^{-1}U) < 1$.

Moreover, as we stated informally on page 173, the rate of convergence increases with decreasing spectral radius.

Notice that pivoting rearranges the equations (6.18) (resulting in different D, L, and U) and thus changes the value of $\rho((D + L)^{-1}U)$; hence pivoting can cause the Gauss-Seidel method to be viable even when it is not with the original system (see Exercise 6.2e).

6.3.1.2 Successive Overrelaxation

The Gauss-Seidel method may be unacceptably slow, so it may be modified so that the update is a weighted average of the regular Gauss-Seidel update and the previous value. This kind of modification is called *successive overrelaxation*, or *SOR*. Instead of equation (6.19), the update is given by

$$\frac{1}{\omega}(D + \omega L)\, x^{(k+1)} = \frac{1}{\omega}\big((1 - \omega)D - \omega U\big)x^{(k)} + b, \tag{6.21}$$

where the relaxation parameter ω is usually chosen to be between 0 and 1. For $\omega = 1$ the method is the ordinary Gauss-Seidel method; see Exercises 6.2c, 6.2d, and 6.2g.

6.3.2 Conjugate Gradient Methods for Symmetric Positive Definite Systems

In the Gauss-Seidel methods the convergence criterion is based on successive differences in the solutions $x^{(k)}$ and $x^{(k-1)}$ or in the residuals $r^{(k)}$ and $r^{(k-1)}$. Other iterative methods focus directly on the magnitude of the residual

$$r^{(k)} = b - Ax^{(k)}. \tag{6.22}$$

We seek a value $x^{(k)}$ such that the residual is small (in some sense). Methods that minimize $\|r^{(k)}\|_2$ are called minimal residual methods. Two names associated with minimum residual methods are MINRES and GMRES, which are names of specific algorithms.

For a system with a symmetric positive definite coefficient matrix A, it turns out that the best iterative method is based on minimizing the conjugate L_2 norm (see equation (3.94))

$$\|r^{(k)\mathrm{T}}A^{-1}r^{(k)}\|_2.$$

A method based on this minimization problem is called a conjugate gradient method.

6.3.2.1 The Conjugate Gradient Method

The problem of solving the linear system $Ax = b$ is equivalent to finding the minimum of the function

$$f(x) = \frac{1}{2}x^{\mathrm{T}}Ax - x^{\mathrm{T}}b. \tag{6.23}$$

By setting the derivative of f to 0, we see that a stationary point of f occurs at the point x where $Ax = b$ (see Sect. 4.4).

If A is positive definite, the (unique) minimum of f is at $x = A^{-1}b$, and the value of f at the minimum is $-\frac{1}{2}b^{\mathrm{T}}Ab$. The minimum point can be approached iteratively by starting at a point $x^{(0)}$, moving to a point $x^{(1)}$ that yields a smaller value of the function, and continuing to move to points yielding smaller values of the function. The k^{th} point is $x^{(k-1)} + \alpha^{(k-1)}p^{(k-1)}$, where $\alpha^{(k-1)}$ is a scalar and $p^{(k-1)}$ is a vector giving the direction of the movement. Hence, for the k^{th} point, we have the linear combination

$$x^{(k)} = x^{(0)} + \alpha^{(1)}p^{(1)} + \cdots + \alpha^{(k-1)}p^{(k-1)}.$$

At the point $x^{(k)}$, the function f decreases most rapidly in the direction of the negative gradient, $-\nabla f(x^{(k)})$, which is just the residual,

$$-\nabla f(x^{(k)}) = r^{(k)}.$$

If this residual is 0, no movement is indicated because we are at the solution.

Moving in the direction of steepest descent may cause a very slow convergence to the minimum. (The curve that leads to the minimum on the quadratic surface is obviously not a straight line. The direction of steepest descent changes as we move to a new point $x^{(k+1)}$.) A good choice for the sequence of directions $p^{(1)}, p^{(2)}, \ldots$ is such that

$$(p^{(k)})^{\mathrm{T}}Ap^{(i)} = 0, \quad \text{for } i = 1, \ldots, k-1. \tag{6.24}$$

Such a vector $p^{(k)}$ is A-conjugate to $p^{(1)}, p^{(2)}, \ldots p^{(k-1)}$ (see page 94). Given a current point $x^{(k)}$ and a direction to move $p^{(k)}$ to the next point, we must also choose a distance $\alpha^{(k)}\|p^{(k)}\|$ to move in that direction. We then have the next point,

$$x^{(k+1)} = x^{(k)} + \alpha^{(k)}p^{(k)}. \tag{6.25}$$

(Notice that here, as often in describing algorithms in linear algebra, we use Greek letters, such as α, to denote scalar quantities.)

We choose the directions as in Newton steps, so the first direction is $Ar^{(0)}$ (see Sect. 4.4.2). The paths defined by the directions $p^{(1)}, p^{(2)}, \ldots$ in equation (6.24) are called the conjugate gradients. A conjugate gradient method for solving the linear system is shown in Algorithm 6.1.

Algorithm 6.1 The conjugate gradient method for solving the symmetric positive definite system $Ax = b$, starting with $x^{(0)}$

0. Input stopping criteria, ϵ and $k_{\max}$.
   Set $k = 0$; $r^{(k)} = b - Ax^{(k)}$; $s^{(k)} = Ar^{(k)}$; $p^{(k)} = s^{(k)}$; and $\gamma^{(k)} = \|s^{(k)}\|^2$.
1. If $\gamma^{(k)} \leq \epsilon$, set $x = x^{(k)}$ and terminate.
2. Set $q^{(k)} = Ap^{(k)}$.
3. Set $\alpha^{(k)} = \frac{\gamma^{(k)}}{\|q^{(k)}\|^2}$.
4. Set $x^{(k+1)} = x^{(k)} + \alpha^{(k)} p^{(k)}$.
5. Set $r^{(k+1)} = r^{(k)} - \alpha^{(k)} q^{(k)}$.
6. Set $s^{(k+1)} = Ar^{(k+1)}$.
7. Set $\gamma^{(k+1)} = \|s^{(k+1)}\|^2$.
8. Set $p^{(k+1)} = s^{(k+1)} + \frac{\gamma^{(k+1)}}{\gamma^{(k)}} p^{(k)}$.
9. If $k < k_{\max}$,
   set $k = k + 1$ and go to step 1;
   otherwise
   issue message that
   "algorithm did not converge in $k_{\max}$ iterations". ∎

There are various ways in which the computations in Algorithm 6.1 could be arranged. Although any vector norm could be used in Algorithm 6.1, the L_2 norm is the most common one.

This method, like other iterative methods, is more appropriate for large systems. ("Large" in this context means bigger than 1000×1000.)

In exact arithmetic, the conjugate gradient method should converge in n steps for an $n \times n$ system. In practice, however, its convergence rate varies widely, even for systems of the same size. Its convergence rate generally decreases with increasing L_2 condition number (which is a function of the maximum and minimum nonzero eigenvalues), but that is not at all the complete story. The rate depends in a complicated way on all of the eigenvalues. The more spread out the eigenvalues are, the slower the rate. For different systems with roughly the same condition number, the convergence is faster if all eigenvalues are in two clusters around the maximum and minimum values. See Greenbaum and Strakoš (1992) for an analysis of the convergence rates.

6.3.2.2 Krylov Methods

Notice that the steps in the conjugate gradient algorithm involve the matrix A only through linear combinations of its rows or columns; that is, in any iteration, only a vector of the form Av or $A^T w$ is used. The conjugate gradient method and related procedures, called *Lanczos methods*, move through a *Krylov space* in the progression to the solution. A Krylov space is the k-dimensional vector space of order n generated by the $n \times n$ matrix A and the vector v by forming the basis $\{v, Av, A^2 v, \ldots, A^{k-1} v\}$. We often denote this space as $\mathcal{K}_k(A, v)$ or just as $\mathcal{K}_k$:

$$\mathcal{K}_k = \mathcal{V}(\{v, Av, A^2v, \ldots, A^{k-1}v\}). \tag{6.26}$$

Methods for computing eigenvalues are often based on Krylov spaces.

6.3.2.3 GMRES Methods

The conjugate gradient method seeks to minimize the residual vector in equation (6.22), $r^{(k)} = b - Ax^{(k)}$, and the convergence criterion is based on the linear combinations of the columns of the coefficient matrix formed by that vector, $\|Ar^{(k)}\|$.

The generalized minimal residual (GMRES) method of Saad and Schultz (1986) for solving $Ax = b$ begins with an approximate solution $x^{(0)}$ and takes $x^{(k)}$ as $x^{(k-1)} + z^{(k)}$, where $z^{(k)}$ is the solution to the minimization problem,

$$\min_{z \in \mathcal{K}_k(A, r^{(k-1)})} \|r^{(k-1)} - Az\|,$$

where, as before, $r^{(k)} = b - Ax^{(k)}$. This minimization problem is a constrained least squares problem. The speed of convergence of GMRES depends very strongly on the arrangements of the computations. See Walker (1988) and Walker and Zhou (1994) for details of the methods. Brown and Walker (1997) consider the behavior of GMRES when the coefficient matrix is singular and give conditions for GMRES to converge to a solution of minimum length (the solution corresponding to the Moore-Penrose inverse; see Sect. 6.6.3, page 293).

6.3.2.4 Preconditioning

As we mentioned above, the convergence rate of the conjugate gradient method depends on the distribution of the eigenvalues in rather complicated ways. The ratio of the largest to the smallest (that is, the L_2 condition number is important) and the convergence rate for the conjugate gradient method is slower for larger L_2 condition numbers. The rate also is slower if the eigenvalues are spread out, especially if there are several eigenvalues near the largest or smallest. This phenomenon is characteristic of other Krylov space methods.

One way of addressing the problem of slow convergence of iterative methods is by *preconditioning*; that is, by replacing the system $Ax = b$ with another system,

$$M^{-1}Ax = M^{-1}b, \tag{6.27}$$

where M is very similar (by some measure) to A, but the system $M^{-1}Ax = M^{-1}b$ has a better condition for the problem at hand. We choose M to be symmetric and positive definite, and such that $Mx = b$ is easy to solve. If M is an approximation of A, then $M^{-1}A$ should be well-conditioned; its eigenvalues should all be close to each other.

A problem with applying the conjugate gradient method to the preconditioned system $M^{-1}Ax = M^{-1}b$ is that $M^{-1}A$ may not be symmetric. We can form an equivalent symmetric system, however, by decomposing the symmetric positive definite M as $M = VCV^{\mathrm{T}}$ and then

$$M^{-1/2} = V\mathrm{diag}((1/\sqrt{c_{11}}, \ldots, 1/\sqrt{c_{nn}}))V^{\mathrm{T}},$$

as in equation (3.273), after inverting the positive square roots of C. Multiplying both sides of $M^{-1}Ax = M^{-1}b$ by $M^{1/2}$, inserting the factor $M^{-1/2}M^{1/2}$, and arranging terms yields

$$(M^{-1/2}AM^{-1/2})M^{1/2}x = M^{-1/2}b.$$

This can all be done and Algorithm 6.1 can be modified without explicit formation of and multiplication by $M^{1/2}$. The preconditioned conjugate gradient method is shown in Algorithm 6.2.

Algorithm 6.2 The preconditioned conjugate gradient method for solving the symmetric positive definite system $Ax = b$, starting with $x^{(0)}$

0. Input stopping criteria, ϵ and $k_{\max}$.
   Set $k = 0$; $r^{(k)} = b - Ax^{(k)}$; $s^{(k)} = Ar^{(k)}$; $p^{(k)} = M^{-1}s^{(k)}$; $y^{(k)} = M^{-1}r^{(k)}$; and $\gamma^{(k)} = y^{(k)\mathrm{T}}s^{(k)}$.
1. If $\gamma^{(k)} \leq \epsilon$, set $x = x^{(k)}$ and terminate.
2. Set $q^{(k)} = Ap^{(k)}$.
3. Set $\alpha^{(k)} = \frac{\gamma^{(k)}}{\|q^{(k)}\|^2}$.
4. Set $x^{(k+1)} = x^{(k)} + \alpha^{(k)}p^{(k)}$.
5. Set $r^{(k+1)} = r^{(k)} - \alpha^{(k)}q^{(k)}$.
6. Set $s^{(k+1)} = Ar^{(k+1)}$.
7. Set $y^{(k+1)} = M^{-1}r^{(k+1)}$.
8. Set $\gamma^{(k+1)} = y^{(k+1)\mathrm{T}}s^{(k+1)}$.
9. Set $p^{(k+1)} = M^{-1}s^{(k+1)} + \frac{\gamma^{(k+1)}}{\gamma^{(k)}}p^{(k)}$.
10. If $k < k_{\max}$,
    set $k = k + 1$ and go to step 1;
    otherwise
    issue message that
    "algorithm did not converge in $k_{\max}$ iterations". ∎

The choice of an appropriate matrix M is not an easy problem, and we will not consider the methods here. Benzi (2002) provides a survey of preconditioning methods. We will also mention the preconditioned conjugate gradient method in Sect. 7.1.4, but there, again, we will refer the reader to other sources for details.

6.3.3 Multigrid Methods

Iterative methods have important applications in solving differential equations. The solution of differential equations by a finite difference discretization involves the formation of a grid. The solution process may begin with a fairly coarse grid on which a solution is obtained. Then a finer grid is formed, and the solution is interpolated from the coarser grid to the finer grid to be used as a starting point for a solution over the finer grid. The process is then continued through finer and finer grids. If all of the coarser grids are used throughout the process, the technique is a *multigrid* method. There are many variations of exactly how to do this. Multigrid methods are useful solution techniques for differential equations.

6.4 Iterative Refinement

Once an approximate solution $x^{(0)}$ to the linear system $Ax = b$ is available, *iterative refinement* can yield a solution that is closer to the true solution. The residual

$$r = b - Ax^{(0)}$$

is used for iterative refinement. Clearly, if $h = A^+ r$, then $x^{(0)} + h$ is a solution to the original system.

The problem considered here is not just an iterative solution to the linear system, as we discussed in Sect. 6.3. Here, we assume $x^{(0)}$ was computed accurately given the finite precision of the computer. In this case, it is likely that r cannot be computed accurately enough to be of any help. If, however, r can be computed using a higher precision, then a useful value of h can be computed. This process can then be iterated as shown in Algorithm 6.3.

Algorithm 6.3 Iterative refinement of the solution to $Ax = b$, starting with $x^{(0)}$

0. Input stopping criteria, ϵ and $k_{\max}$.
   Set $k = 0$.
1. Compute $r^{(k)} = b - Ax^{(k)}$ in higher precision.
2. Compute $h^{(k)} = A^+ r^{(k)}$.
3. Set $x^{(k+1)} = x^{(k)} + h^{(k)}$.
4. If $\|h^{(k)}\| \leq \epsilon \|x^{(k+1)}\|$, then
   set $x = x^{(k+1)}$ and terminate; otherwise,
   if $k < k_{\max}$,
   set $k = k + 1$ and go to step 1;
   otherwise,
   issue message that
   "algorithm did not converge in $k_{\max}$ iterations". ∎

In step 2, if A is of full rank then A^+ is A^{-1}. Also, as we have emphasized already, the fact that we write *an expression such as* A^+r *does not mean that we compute* A^+. The norm in step 4 is usually chosen to be the ∞ norm. The algorithm may not converge, so it is necessary to have an alternative exit criterion, such as a maximum number of iterations.

The use of iterative refinement as a general-purpose method is severely limited by the need for higher precision in step 1. On the other hand, if computations in higher precision can be performed, they can be applied to step 2—or just in the original computations for $x^{(0)}$. In terms of both accuracy and computational efficiency, using higher precision throughout is usually better.

6.5 Updating a Solution to a Consistent System

In applications of linear systems, it is often the case that after the system $Ax = b$ has been solved, the right-hand side is changed and the system $Ax = c$ must be solved. If the linear system $Ax = b$ has been solved by a direct method using one of the factorizations discussed in Chap. 5, the factors of A can be used to solve the new system $Ax = c$. If the right-hand side is a small perturbation of b, say $c = b + \delta b$, an iterative method can be used to solve the new system quickly, starting from the solution to the original problem.

If the coefficient matrix in a linear system $Ax = b$ is perturbed to result in the system $(A + \delta A)x = b$, it may be possible to use the solution x_0 to the original system efficiently to arrive at the solution to the perturbed system. One way, of course, is to use x_0 as the starting point in an iterative procedure. Often, in applications, the perturbations are of a special type, such as

$$\widetilde{A} = A - uv^{\mathrm{T}},$$

where u and v are vectors. (This is a "rank-one" perturbation of A, and when the perturbed matrix is used as a transformation, it is called a "rank-one" update. As we have seen, a Householder reflection is a special rank-one update.) Assuming A is an $n \times n$ matrix of full rank, it is easy to write $\widetilde{A}^{-1}$ in terms of A^{-1}:

$$\widetilde{A}^{-1} = A^{-1} + \alpha(A^{-1}u)(v^{\mathrm{T}}A^{-1}) \tag{6.28}$$

with

$$\alpha = \frac{1}{1 - v^{\mathrm{T}}A^{-1}u}.$$

These are called the Sherman-Morrison formulas (from Sherman and Morrison 1950). $\widetilde{A}^{-1}$ exists so long as $v^{\mathrm{T}}A^{-1}u \neq 1$. Because $x_0 = A^{-1}b$, the solution to the perturbed system is

$$\tilde{x}_0 = x_0 + \frac{(A^{-1}u)(v^{\mathrm{T}}x_0)}{(1 - v^{\mathrm{T}}A^{-1}u)}.$$

If the perturbation is more than rank one (that is, if the perturbation is

$$\widetilde{A} = A - UV^{\mathrm{T}}, \tag{6.29}$$

where U and V are $n \times m$ matrices with $n \geq m$), a generalization of the Sherman-Morrison formula, sometimes called the Woodbury formula, is

$$\widetilde{A}^{-1} = A^{-1} + A^{-1}U(I_m - V^{\mathrm{T}}A^{-1}U)^{-1}V^{\mathrm{T}}A^{-1} \tag{6.30}$$

(from Woodbury 1950). The solution to the perturbed system is easily seen to be

$$\tilde{x}_0 = x_0 + A^{-1}U(I_m - V^{\mathrm{T}}A^{-1}U)^{-1}V^{\mathrm{T}}x_0.$$

As we have emphasized many times, we rarely compute the inverse of a matrix, and so the Sherman-Morrison-Woodbury formulas are not used directly. Having already solved $Ax = b$, it should be easy to solve another system, say $Ay = u_i$, where u_i is a column of U. If m is relatively small, as it is in most applications of this kind of update, there are not many systems $Ay = u_i$ to solve. Solving these systems, of course, yields $A^{-1}U$, the most formidable component of the Sherman-Morrison-Woodbury formula. The system to solve is of order m also.

Occasionally the updating matrices in equation (6.29) may be used with a weighting matrix, so we have $\widetilde{A} = A - UWV^{\mathrm{T}}$. An extension of the Sherman-Morrison-Woodbury formula is

$$(A - UWV^{\mathrm{T}})^{-1} = A^{-1} + A^{-1}U(W^{-1} - V^{\mathrm{T}}A^{-1}U)^{-1}V^{\mathrm{T}}A^{-1}. \tag{6.31}$$

This is sometimes called the Hemes formula. (The attributions of discovery are somewhat murky, and statements made by historians of science of the form "___ was the first to ___" must be taken with a grain of salt; not every discovery has resulted in an available publication. This is particularly true in numerical analysis, where scientific programmers often just develop a method in the process of writing code and have neither the time nor the interest in getting a publication out of it.)

Another situation that requires an update of a solution occurs when the system is augmented with additional equations and more variables:

$$\begin{bmatrix} A & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} x \\ x_+ \end{bmatrix} = \begin{bmatrix} b \\ b_+ \end{bmatrix}.$$

A simple way of obtaining the solution to the augmented system is to use the solution x_0 to the original system in an iterative method. The starting point for a method based on Gauss-Seidel or a conjugate gradient method can be taken as $(x_0, 0)$, or as $(x_0, x_+^{(0)})$ if a better value of $x_+^{(0)}$ is known.

In many statistical applications, the systems are overdetermined, with A being $n \times m$ and $n > m$. In the next section, we consider the general problem of solving overdetermined systems by using least squares, and then in Sect. 6.6.5 we discuss updating a least squares solution to an overdetermined system.

6.6 Overdetermined Systems: Least Squares

In applications, linear systems are often used as models of relationships between one observable variable, a "response", and another group of observable variables, "predictor variables". The model is unlikely to fit exactly any set of observed values of responses and predictor variables. This may be due to effects of other predictor variables that are not included in the model, measurement error, the relationship among the variables being nonlinear, or some inherent randomness in the system. In such applications, we generally take a larger number of observations than there are variables in the system; thus, with each set of observations on the response and associated predictors making up one equation, we have a system with more equations than variables.

An overdetermined system may be written as

$$Xb \approx y, \tag{6.32}$$

where X is $n \times m$ and $\text{rank}(X|y) > m$; that is, the system is not consistent. We have changed the notation slightly from the consistent systems $Ax = b$ that we have been using because now we have in mind statistical applications and in those the notation $y \approx X\beta$ is more common. The problem is to determine a value of b that makes the approximation close in some sense. In applications of linear systems, we refer to this as "fitting" the system, which is referred to as a "model".

Overdetermined systems abound in fitting equations to data. The usual linear regression model is an overdetermined system and we discuss regression problems further in Sect. 9.3.2. We should not confuse *statistical inference* with *fitting equations to data*, although the latter task is a component of the former activity. In this section, we consider some of the more mechanical and computational aspects of the problem.

6.6.0.1 Accounting for an Intercept

Given a set of observations, the i^{th} row of the system $Xb \approx y$ represents the linear relationship between y_i and the corresponding xs in the vector x_i:

$$y_i \approx b_1 x_{1i} + \cdots + b_m x_{mi}.$$

A different formulation of the relationship between y_i and the corresponding xs might include an intercept term:

$$y_i \approx \tilde{b}_0 + \tilde{b}_1 x_{1i} + \cdots + \tilde{b}_m x_{mi}.$$

There are two ways to incorporate this intercept term. One way is just to include a column of 1s in the X matrix. This approach makes the matrix X in equation (6.32) $n \times (m+1)$, or else it means that we merely redefine x_{1i} to be the constant 1. Another way is to assume that the model is an exact fit

for some set of values of y and the xs. If we assume that the model fits $y = 0$ and $x = 0$ exactly, we have a model without an intercept (that is, with a zero intercept).

Often, a reasonable assumption is that the model may have a nonzero intercept, but it fits the means of the set of observations; that is, the equation is exact for $y = \bar{y}$ and $x = \bar{x}$, where the j^{th} element of $\bar{x}$ is the mean of the j^{th} column vector of X. (Students with some familiarity with the subject may think this is a natural consequence of fitting the model. It is not unless the model fitting is by ordinary least squares.) If we require that the fitted equation be exact for the means (or if this happens naturally, as in the case of ordinary least squares), we may center each column by subtracting its mean from each element in the same manner as we centered vectors on page 48. In place of y, we have the vector $y - \bar{y}$. The matrix formed by centering all of the columns of a given matrix is called a centered matrix, and if the original matrix is X, we represent the centered matrix as X_{c} in a notation analogous to what we introduced for centered vectors. If we represent the matrix whose i^{th} column is the constant mean of the i^{th} column of X as $\overline{X}$,

$$X_{\text{c}} = X - \overline{X}.$$

Using the centered data provides two linear systems: a set of approximate equations in which the intercept is ignored and an equation that fits the point that is assumed to be satisfied exactly:

$$\bar{y} = \overline{X}b.$$

In the rest of this section, we will generally ignore the question of an intercept. Except in a method discussed on page 304, the X can be considered to include a column of 1s, to be centered, or to be adjusted by any other point. We will return to this idea of centering the data in Sect. 8.6.3.

6.6.1 Least Squares Solution of an Overdetermined System

Although there may be no b that will make the system in (6.32) an equation, the system can be written as the equation

$$Xb = y - r, \tag{6.33}$$

where r is an n-vector of possibly arbitrary residuals or "errors".

A *least squares* solution $\widehat{b}$ to the system in (6.32) is one such that the Euclidean norm of the vector of residuals is minimized; that is, the solution to the problem

$$\min_b \|y - Xb\|_2. \tag{6.34}$$

The least squares solution is also called the "ordinary least squares" (OLS) fit.

By rewriting the square of this norm as

$$(y - Xb)^{\mathrm{T}}(y - Xb), \tag{6.35}$$

differentiating, and setting it equal to 0, we see that the minimum (of both the norm and its square) occurs at the $\widehat{b}$ that satisfies the square system

$$X^{\mathrm{T}}X\widehat{b} = X^{\mathrm{T}}y. \tag{6.36}$$

The system (6.36) is called the *normal equations*. The matrix $X^{\mathrm{T}}X$ is called the Gram matrix or the Gramian (see Sect. 8.6.1). Its condition determines the expected accuracy of a solution to the least squares problem. As we mentioned in Sect. 6.1, however, because the condition number of $X^{\mathrm{T}}X$ is the square of the condition number of X, it may be better to work directly on X in (6.32) rather than to use the normal equations. The normal equations are useful expressions, however, whether or not they are used in the computations. This is another case where *a formula does not define an algorithm,* as with other cases we have encountered many times. We should note, of course, that any information about the stability of the problem that the Gramian may provide can be obtained from X directly.

6.6.1.1 Orthogonality of Least Squares Residuals to span(X)

The least squares fit to the overdetermined system has a very useful property with important consequences. The least squares fit partitions the space into two interpretable orthogonal spaces. As we see from equation (6.36), the residual vector $y - X\widehat{b}$ is orthogonal to each column in X:

$$X^{\mathrm{T}}(y - X\widehat{b}) = 0. \tag{6.37}$$

This fact illustrates the close relationship of least squares approximations to the Gram-Schmidt transformations discussed on page 38.

A consequence of this orthogonality for models that include an intercept is that the sum of the residuals is 0. (The residual vector is orthogonal to the 1 vector.) Another consequence for models that include an intercept is that the least squares solution provides an exact fit to the mean.

The orthogonality of the residuals of a least squares fit to all columns of X characterizes the least squares fit. To see this, let $\widetilde{b}$ be such that $X^{\mathrm{T}}(y - X\widetilde{b}) = 0$. Now, for any b,

$$\begin{aligned}
\|y - Xb\|_2^2 &= \|y - X\widetilde{b} + (X\widetilde{b} - Xb)\|_2^2 \\
&= \|y - X\widetilde{b}\|_2^2 + 2(y - X\widetilde{b})^{\mathrm{T}}(X\widetilde{b} - Xb) + \|X\widetilde{b} - Xb\|_2^2 \\
&= \|y - X\widetilde{b}\|_2^2 + 2(y - X\widetilde{b})^{\mathrm{T}}X(\widetilde{b} - b) + \|X\widetilde{b} - Xb\|_2^2 \\
&= \|y - X\widetilde{b}\|_2^2 + 2(X^{\mathrm{T}}(y - X\widetilde{b}))^{\mathrm{T}}(\widetilde{b} - b) + \|X\widetilde{b} - Xb\|_2^2 \\
&= \|y - X\widetilde{b}\|_2^2 + \|X\widetilde{b} - Xb\|_2^2 \\
&\geq \|y - X\widetilde{b}\|_2^2;
\end{aligned}$$

hence, $\widetilde{b}$ is a least squares fit.

These properties are so familiar to statisticians that some think that they are essential characteristics of any regression modeling; they are not. We will see in later sections that they do not hold for other approaches to fitting the basic model $y \approx Xb$. The least squares solution, however, has some desirable statistical properties under fairly common distributional assumptions, as we discuss in Chap. 9.

6.6.1.2 Numerical Accuracy in Overdetermined Systems

In Sect. 6.1.3, we discussed numerical accuracy in computations for solving a consistent (square) system of equations and showed how bounds on the numerical error could be expressed in terms of the condition number of the coefficient matrix, which we had defined (on page 269) as the product of norms of the coefficient matrix and its inverse. One of the most useful versions of this condition number is the one using the L_2 matrix norm, which is called the spectral condition number. This is the most commonly used condition number, and we generally just denote it by $\kappa(\cdot)$. The spectral condition number is the ratio of the largest eigenvalue in absolute value to the smallest in absolute value, and this extends easily to a definition of the spectral condition number that applies both to nonsquare matrices and to singular matrices: the *condition number* of a matrix is the ratio of the largest singular value to the smallest nonzero singular value. As we saw on page 162, the nonzero singular values of X are the square roots of the nonzero eigenvalues of $X^{\mathrm{T}}X$; hence

$$\kappa(X^{\mathrm{T}}X) = (\kappa(X))^2. \tag{6.38}$$

The condition number of $X^{\mathrm{T}}X$ is a measure of the numerical accuracy we can expect in solving the normal equations (6.36). Because the condition number of X is smaller, we have an indication that it might be better not to form the normal equations unless we must. It might be better to work just with X. *This is one of the most important principles in numerical linear algebra.* We will work with X instead of $X^{\mathrm{T}}X$ in the next sections.

6.6.2 Least Squares with a Full Rank Coefficient Matrix

If the $n \times m$ matrix X is of full column rank, the least squares solution, from equation (6.36), is $\widehat{b} = (X^{\mathrm{T}}X)^{-1}X^{\mathrm{T}}y$ and is obviously unique. A good way to compute this is to form the QR factorization of X.

First we write $X = QR$, as in equation (5.35) on page 248, where R is as in equation (5.36),

$$R = \begin{bmatrix} R_1 \\ 0 \end{bmatrix},$$

with R_1 an $m \times m$ upper triangular matrix. The squared residual norm (6.35) can be written as

$$\begin{aligned}(y - Xb)^{\mathrm{T}}(y - Xb) &= (y - QRb)^{\mathrm{T}}(y - QRb) \\ &= (Q^{\mathrm{T}}y - Rb)^{\mathrm{T}}(Q^{\mathrm{T}}y - Rb) \\ &= (c_1 - R_1 b)^{\mathrm{T}}(c_1 - R_1 b) + c_2^{\mathrm{T}} c_2, \end{aligned} \tag{6.39}$$

where c_1 is a vector with m elements and c_2 is a vector with $n - m$ elements, such that

$$Q^{\mathrm{T}}y = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}. \tag{6.40}$$

Because the squared norm (6.35), and hence, the expression (6.39), is nonnegative, the minimum of the squared residual norm in equation (6.39) occurs when $(c_1 - R_1 b)^{\mathrm{T}}(c_1 - R_1 b) = 0$; that is, when $(c_1 - R_1 b) = 0$, or

$$R_1 b = c_1. \tag{6.41}$$

We could also use differentiation to find the minimum of equation (6.39), because in that case the derivative of $c_2^{\mathrm{T}} c_2$ with respect to b is 0.

Because R_1 is triangular, the system is easy to solve: $\widehat{b} = R_1^{-1} c_1$. From equation (5.38), we have

$$X^{+} = \begin{bmatrix} R_1^{-1} & 0 \end{bmatrix} Q^{\mathrm{T}}, \tag{6.42}$$

and so we have

$$\widehat{b} = X^{+}y. \tag{6.43}$$

We also see from equation (6.39) that the minimum of the residual norm is $c_2^{\mathrm{T}} c_2$. This is called the *residual sum of squares* in the least squares fit.

6.6.3 Least Squares with a Coefficient Matrix Not of Full Rank

If X is not of full rank (that is, if X has rank $r < m$), the least squares solution is not unique, and in fact a solution is any vector $\widehat{b} = (X^{\mathrm{T}}X)^{-}X^{\mathrm{T}}y$, where $(X^{\mathrm{T}}X)^{-}$ is any generalized inverse. This is a solution to the normal equations (6.36). If X is not of full rank, equation (6.43), $\widehat{b} = X^{+}y$, still provides a least squares solution, however the Moore-Penrose inverse in equation (6.42) would be replaced by one as shown in equation (5.45) on page 251.

The residual corresponding to this solution is

$$y - X(X^{\mathrm{T}}X)^{-}X^{\mathrm{T}}y = (I - X(X^{\mathrm{T}}X)^{-}X^{\mathrm{T}})y.$$

The residual vector is invariant to the choice of generalized inverse. (We will see this important fact in equation (8.48) on page 361.)

6.6.3.1 An Optimal Property of the Solution Using the Moore-Penrose Inverse

The solution corresponding to the Moore-Penrose inverse is unique because, as we have seen, that generalized inverse is unique. That solution is interesting for another reason, however: the b from the Moore-Penrose inverse has the minimum L_2-norm of all solutions.

To see that this solution has minimum norm, first factor X, as in equation (5.43) on page 251,

$$X = QRU^{\mathrm{T}},$$

and form the Moore-Penrose inverse as in equation (5.45):

$$X^{+} = U \begin{bmatrix} R_1^{-1} & 0 \\ 0 & 0 \end{bmatrix} Q^{\mathrm{T}}.$$

Then

$$\widehat{b} = X^{+} y \tag{6.44}$$

is a least squares solution, just as in the full rank case. Now, let

$$Q^{\mathrm{T}} y = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix},$$

as in equation (6.40), except ensure that c_1 has exactly r elements and c_2 has $n - r$ elements, and let

$$U^{\mathrm{T}} b = \begin{pmatrix} z_1 \\ z_2 \end{pmatrix},$$

where z_1 has r elements. We proceed as in the equations (6.39). We seek to minimize $\|y - Xb\|_2$ (which is the square root of the expression in equations (6.39)); and because multiplication by an orthogonal matrix does not change the norm, we have

$$\begin{aligned} \|y - Xb\|_2 &= \|Q^{\mathrm{T}}(y - XUU^{\mathrm{T}}b)\|_2 \\ &= \left\| \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} - \begin{bmatrix} R_1 & 0 \\ 0 & 0 \end{bmatrix} \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} \right\|_2 \\ &= \left\| \begin{pmatrix} c_1 - R_1 z_1 \\ c_2 \end{pmatrix} \right\|_2 . \end{aligned} \tag{6.45}$$

The residual norm is minimized for $z_1 = R_1^{-1} c_1$ and z_2 arbitrary. However, if $z_2 = 0$, then $\|z\|_2$ is also minimized. Because $U^{\mathrm{T}} b = z$ and U is orthogonal, $\|\widehat{b}\|_2 = \|z\|_2$, and so $\|\widehat{b}\|_2$ is the minimum among all least squares solutions.

6.6.4 Weighted Least Squares

One of the simplest variations on fitting the linear model $Xb \approx y$ is to allow different weights on the observations; that is, instead of each row of X and corresponding element of y contributing equally to the fit, the elements of X and y are possibly weighted differently. The relative weights can be put into an n-vector w and the squared norm in equation (6.35) replaced by a quadratic form in $\mathrm{diag}(w)$. More generally, we form the quadratic form as

$$(y - Xb)^{\mathrm{T}} W (y - Xb), \tag{6.46}$$

where W is a positive definite matrix. Because the weights apply to both y and Xb, there is no essential difference in the weighted or unweighted versions of the problem.

The use of the QR factorization for the overdetermined system in which the weighted norm (6.46) is to be minimized is similar to the development above. It is exactly what we get if we replace $y - Xb$ in equation (6.39) or (6.45) by $W_{\mathrm{C}}(y - Xb)$, where W_{C} is the Cholesky factor of W.

6.6.5 Updating a Least Squares Solution of an Overdetermined System

In Sect. 6.5 on page 287, we considered the problem of updating a given solution to be a solution to a perturbed consistent system. An overdetermined system is often perturbed by adding either some rows or some columns to the coefficient matrix X. This corresponds to including additional equations in the system,

$$\begin{bmatrix} X \\ X_+ \end{bmatrix} b \approx \begin{bmatrix} y \\ y_+ \end{bmatrix},$$

or to adding variables,

$$\begin{bmatrix} X & X_+ \end{bmatrix} \begin{bmatrix} b \\ b_+ \end{bmatrix} \approx y.$$

In either case, if the QR decomposition of X is available, the decomposition of the augmented system can be computed readily. Consider, for example, the addition of k equations to the original system $Xb \approx y$, which has n approximate equations. With the QR decomposition, for the original full rank system, putting $Q^{\mathrm{T}}X$ and $Q^{\mathrm{T}}y$ as partitions in a matrix, we have

$$\begin{bmatrix} R_1 & c_1 \\ 0 & c_2 \end{bmatrix} = Q^{\mathrm{T}} \begin{bmatrix} X & y \end{bmatrix}.$$

Augmenting this with the additional rows yields

$$\begin{bmatrix} R & c_1 \\ 0 & c_2 \\ X_+ & y_+ \end{bmatrix} = \begin{bmatrix} Q^{\mathrm{T}} & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} X & y \\ X_+ & y_+ \end{bmatrix}. \tag{6.47}$$

All that is required now is to apply orthogonal transformations, such as Givens rotations, to the system (6.47) to produce

$$\begin{bmatrix} R_* & c_{1*} \\ 0 & c_{2*} \end{bmatrix},$$

where R_* is an $m \times m$ upper triangular matrix and c_{1*} is an m-vector as before but c_{2*} is an $(n - m + k)$-vector.

The updating is accomplished by applying m rotations to system (6.47) so as to zero out the $(n+q)^{\text{th}}$ row for $q = 1, 2, \ldots, k$. These operations go through an outer loop with $p = 1, 2, \ldots, n$ and an inner loop with $q = 1, 2, \ldots, k$. The operations rotate R through a sequence $R^{(p,q)}$ into R_*, and they rotate X_+ through a sequence $X_+^{(p,q)}$ into 0. At the p, q step, the rotation matrix Q_{pq} corresponding to equation (5.12) on page 239 has

$$\cos\theta = \frac{R_{pp}^{(p,q)}}{r}$$

and

$$\sin\theta = \frac{\left(X_+^{(p,q)}\right)_{qp}}{r},$$

where

$$r = \sqrt{\left(R_{pp}^{(p,q)}\right)^2 + \left(\left(X_+^{(p,q)}\right)_{qp}\right)^2}.$$

Gentleman (1974) and Miller (1992) give Fortran programs that implement this kind of updating. The software, which was published in *Applied Statistics*, is available in `statlib` (see page 619).

6.7 Other Solutions of Overdetermined Systems

The basic form of an overdetermined linear system may be written as in equation (6.32) as

$$Xb \approx y,$$

where X is $n \times m$ and $\text{rank}(X|y) > m$.

As in equation (6.33) in Sect. 6.6.1, we can write this as an equation,

$$Xb = y - r,$$

where r is a vector of residuals. Fitting the equation $y = Xb$ means minimizing r; that is, minimizing some norm of r.

There are various norms that may provide a reasonable fit. In Sect. 6.6, we considered use of the L_2 norm; that is, an ordinary least squares (OLS) fit. There are various other ways of approaching the problem, and we will briefly consider a few of them in this section.

As we have stated before, *we should not confuse statistical inference with fitting equations to data, although the latter task is a component of the former activity.* Applications in statistical data analysis are discussed in Chap. 9. In those applications, we need to make statements (that is, assumptions) about relevant probability distributions. These probability distributions, together with the methods used to collect the data, may indicate specific methods for fitting the equations to the given data. In this section, we continue to address the more mechanical aspects of the problem of fitting equations to data.

6.7.1 Solutions that Minimize Other Norms of the Residuals

A solution to an inconsistent, overdetermined system

$$Xb \approx y,$$

where X is $n \times m$ and $\text{rank}(X|y) > m$, is some value b that makes $y - Xb$ close to zero. We define "close to zero" in terms of a norm on $y - Xb$. The most common norm, of course, is the L_2 norm as in expression (6.34), and the minimization of this norm is straightforward, as we have seen. In addition to the simple analytic properties of the L_2 norm, the least squares solution has some desirable statistical properties under fairly common distributional assumptions, as we have seen.

6.7.1.1 Minimum L_1 Norm Fitting: Least Absolute Values

A common alternative norm is the L_1 norm. The minimum L_1 norm solution is called the *least absolute values* fit or the *LAV* fit. It is not as affected by outlying observations as the least squares fit is.

Consider a simple example. Assume we have observations on a response, $y = (0, 3, 4, 0, 8)$, and on a single predictor variable, $x = (1, 3, 4, 6, 7)$. We have $\bar{y} = 3$ and $\bar{x} = 4.2$. We write the model equation as

$$y \approx b_0 + b_1 x. \tag{6.48}$$

The model with the data is

$$\begin{bmatrix} 0 \\ 3 \\ 4 \\ 0 \\ 8 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 3 \\ 1 & 4 \\ 1 & 6 \\ 1 & 7 \end{bmatrix} b + r. \tag{6.49}$$

A least squares solution yields $\hat{b}_0 = -0.3158$ and $\hat{b}_1 = 0.7895$. With these values, equation (6.48) goes through the mean $(4.2,\ 3)$. The residual vector from this fit is orthogonal to 1 (that is, the sum of the residuals is 0) and to x.

A solution that minimizes the L_1 norm is $\tilde{b}_0 = -1.333$ and $\tilde{b}_1 = 1.333$. The LAV fit may not be unique (although it is in this case). We immediately note that the least absolute deviations fit does not go through the mean of the data, nor is the residual vector orthogonal to the 1 vector and to x. The LAV fit does go through two points in the dataset, however. (This is a special case of one of several interesting properties of LAV fits, which we will not discuss here. The interested reader is referred to Kennedy and Gentle 1980, Chapter 11, for discussion of some of these properties, as well as assumptions

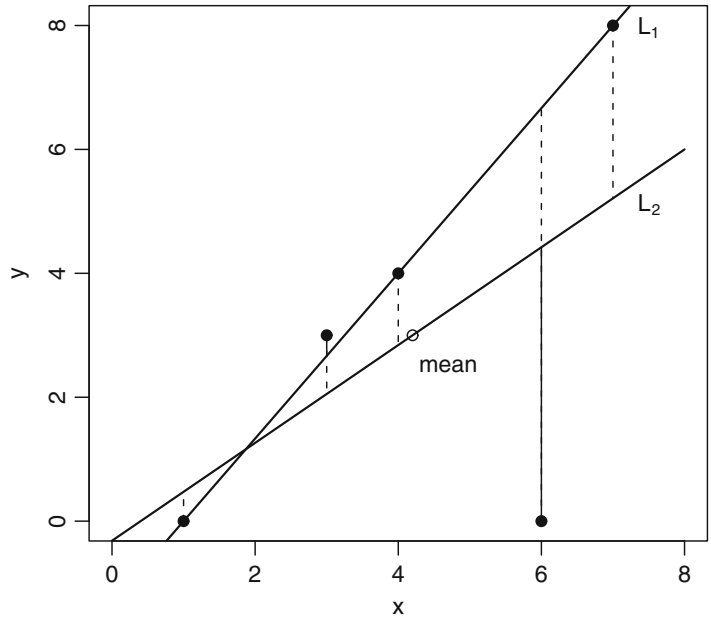


Figure 6.2. OLS and minimum L_1 norm fits

about probability distributions that result in desirable statistical properties for LAV estimators.) A plot of the data, the two fitted lines, and the residuals is shown in Fig. 6.2.

The problem of minimizing the L_1 norm can be formulated as the linear programming problem

$$\begin{aligned} \min_{b} \quad & 1^{\mathrm{T}}(e^+ + e^-) \\ \text{s.t. } & Xb + Ie^+ - Ie^- = y \\ & e^+,\ e^- \geq 0 \end{aligned} \tag{6.50}$$

$$b \quad \text{unrestricted},$$

where e^+ and e^- are nonnegative n-vectors. There are special algorithms that take advantage of the special structure of the problem to speed up the basic linear programming simplex algorithm (see Kennedy and Gentle 1980; Chapter 11).

6.7.1.2 Minimum L_∞ Norm Fitting: Minimax

Another norm that may be useful in some applications is the L_∞ norm. A solution minimizing that norm is called the *least maximum deviation* fit. A least maximum deviation fit is greatly affected by outlying observations. As with the LAV fit, the least maximum deviation fit does not necessarily go through the mean of the data. The least maximum deviation fit also may not be unique.

This problem can also be formulated as a linear programming problem, and as with the least absolute deviations problem, there are special algorithms that take advantage of the special structure of the problem to speed up the basic linear programming simplex algorithm. Again, the interested reader is referred to Kennedy and Gentle (1980; Chapter 11) for a discussion of some of the properties of minimum L_∞ norm fits, as well as assumptions about probability distributions that result in desirable statistical properties for minimum L_∞ norm estimators.

6.7.1.3 L_p Norms and Iteratively Reweighted Least Squares

More general L_p norms may also be of interest. For $1 < p < 2$, if no element of $y - Xb$ is zero, we can formulate the p^{th} power of the norm as

$$\|y - Xb\|_p^p = (y - Xb)^{\mathrm{T}} W (y - Xb), \tag{6.51}$$

where

$$W = \text{diag}(|\, y_i - x_i^{\mathrm{T}} b|^{2-p}) \tag{6.52}$$

and x_i^{T} is the i^{th} row of X. The formulation (6.51) leads to the iteratively reweighted least squares (IRLS) algorithm, in which a sequence of weights $W^{(k)}$ and weighted least squares solutions $b^{(k)}$ are formed as in Algorithm 6.4.

Algorithm 6.4 Iteratively reweighted least squares

0. Input a threshold for a zero residual, ϵ_1 (which may be data dependent), and a large value for weighting zero residuals, w_{big}. Input stopping criteria, ϵ_2 and $k_{\max}$.
   Set $k = 0$. Choose an initial value $b^{(k)}$, perhaps as the OLS solution.
1. Compute the diagonal matrix $W^{(k)}$: $w_i^{(k)} = |y_i - x_i^{\mathrm{T}} b^{(k)}|^{2-p}$, except, if $|y_i - x_i^{\mathrm{T}} b^{(k)}| < \epsilon_1$, set $w_i^{(k)} = w_{\text{big}}$.
2. Compute $b^{(k+1)}$ by solving the weighted least squares problem: minimize $(y - Xb)^{\mathrm{T}} W^{(k)} (y - Xb)$.

3. If $\|b^{(k+1)} - b^{(k)}\| \leq \epsilon_2$, set $b = b^{(k+1)}$ and terminate.
4. If $k < k_{\max}$,
   set $k = k + 1$ and go to step 1;
   otherwise,
   issue message that
   "algorithm did not converge in $k_{\max}$ iterations". ∎

Compute $b^{(1)}$ by minimizing equation (6.51) with $W = W^{(0)}$; then compute $W^{(1)}$ from $b^{(1)}$, and iterate in this fashion. This method is easy to implement and will generally work fairly well, except for the problem of zero (or small) residuals. The most effective way of dealing with zero residuals is to set them to some large value.

Algorithm 6.4 will work for LAV fitting, although the algorithms based on linear programming alluded to above are better for this task. As mentioned above, LAV fits generally go through some observations; that is, they fit them exactly, yielding zero residuals. This means that in using Algorithm 6.4, the manner of dealing with zero residuals may become an important aspect of the efficiency of the algorithm.

6.7.2 Regularized Solutions

Overdetermined systems often arise because of a belief that some response y is linearly related to some other set of variables. This relation is expressed in the system

$$y \approx Xb.$$

The fact that $y \neq Xb$ for any b results because the relationship is not exact. There is perhaps some error in the measurements. It is also possible that there is some other variable not included in the columns of X. In addition, there may be some underlying randomness that could never be accounted for.

In any application in which we fit an overdetermined system, it is likely that the given values of X and y are only a sample (not necessarily a random sample) from some universe of interest. Whatever value of b provides the best fit (in terms of the criterion chosen) may not provide the best fit if some other equally valid values of X and y were used. The given dataset is fit optimally, but the underlying phenomenon of interest may not be modeled very well. The given dataset may suggest relationships among the variables that are not present in the larger universe of interest. Some element of the "true" b may be zero, but in the best fit for a given dataset, the value of that element may be significantly different from zero. Deciding on the evidence provided by a given dataset that there is a relationship among certain variables when indeed there is no relationship in the broader universe is an example of *overfitting*.

There are various approaches we may take to avoid overfitting, but there is no panacea. The problem is inherent in the process.

One approach to overfitting is *regularization*. In this technique, we restrain the values of b in some way. Minimizing $\|y - Xb\|$ may yield a b with large

elements, or values that are likely to vary widely from one dataset to another. One way of "regularizing" the solution is to minimize also some norm of b. The general formulation of the problem then is

$$\min_b(\|y - Xb\|_r + \lambda\|b\|_b), \tag{6.53}$$

where λ is some appropriately chosen nonnegative number. The norm on the residuals, $\|\cdot\|_r$, and that on the solution vector b, $\|\cdot\|_b$, are often chosen to be the same, and, of course, most often, they are chosen as the L_2 norm. If both norms are the L_2 norm, the fitting is called Tikhonov regularization. In statistical applications, this leads to "ridge regression". If $\|\cdot\|_r$ is the L_2 norm and $\|\cdot\|_b$ is the L_1 norm, the statistical method is called the "lasso". We discuss these formulations briefly in Sect. 9.5.4.

As an example, let us consider the data in equation (6.49) for the equation

$$y = b_0 + b_1 x.$$

We found the least squares solution to be $\hat{b}_0 = -0.3158$ and $\hat{b}_1 = 0.7895$, which fits the means and has a residual vector that is orthogonal to 1 and to x. Now let us regularize the least squares fit with an L_2 norm on b and with $\lambda = 5$. (The choice of λ depends on the scaling of the data and a number of other things we will not consider here. Typically, in an application, various values of λ are considered.) Again, we face the question of treating b_0 and b_1 differently. The regularization, which is a shrinkage, can be applied to both or just to b_1. Furthermore, we have the question of whether we want to force the equation to fit some data point exactly. In statistical applications, it is common not to apply the shrinkage to the intercept term and to force the fitted equation to fit the means exactly. Doing that, we get $\hat{b}_{1_\lambda} = 0.6857$, which is shrunken from the value of $\hat{b}_1$, and $\hat{b}_{0_\lambda} = 0.1200$, which is chosen so as to fit the mean. A plot of the data and the two fitted lines is shown in Fig. 6.3.

6.7.3 Minimizing Orthogonal Distances

In writing the equation $Xb = y + r$ in place of the overdetermined linear system $Xb \approx y$, we are allowing adjustments to y so as to get an equation. Another way of making an equation out of the overdetermined linear system $Xb \approx y$ is to write it as

$$(X + E)b = y + r; \tag{6.54}$$

that is, to allow adjustments to both X and y. Both X and E are in $\mathbb{R}^{n\times m}$ (and we assume $n > m$).

In fitting the linear model only with adjustments to y, we determine b so as to minimize some norm of r. Likewise, with adjustments to both X and y, we seek b so as to minimize some norm of the matrix E and the vector r. There are obviously several ways to approach this. We could take norms of

E and r separately and consider some weighted combination of the norms. Another way is to adjoin r to E and minimize some norm of the $n \times (m+1)$ matrix $[E|r]$.

A common approach is to minimize $\|[E|r]\|_{\mathrm{F}}$. This, of course, is the sum of squares of all elements in $[E|r]$. The method is therefore sometimes called "total least squares".

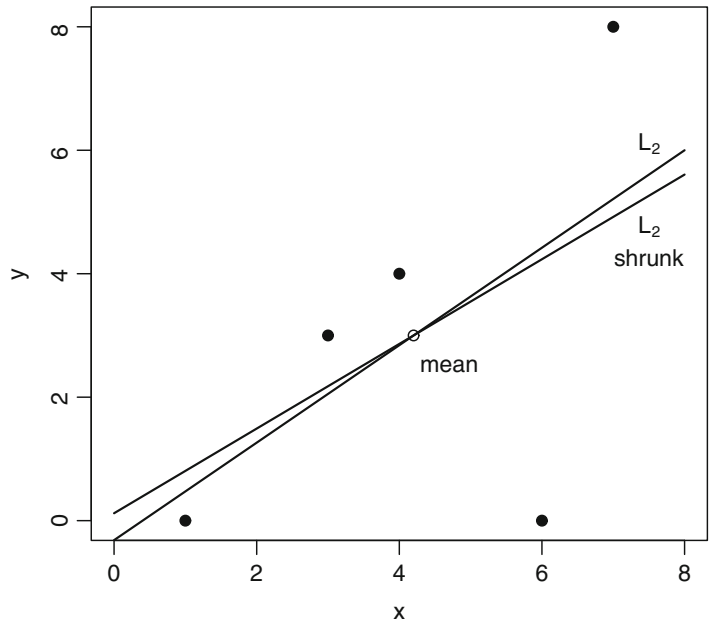


Figure 6.3. OLS and L_2 norm regularized minimum L_2 norm fits

If it exists, the minimum of $\|[E|r]\|_{\mathrm{F}}$ is achieved at

$$b = -v_{2*}/v_{22}, \tag{6.55}$$

where

$$[X|y] = UDV^{\mathrm{T}} \tag{6.56}$$

is the singular value decomposition (see equation (3.276) on page 161), and V is partitioned as

$$V = \begin{bmatrix} V_{11} & v_{*2} \\ v_{2*} & v_{22} \end{bmatrix}.$$

If E has some special structure, the problem of minimizing the orthogonal residuals may not have a solution. Golub and Van Loan (1980) show that a sufficient condition for a solution to exist is that $d_m > d_{m+1}$. (Recall that the

ds in the SVD are nonnegative and they are indexed so as to be nonincreasing. If $d_m = d_{m+1}$, a solution may or may not exist.)

Again, as an example, let us consider the data in equation (6.49) for the equation

$$y = b_0 + b_1 x.$$

We found the least squares solution to be $\hat{b}_0 = -0.3158$ and $\hat{b}_1 = 0.7895$, which fits the mean and has a residual vector that is orthogonal to 1 and to x. Now we determine a fit so that the L_2 norm of the orthogonal residuals is minimized. Again, we will force the equation to fit the mean exactly. We get $\hat{b}_{0_{\text{orth}}} = -4.347$ and $\hat{b}_{0_{\text{orth}}} = 1.749$. A plot of the data, the two fitted lines, and the residuals is shown in Fig. 6.4.

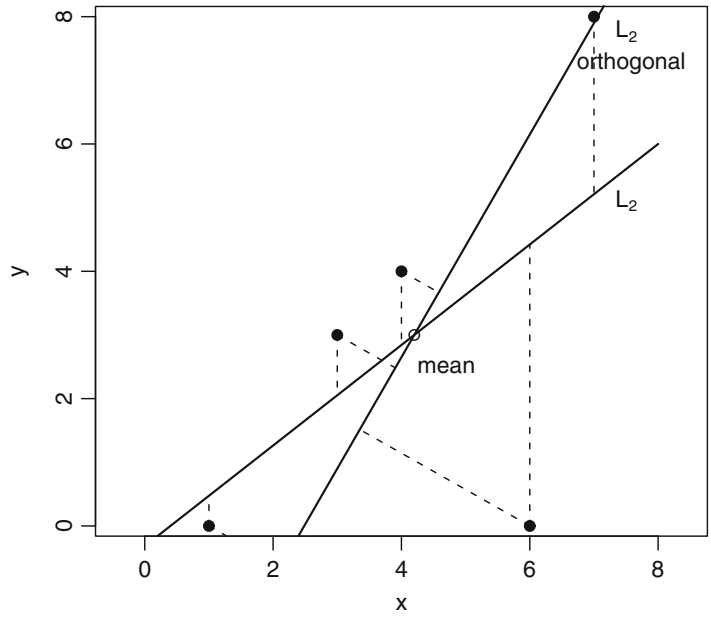


Figure 6.4. OLS and minimum orthogonal L_2-norm fits

The orthogonal residuals can be weighted in the usual way by premultiplication by a Cholesky factor of a weight matrix, as discussed on page 295.

If some norm other than the L_2 norm is to be minimized, an iterative approach must be used. Ammann and Van Ness (1988) describe an iterative method that is applicable to any norm, so long as a method is available to compute a value of b that minimizes the norm of the usual vertical distances in a model such as equation (9.11). The method is simple. We first fit $y = Xb$, minimizing the vertical distances in the usual way; we then rotate y into $\tilde{y}$ and X into $\widetilde{X}$, so that the fitted plane is horizontal. Next, we fit $\tilde{y} = \widetilde{X}b$ and repeat. After continuing this way until the fits in the rotated spaces do not

change from step to step, we adjust the fitted b back to the original unrotated space. Because of these rotations, if we assume that the model fits some point exactly, we must adjust y and X accordingly (see the discussion on page 289). In the following, we assume that the model fits the means exactly, so we center the data. We let m be the number of columns in the centered data matrix. (The centered matrix does not contain a column of 1s. If the formulation of the model $y = Xb$ includes an intercept term, then X is $n \times (m+1)$.)

Algorithm 6.5 Iterative orthogonal residual fitting through the means

0. Input stopping criteria, ϵ and $k_{\max}$.
   Set $k = 1$, $y_c^{(0)} = y_c$, $X_c^{(0)} = X_c$, and $D^{(0)} = I_{m+1}$.
1. Determine a value $b_c^{(k)}$ that minimizes the norm of $\left(y_c^{(k-1)} - X_c^{(k-1)} b_c^{(k)}\right)$.
2. If $b_c^{(k)} \leq \epsilon$, go to step 7.
3. Determine a rotation matrix $Q^{(k)}$ that makes the k^{th} fit horizontal.
4. Transform the matrix $\left[y_c^{(k-1)} | X_c^{(k-1)}\right]$ $\left[y_c^{(k)} | X_c^{(k)}\right]$ by a rotation matrix:

$$\left[y_c^{(k)} | X_c^{(k)}\right] = \left[y_c^{(k-1)} | X_c^{(k-1)}\right] Q^{(k)}.$$

5. Transform $D^{(k-1)}$ by the same rotation: $D^{(k)} = D^{(k-1)} Q^{(k)}$.
6. If $k < k_{\max}$,
   set $k = k + 1$ and go to step 1;
   otherwise,
   issue message that
   "algorithm did not converge in $k_{\max}$ iterations".
7. For $j = 2, \ldots, m$, choose $b_j = d_{j,m+1}/d_{m+1,m+1}$ (So long as the rotations have not produced a vertical plane in the unrotated space, $d_{m+1,m+1}$ will not be zero.)
8. Compute $b_1 = \bar{y} - \sum_{j=2}^{k} b_j * \bar{x}_j$ (where $\bar{x}_j$ is the mean of the j^{th} column of the original uncentered X). ∎

An appropriate rotation matrix for Algorithm 6.5 is Q in the QR decomposition of

$$\begin{bmatrix} I_m & 0 \\ (b^{(k)})^{\mathrm{T}} & 1 \end{bmatrix}.$$

Note that forcing the fit to go through the means, as we do in Algorithm 6.5, is not usually done for norms other than the L_2 norm (see Fig. 6.2).

Exercises

6.1. Let A be nonsingular, and let $\kappa(A) = \|A\| \, \|A^{-1}\|$.

a) Prove equation (6.10):

$$\kappa_2(A) = \frac{\max_{x \neq 0} \frac{\|Ax\|}{\|x\|}}{\min_{x \neq 0} \frac{\|Ax\|}{\|x\|}}.$$

b) Using the relationship above, explain heuristically why $\kappa(A)$ is called the "condition number" of A.

6.2. Consider the system of linear equations

$$\begin{aligned} x_1 + 4x_2 + \ \ x_3 &= 12, \\ 2x_1 + 5x_2 + 3x_3 &= 19, \\ x_1 + 2x_2 + 2x_3 &= \ \ 9. \end{aligned} \tag{6.57}$$

a) Solve the system using Gaussian elimination with partial pivoting.

b) Solve the system using Gaussian elimination with complete pivoting.

c) Determine the D, L, and U matrices of the Gauss-Seidel method (equation (6.19), page 280) and determine the spectral radius of

$$(D + L)^{-1}U.$$

d) Do three steps of the Gauss-Seidel method using the original formulation (6.57), and starting with $x^{(0)} = (1, 1, 1)$, and evaluate the L_2 norm of the difference of two successive approximate solutions.

e) Now do one partial pivot on the system (6.57) to form the coefficient matrix

$$\widetilde{A} = \begin{bmatrix} 2 & 5 & 3 \\ 1 & 4 & 1 \\ 1 & 2 & 2 \end{bmatrix}.$$

Write the corresponding $\widetilde{D}$, $\widetilde{L}$, and $\widetilde{U}$, and determine $\rho((\widetilde{D}+\widetilde{L})^{-1}\widetilde{U})$.

f) Do three steps of the Gauss-Seidel method using the one-pivot formulation of Exercise 6.2e, and starting with $x^{(0)} = (1, 1, 1)$, and evaluate the L_2 norm of the difference of two successive approximate solutions.

g) Do three steps of the Gauss-Seidel method with successive overrelaxation using the one-pivot formulation of Exercise 6.2e and with $\omega = 0.1$, starting with $x^{(0)} = (1, 1, 1)$, and evaluate the L_2 norm of the difference of two successive approximate solutions.

h) Do three steps of the conjugate gradient method using the original formulation(6.57), starting with $x^{(0)} = (1, 1, 1)$, and evaluate the L_2 norm of the difference of two successive approximate solutions.

6.3. The normal equations.
   a) For any matrix X with real elements, show that $X^{\mathrm{T}}X$ is nonnegative definite.
   b) For any $n \times m$ matrix X with real elements and with $n < m$, show that $X^{\mathrm{T}}X$ is not positive definite.
   c) Let X be an $n \times m$ matrix of full column rank. Show that $X^{\mathrm{T}}X$ is positive definite.

6.4. Solving an overdetermined system $Xb = y$, where X is $n \times m$.
   a) Count how many multiplications and additions are required to form $X^{\mathrm{T}}X$. (A multiplication or addition such as this is performed in floating point on a computer, so the operation is called a "flop". Sometimes a flop is considered a combined operation of multiplication and addition; at other times, each is considered a separate flop. See page 507. The distinction is not important here; just count the total number.)
   b) Count how many flops are required to form $X^{\mathrm{T}}y$.
   c) Count how many flops are required to solve $X^{\mathrm{T}}Xb = X^{\mathrm{T}}y$ using a Cholesky decomposition.
   d) Count how many flops are required to form a QR decomposition of X using reflectors.
   e) Count how many flops are required to form a $Q^{\mathrm{T}}y$.
   f) Count how many flops are required to solve $R_1 b = c_1$ (equation (6.41), page 293).
   g) If n is large relative to m, what is the ratio of the total number of flops required to form and solve the normal equations using the Cholesky method to the total number required to solve the system using a QR decomposition? Why is the QR method generally preferred?

6.5. On page 293, we derived the least squares solution, $\widehat{b} = X^{+}y$, to the overdetermined system $Xb = y$ by use of the QR decomposition of X. This equation for the least squares solution can also be shown to be correct in other ways.
   a) For simplicity in this part, let us assume that X is of full rank; that is, we will write $\widehat{b}$ as the solution to the normal equations, $\widehat{b} = (X^{\mathrm{T}}X)^{-1}X^{\mathrm{T}}y$. (This assumption is not necessary, but the proof without it is much messier.)
   Show that $\widehat{b} = X^{+}y$ by showing that $(X^{\mathrm{T}}X)^{-1}X^{\mathrm{T}} = X^{+}$.
   b) On page 291 we showed that orthogonality of the residuals characterized a least squares solution. Show that $\widehat{b} = X^{+}y$ is a least squares solution by showing that in this case $X^{\mathrm{T}}(y - X\widehat{b}) = 0$.

6.6. Verify equation (6.51).

7

Evaluation of Eigenvalues and Eigenvectors

Before we discuss methods for computing eigenvalues, we recall a remark made in Chap. 5. A given n^{th}-degree polynomial $p(c)$ is the characteristic polynomial of some matrix. The companion matrix of equation (3.225) is one such matrix. Thus, given a general polynomial p, we can form a matrix A whose eigenvalues are the roots of the polynomial; and likewise, given a square matrix, we can write a polynomial in its eigenvalues. It is a well-known fact in the theory of equations that there is no general formula for the roots of a polynomial of degree greater than 4. This means that we cannot expect to have a direct method for calculating eigenvalues of any given matrix.

The eigenvalues of some matrices, of course, can be evaluated directly. The eigenvalues of a diagonal matrix, for example, are merely the diagonal elements. In that case, the characteristic polynomial is of the factored form $\prod(a_{ii} - c)$, whose roots are immediately obtainable. For general eigenvalue computations, however, we must use an iterative method.

In statistical applications, the matrices whose eigenvalues are of interest are often symmetric. Symmetric matrices are diagonalizable and have only real eigenvalues. (As usual, we will assume the matrices themselves are real.) The problem of determining the eigenvalues of a symmetric matrix therefore is simpler than the corresponding problem for a general matrix. In many statistical applications, the symmetric matrices of interest are nonnegative definite, and this can allow use of simpler methods for computing eigenvalues and eigenvectors. In addition, nonsymmetric matrices of interest in statistical applications are often irreducible nonnegative matrices, and computations for eigenvalues and eigenvectors for matrices of this type are also often simpler. (We will discuss such matrices in Sect. 8.7.3.)

In this chapter, we describe various methods for computing eigenvalues. A given method may have some desirable property for particular applications, and in some cases, the methods may be used in combination. Some of the methods rely on sequences that converge to a particular eigenvalue or eigenvector. The power method, discussed in Sect. 7.2, is of this type; one

J.E. Gentle, *Matrix Algebra*, Springer Texts in Statistics,
DOI 10.1007/978-3-319-64867-5_7

eigenpair at a time is computed. Other methods are based on sequences of orthogonally similar matrices that converge to a diagonal matrix. An example of such a method is called the *LR method*. This method, which we will not consider in detail, is based on a factorization of A into left and right factors, F_{L} and F_{R}, and the fact that if c is an eigenvalue of $F_{\mathrm{L}}F_{\mathrm{R}}$, then it is also an eigenvalue of $F_{\mathrm{R}}F_{\mathrm{L}}$ (property 8, page 136). If $A = L^{(0)}U^{(0)}$ is an LU decomposition of A with 1s on the diagonal of either $L^{(0)}$ or $U^{(0)}$, iterations of LU decompositions of the similar matrices

$$L^{(k+1)}U^{(k+1)} = U^{(k)}L^{(k)},$$

under some conditions, will converge to a similar diagonal matrix. The sufficient conditions for convergence include nonnegative definiteness.

7.1 General Computational Methods

For whatever approach is taken for finding eigenpairs, there are some general methods that may speed up the process or that may help in achieving higher numerical accuracy. Before describing some of the techniques, we consider a bound on the sensitivity of eigenvalues to perturbations of the matrix.

7.1.1 Numerical Condition of an Eigenvalue Problem

The upper bounds on the largest eigenvalue, given in inequalities (3.235) and (3.236) on page 142, provide a simple indication of the region in the complex plane in which the eigenvalues lie.

The Gershgorin disks (inequalities (3.239) and (3.240), page 145) provide additional information about the regions of the complex plane in which the eigenvalues lie. The Gershgorin disks can be extended to define separate regions that contain eigenvalues, but we will not consider those refinements here. The spectral radius and/or Gershgorin disks can be used to obtain approximate values to use in some iterative approximation methods; see equation (7.13) below, for example.

In any computational problem, it is of interest to know what is the effect on the solution when there are small changes in the problem itself. This leads to the concept of a condition number, as we discussed in Sect. 6.1.1 beginning on page 267. The objective is to quantify or at least determine bounds on the rate of change in the "output" relative to changes in the "input".

In the eigenvalue problem, we begin with a square matrix A. We assume that A is diagonalizable. (All symmetric matrices are diagonalizable, and equation (3.248) on page 149 gives necessary and sufficient conditions which many other matrices encountered in statistical applications also satisfy.) We form $V^{-1}AV = C = \mathrm{diag}((c_1, \ldots, c_n))$, where the c_i are the eigenvalues of A.

The approach, as in Sect. 6.1.1, is to perturb the problem slightly by adding a small amount δA to A. Let $\widetilde{A} = A + \delta A$. (Notice that δA does not necessarily represent a scalar multiple of the matrix.)

If A is well-conditioned for the eigenvalue problem, then if $\|\delta A\|$ is small relative to $\|A\|$, the differences in the eigenvalues of A and of $\widetilde{A}$ are likewise small. Let d be any eigenvalue of $\widetilde{A}$ that is not an eigenvalue of A. (If all eigenvalues of $\widetilde{A}$ are eigenvalues of A, then the perturbation has had no effect, and the question we are addressing is not of interest.) Our interest will be in

$$\min_{c \in \sigma(A)} |c - d|.$$

If d is an eigenvalue of $\widetilde{A}$, then $A + \delta A - dI$ is singular and so $V^{-1}(A + \delta A - dI)V$ is also singular. Simplifying this latter expression, we have that $C - dI + V^{-1}\delta AV$ is singular. Since d is not an eigenvalue of A, however, $C - dI$ must be nonsingular, and so $(C - dI)^{-1}$ exists. Multiplying the two expressions we have that $I + (C - dI)^{-1}V^{-1}\delta AV$ is also singular; hence -1 is an eigenvalue of $(C - dI)^{-1}V^{-1}\delta AV$, and so by property 16 on page 140, we have

$$1 \leq \|(C - dI)^{-1}V^{-1}\delta AV\|,$$

for any consistent norm. (Recall that all matrix norms are consistent in my definition.) Furthermore, again using the consistency property multiple times,

$$\|(C - dI)^{-1}V^{-1}\delta AV\| \leq \|(C - dI)^{-1}\| \, \|V^{-1}\| \, \|\delta A\| \, \|V\|.$$

In equation (6.7) on page 269, we defined "the" condition number for a nonsingular matrix V as $\kappa(V) = \|V\| \, \|V^{-1}\|$. Now, since $C - dI$ is a diagonal matrix, we can rewrite the two inequalities above as

$$\min_{c \in \sigma(A)} |c - d| \leq \kappa(V)\|\delta A\|; \tag{7.1}$$

that is, the eigenvalues of the perturbed matrix are within given bounds from the eigenvalues of the original matrix.

This fact is called the Bauer-Fike theorem, and it has several variations and ramifications. It is closely related to Gershgorin disks. Our interest here is just to provide a perturbation bound that conveniently relates to the condition number of the diagonalizing matrix.

If A is symmetric, it is orthogonally diagonalizable, and the V above is an orthogonal matrix. Hence, if A is a symmetric matrix, $\widetilde{A} = A + \delta A$, and d is an eigenvalue of $\widetilde{A} = A + \delta A$, then

$$\min_{c \in \sigma(A)} |c - d| \leq \|\delta A\|.$$

7.1.2 Eigenvalues from Eigenvectors and Vice Versa

Some methods for eigenanalysis yield the eigenvalues, and other methods yield the eigenvectors. Given one member of an eigenpair, we usually want to find the other member.

If we are given an eigenvector v of the matrix A, there must be some element v_j that is not zero. For any nonzero element of the eigenvector, the eigenvalue corresponding to v is

$$(Av)_j/v_j. \tag{7.2}$$

Likewise, if the eigenvalue c is known, a corresponding eigenvector is any solution to the singular system

$$(A - cI)v = 0. \tag{7.3}$$

(It is relevant to note that the system is singular because many standard software packages will refuse to solve singular systems whether or not they are consistent!)

An eigenvector associated with the eigenvalue c can be found using equation (7.3) if we know the position of any nonzero element in the vector. Suppose, for example, it is known that $v_1 \neq 0$. We can set $v_1 = 1$ and form another system to solve for the remaining elements of v by writing

$$\begin{bmatrix} a_{11} - 1 & a_1^{\mathrm{T}} \\ a_2 & A_{22} - cI_{n-1} \end{bmatrix} \begin{bmatrix} 1 \\ v_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \tag{7.4}$$

where v_2 is an $(n-1)$-vector and a_1^{T} and a_2 are the remaining elements in the first row and first column, respectively, of A. Rearranging this, we get the $(n-1) \times (n-1)$ system

$$(A_{22} - cI_{n-1})v_2 = -a_2. \tag{7.5}$$

The locations of any zero elements in the eigenvector are critical for using this method. To form a system as in equation (7.4), the position of some nonzero element must be known. Another problem in using this method arises when the geometric multiplicity of the eigenvalue is greater than 1. In that case, the system in equation (7.5) is also singular, and the process must be repeated to form an $(n-2)\times(n-2)$ system. If the multiplicity of the eigenvalue is k, the first full rank system encountered while continuing in this way is the one that is $(n-k) \times (n-k)$.

7.1.3 Deflation

Whenever an eigenvalue together with its associated left and right eigenvectors for a real matrix A are available, another matrix can be formed for which all the other nonzero eigenvalues and corresponding eigenvectors are the same

as for A. (Of course the left and right eigenvalues for many matrices are the same.)

Suppose c_i is an eigenvalue of A with associated right and left eigenvectors v_i and w_i, respectively. Now, suppose that c_j is a nonzero eigenvalue of A such that $c_j \neq c_i$. Let v_j and w_j be, respectively, right and left eigenvectors associated with c_j. Now,

$$\langle Av_i, w_j\rangle = \langle c_i v_i, w_j\rangle = c_i\langle v_i, w_j\rangle,$$

but also

$$\langle Av_i, w_j\rangle = \langle v_i, A^{\mathrm{T}} w_j\rangle = \langle v_i, c_j w_j\rangle = c_j\langle v_i, w_j\rangle.$$

But if

$$c_i\langle v_i, w_j\rangle = c_j\langle v_i, w_j\rangle$$

and $c_j \neq c_i$, then $\langle v_i, w_j\rangle = 0$. Consider the matrix

$$B = A - c_i v_i w_i^{\mathrm{H}}. \tag{7.6}$$

We see that

$$\begin{aligned} Bw_j &= Aw_j - c_i v_i w_i^{\mathrm{H}} w_j \\ &= Aw_j \\ &= c_j w_j, \end{aligned}$$

so c_j and w_j are, respectively, an eigenvalue and an eigenvector of B.

The matrix B has some of the flavor of the sum of some terms in a spectral decomposition of A. (Recall that the spectral decomposition is guaranteed to exist only for matrices with certain properties. In Chap. 3, we stated the existence for diagonalizable matrices but derived it only for symmetric matrices.)

The ideas above lead to a useful method for finding eigenpairs of a diagonalizable matrix. (The method also works if we begin with a simple eigenvalue.) We will show the details only for a real symmetric matrix.

7.1.3.1 Deflation of Symmetric Matrices

Let A be an $n \times n$ symmetric matrix. A therefore is diagonalizable, its eigenvalues and eigenvectors are real, and the left and right eigenvalues are the same.

Let (c, v), with $v^{\mathrm{T}}v = 1$, be an eigenpair of A. Now let X be an $n \times n-1$ matrix whose columns form an orthogonal basis for $\mathcal{V}(A - vv^{\mathrm{T}})$. One easy way of doing this is to choose $n-1$ of the n unit vectors of order n such that none are equal to v and then, beginning with v, use Gram-Schmidt transformations to orthogonalize the vectors, using Algorithm 2.1 on page 39. (Assuming v is not a unit vector, we merely choose $e_1, \ldots, e_{n-1}$ together with v as the starting set of linearly independent vectors.) Now let $P = [v|X]$. We have

$$P^{-1} = \begin{bmatrix} v^{\mathrm{T}} \\ X^{\mathrm{T}}(I - vv^{\mathrm{T}}) \end{bmatrix},$$

as we see by direct multiplication, and

$$P^{-1}AP = \begin{bmatrix} c & 0 \\ 0 & B \end{bmatrix}, \tag{7.7}$$

where B is the $(n-1) \times (n-1)$ matrix $X^{\mathrm{T}}AX$.

Clearly, B is symmetric and the eigenvalues of B are the same as the other $n-1$ eigenvalues of A. The important point is that B is $(n-1) \times (n-1)$.

7.1.4 Preconditioning

The convergence of iterative methods applied to a linear system $Ax = b$ can often be speeded up by replacing the system by an equivalent system $M^{-1}Ax = M^{-1}b$. The iterations then depend on the properties, such as the relative magnitudes of the eigenvalues, of $M^{-1}A$ rather than A. The replacement of the system $Ax = b$ by $M^{-1}Ax = M^{-1}b$ is called *preconditioning*. (It is also sometimes called *left preconditioning*, and the use of the system $AM^{-1}y = b$ with $y = Mx$ is called *right preconditioning*. Either or both kinds of preconditioning may be used in a given iterative algorithm.) The matrix M is called a *preconditioner*.

Determining an effective preconditioner matrix M^{-1} for eigenvalue computations is not straightforward. In general, the objective would be to determine $M^{-1}A$ so that it is "close" to I, because then the eigenvalues might be easier to obtain by whatever method we may use. The salient properties of I are that it is normal (see Sect. 8.2.3 beginning on page 345) and its eigenvalues are clustered.

There are various kinds of preconditioning. We have considered preconditioning in the context of an iterative algorithm for solving linear systems on page 284. Some preconditioning methods work better as an adjunct to one algorithm, and others work better in conjunction with some other algorithm. Obviously, the efficacy depends on the nature of the data input to the problem. In the case of a sparse matrix A, for example an incomplete factorization $A \approx \widetilde{L}\widetilde{U}$ where both $\widetilde{L}$ and $\widetilde{U}$ are sparse, $M = \widetilde{L}\widetilde{U}$ may be a good preconditioner. We will not consider any of the details here. Benzi (2002) provides a good survey of techniques, but it is difficult to identify general methods that work well.

7.1.5 Shifting

If c is an eigenvalue of A, then $c - d$ is an eigenvalue of $A - dI$, and the associated eigenvectors are the same. (This is property 7 on page 136.) Hence, instead of seeking an eigenvalue of A, we might compute (or approximate) an

eigenvalue of $A - dI$. (We recall also, from equation (6.11) on page 272, that, for appropriate signs of d and the eigenvalues, the condition number of $A - dI$ is better than the condition number of A.)

Use of $A - dI$ amounts to a "shift" in the eigenvalue. This can often improve the convergence rate in an algorithm to compute an eigenvalue. (Remember that all general algorithms to compute eigenvalues are iterative.)

The best value of d in the shift depends on both the algorithm and the characteristics of the matrix. Various shifts have been suggested. One common value of the shift is based on the Rayleigh quotient shift; another common value is called the "Wilkinson shift", after James Wilkinson. We will not discuss any of the particular shift values here.

7.2 Power Method

The power method is a straightforward method that can be used for a real diagonalizable matrix with a simple dominant eigenvalue. A symmetric matrix is diagonalizable, of course, but it may not have a simple dominant eigenvalue.

The power method finds the dominant eigenvalue. In some applications, only the dominant eigenvalue is of interest. If other eigenvalues are needed, however, we can find them one at a time by deflation.

Let A be a real $n \times n$ diagonalizable matrix with a simple dominant eigenvalue. Index the eigenvalues c_i so that $|c_1| > |c_2| \geq \cdots |c_n|$, with corresponding normalized eigenvectors v_i. Note that the requirement for the dominant eigenvalue that $c_1 > c_2$ implies that c_1 and the dominant eigenvector v_1 are unique and that c_1 is real (because otherwise $\bar{c}_1$ would also be an eigenvalue, and that would violate the requirement).

Now let x be an n-vector that is not orthogonal to v_1. Because A is assumed to be diagonalizable, the eigenvectors are linearly independent and so x can be represented as a linear combination of the eigenvectors,

$$x = b_1 v_1 + \cdots + b_n v_n. \tag{7.8}$$

Because x is not orthogonal to v_1, $b_1 \neq 0$. The power method is based on a sequence

$$x,\ Ax,\ A^2x, \ldots.$$

(This sequence is a finite Krylov space generating set; see equation (6.26).) From the relationships above and the definition of eigenvalues and eigenvectors, we have

$$\begin{aligned} Ax &= b_1 Av_1 + \cdots + b_n Av_n \\ &= b_1 c_1 v_1 + \cdots + b_n c_n v_n \\ A^2x &= b_1 c_1^2 v_1 + \cdots + b_n c_n^2 v_n \\ \cdots &= \cdots \end{aligned}$$

$$\begin{aligned} A^j x &= b_1 c_1^j v_1 + \cdots + b_n c_n^j v_n \\ &= c_1^j \left(b_1 v_1 + \cdots + b_n \left(\frac{c_n}{c_1} \right)^j v_n \right). \end{aligned} \tag{7.9}$$

To simplify the notation, let

$$u^{(j)} = A^j x / c_1^j \tag{7.10}$$

(or, equivalently, $u^{(j)} = Au^{(j-1)}/c_1$). From equations (7.9) and the fact that $|c_1| > |c_i|$ for $i > 1$, we see that $u^{(j)} \to b_1 v_1$, which is the nonnormalized dominant eigenvector.

We have the bound

$$\begin{aligned} \left\| u^{(j)} - b_1 v_1 \right\| &= \left\| b_2 \left(\frac{c_2}{c_1} \right)^j v_2 + \cdots \right. \\ &\qquad \left. \cdots + b_n \left(\frac{c_n}{c_1} \right)^j v_n \right\| \\ &\le |b_2| \left| \frac{c_2}{c_1} \right|^j \|v_2\| + \cdots \\ &\qquad \cdots + |b_n| \left| \frac{c_n}{c_1} \right|^j \|v_n\| \\ &\le (|b_2| + \cdots + |b_n|) \left| \frac{c_2}{c_1} \right|^j. \end{aligned} \tag{7.11}$$

The last expression results from the fact that $|c_2| \ge |c_i|$ for $i > 2$ and that the v_i are unit vectors.

From equation (7.11), we see that the norm of the difference of $u^{(j)}$ and $b_1 v_1$ decreases by a factor of approximately $|c_2/c_1|$ with each iteration; hence, this ratio is an important indicator of the rate of convergence of $u^{(j)}$ to the dominant eigenvector.

If $|c_1| > |c_2| > |c_3|$, $b_2 \neq 0$, and $b_1 \neq 0$, the power method converges linearly (see page 511); that is,

$$0 < \lim_{j \to \infty} \frac{\|u^{(j+1)} - b_1 v_1\|}{\|u^{(j)} - b_1 v_1\|} < 1 \tag{7.12}$$

(see Exercise 7.1c, page 324). Shifting the matrix to form $A - dI$ results in a matrix with eigenvalues with different relative sizes, and may be useful in speeding up the convergence.

If an approximate value of the eigenvector v_1 is available and x is taken to be that approximate value, the convergence will be faster. If an approximate value of the dominant eigenvalue, $\widehat{c}_1$, is available, starting with any $y^{(0)}$, a few iterations on

$$(A - \widehat{c}_1 I)y^{(k)} = y^{(k-1)} \tag{7.13}$$

may yield a better starting value for x. Once the eigenvector associated with the dominant eigenvalue is determined, the eigenvalue c_1 can easily be determined, as described above.

7.2.1 Inverse Power Method

If A is nonsingular, we can also use the power method on A^{-1} to determine the smallest eigenvalue of A. This is called the "inverse power method".

The rate of convergence may be very different from that of the power method applied to A. Shifting is also generally important in the inverse power method. Of course this method only determines the eigenvalue with the smallest absolute value. If other eigenvalues are needed, we can find them one at a time by deflation.

7.3 Jacobi Method

The Jacobi method for determining the eigenvalues of a simple symmetric matrix A uses a sequence of orthogonal similarity transformations that eventually results in the transformation

$$A = PCP^{-1}$$

(see equation (3.247) on page 149) or

$$C = P^{-1}AP,$$

where C is diagonal. Recall that similar matrices have the same eigenvalues.

The matrices for the similarity transforms are the Givens rotation or Jacobi rotation matrices discussed on page 238. The general form of one of these orthogonal matrices, $G_{pq}(\theta)$, given in equation (5.12) on page 239, is the identity matrix with $\cos\theta$ in the $(p,p)^{\text{th}}$ and $(q,q)^{\text{th}}$ positions, $\sin\theta$ in the $(p,q)^{\text{th}}$ position, and $-\sin\theta$ in the $(q,p)^{\text{th}}$ position:

$$G_{pq}(\theta) = \begin{array}{c} \\ p \\ \\ q \\ \\ \end{array} \begin{array}{c} \begin{array}{ccccc} p & & & q & \end{array} \\ \begin{bmatrix} I & 0 & 0 & 0 & 0 \\ 0 & \cos\theta & 0 & \sin\theta & 0 \\ 0 & 0 & I & 0 & 0 \\ 0 & -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 0 & I \end{bmatrix} \end{array}.$$

The Jacobi iteration is

$$A^{(k)} = G^{\mathrm{T}}_{p_k q_k}(\theta_k) A^{(k-1)} G_{p_k q_k}(\theta_k),$$

where p_k, q_k, and θ_k are chosen so that the $A^{(k)}$ is "more diagonal" than $A^{(k-1)}$. Specifically, the iterations will be chosen so as to reduce the sum of the squares of the off-diagonal elements, which for any square matrix A is

$$\|A\|_{\mathrm{F}}^2 - \sum_i a_{ii}^2.$$

The orthogonal similarity transformations preserve the Frobenius norm

$$\left\|A^{(k)}\right\|_{\mathrm{F}} = \left\|A^{(k-1)}\right\|_{\mathrm{F}}.$$

Because the rotation matrices change only the elements in the $(p,p)^{\text{th}}$, $(q,q)^{\text{th}}$, and $(p,q)^{\text{th}}$ positions (and also the $(q,p)^{\text{th}}$ position since both matrices are symmetric), we have

$$\left(a_{pp}^{(k)}\right)^2 + \left(a_{qq}^{(k)}\right)^2 + 2\left(a_{pq}^{(k)}\right)^2 = \left(a_{pp}^{(k-1)}\right)^2 + \left(a_{qq}^{(k-1)}\right)^2 + 2\left(a_{pq}^{(k-1)}\right)^2.$$

The off-diagonal sum of squares at the k^{th} stage in terms of that at the $(k-1)^{\text{th}}$ stage is

$$\begin{aligned}\left\|A^{(k)}\right\|_{\mathrm{F}}^2 - \sum_i \left(a_{ii}^{(k)}\right)^2 &= \left\|A^{(k)}\right\|_{\mathrm{F}}^2 - \sum_{i\neq p,q} \left(a_{ii}^{(k)}\right)^2 - \left(\left(a_{pp}^{(k)}\right)^2 + \left(a_{qq}^{(k)}\right)^2\right) \\ &= \left\|A^{(k-1)}\right\|_{\mathrm{F}}^2 - \sum_i \left(a_{ii}^{(k-1)}\right)^2 - 2\left(a_{pq}^{(k-1)}\right)^2 + 2\left(a_{pq}^{(k)}\right)^2.\end{aligned} \tag{7.14}$$

Hence, for a given index pair, (p,q), at the k^{th} iteration, the sum of the squares of the off-diagonal elements is minimized by choosing the rotation matrix so that

$$a_{pq}^{(k)} = 0. \tag{7.15}$$

As we saw on page 239, it is easy to determine the angle θ so as to introduce a zero in a single Givens rotation. Here, we are using the rotations in a similarity transformation, so it is a little more complicated.

The requirement that $a_{pq}^{(k)} = 0$ implies

$$a_{pq}^{(k-1)}\left(\cos^2\theta - \sin^2\theta\right) + \left(a_{pp}^{(k-1)} - a_{qq}^{(k-1)}\right)\cos\theta\sin\theta = 0. \tag{7.16}$$

Using the trigonometric identities

$$\begin{aligned}\cos(2\theta) &= \cos^2\theta - \sin^2\theta \\ \sin(2\theta) &= 2\cos\theta\sin\theta,\end{aligned}$$

in equation (7.16), we have

$$\tan(2\theta) = \frac{2a_{pq}^{(k-1)}}{a_{pp}^{(k-1)} - a_{qq}^{(k-1)}},$$

which yields a unique angle in $[-\pi/4,\ \pi/4]$. Of course, the quantities we need are $\cos\theta$ and $\sin\theta$, not the angle itself. First, using the identity

$$\tan\theta = \frac{\tan(2\theta)}{1 + \sqrt{1 + \tan^2(2\theta)}},$$

we get $\tan\theta$ from $\tan(2\theta)$; and then from $\tan\theta$ we can compute the quantities required for the rotation matrix $G_{pq}(\theta)$:

$$\begin{aligned} \cos\theta &= \frac{1}{\sqrt{1 + \tan^2\theta}}, \\ \sin\theta &= \cos\theta \tan\theta. \end{aligned}$$

Convergence occurs when the off-diagonal elements are sufficiently small. The quantity (7.14) using the Frobenius norm is the usual value to compare with a convergence criterion, ϵ.

From equation (7.15), we see that the best index pair, (p, q), is such that

$$\left|a_{pq}^{(k-1)}\right| = \max_{i<j} \left|a_{ij}^{(k-1)}\right|.$$

If this choice is made, the Jacobi method can be shown to converge (see Watkins 2002). The method with this choice is called the *classical Jacobi* method.

For an $n \times n$ matrix, the number of operations to identify the maximum off-diagonal is $O(n^2)$. The computations for the similarity transform itself are only $O(n)$ because of the sparsity of the rotators. Of course, the computations for the similarity transformations are more involved than those to identify the maximum off-diagonal, so, for small n, the classical Jacobi method should be used. If n is large, however, it may be better not to spend time looking for the maximum off-diagonal. Various *cyclic Jacobi* methods have been proposed in which the pairs (p, q) are chosen systematically without regard to the magnitude of the off-diagonal being zeroed. Depending on the nature of the cyclic Jacobi method, it may or may not be guaranteed to converge. For certain schemes, quadratic convergence has been proven; for at least one other scheme, an example showing failure of convergence has been given. See Watkins (2002) for a discussion of the convergence issues.

The Jacobi method is one of the oldest algorithms for computing eigenvalues, and has recently become important again because it lends itself to easy implementation on parallel processors (see Zhou and Brent 2003).

Notice that at the k^{th} iteration, only two rows and two columns of $A^{(k)}$ are modified. This is what allows the Jacobi method to be performed in parallel.

We can form $\lfloor n/2 \rfloor$ pairs and do $\lfloor n/2 \rfloor$ rotations simultaneously. Thus, each parallel iteration consists of a choice of a set of index pairs and then a batch of rotations. Although, as we have indicated, the convergence may depend on which rows are chosen for the rotations, if we are to achieve much efficiency by performing the operations in parallel, we cannot spend much time in deciding how to form the pairs for the rotations. Various schemes have been suggested for forming the pairs for a parallel iteration. A simple scheme, called "mobile Jacobi" (see Watkins 2002), is:

1. Perform $\lfloor n/2 \rfloor$ rotations using the pairs
$$(1,2),\ (3,4),\ (5,6),\ldots.$$
2. Interchange all rows and columns that were rotated.
3. Perform $\lfloor (n-1)/2 \rfloor$ rotations using the pairs
$$(2,3),\ (4,5),\ (6,7),\ldots.$$
4. Interchange all rows and columns that were rotated.
5. If convergence has not been achieved, go to 1.

The notation above that specifies the pairs refers to the rows and columns at the current state; that is, after the interchanges up to that point. The interchange operation is a similarity transformation using an elementary permutation matrix (see page 81), and hence the eigenvalues are left unchanged by this operation. The method described above is a good one, but there are other ways of forming pairs. Some of the issues to consider are discussed by Luk and Park (1989), who analyzed and compared some proposed schemes.

7.4 QR Method

The most common algorithm for extracting eigenvalues is the QR method. While the power method and the Jacobi method require diagonalizable matrices, which restricts their practical use to symmetric matrices, the QR method can be used for nonsymmetric matrices. It is simpler for symmetric matrices, of course, because the eigenvalues are real. Also, for symmetric matrices the computer storage is less, the computations are fewer, and some transformations are particularly simple. In the following description, we will assume that the matrix is symmetric.

The basic idea behind the use of the QR method is that for a symmetric matrix A, the simple iterations beginning with $A^{(0)} = A$, for $k = 1, 2, \ldots$,

$$\begin{aligned} Q^{(k)}R^{(k)} &= A^{(k-1)} \\ A^{(k)} &= R^{(k)}Q^{(k)} \end{aligned}$$

lead to an orthogonal triangularization of A.

These iterations by themselves would be slow and would only work for certain matrices, so the QR method requires that the matrix first be transformed into upper Hessenberg form (see page 59). A matrix can be reduced to Hessenberg form in a finite number of similarity transformations using either Householder reflections or Givens rotations.

The Hessenberg form for a symmetric matrix is tridiagonal. The Hessenberg form allows a large savings in the subsequent computations, even for nonsymmetric matrices.

Even in the Hessenberg form, the matrices $A^{(k)}$ are shifted by $c^{(k)}I$, where $c^{(k)}I$ is an approximation of an eigenvalue, which can be obtained in various ways (see Trefethen and Bau 1997; pages 219 and following).

After the matrix has been transformed into a similar Hessenberg matrix, a sequence of similar Hessenberg matrices that converge to triangular matrix is formed. The QR method for determining the eigenvalues is iterative and produces a sequence of Hessenberg matrices that converge to a triangular matrix. An upper Hessenberg matrix is formed and its eigenvalues are extracted by a process called "chasing", which consists of steps that alternate between creating nonzero entries in positions $(i+2, i)$, $(i+3, i)$, and $(i+3, i+1)$ and restoring these entries to zero, as the nonzero entries are moved farther down the matrix. For example,

$$\begin{bmatrix} \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} \\ \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} \\ \mathtt{0} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} \\ \mathtt{0} & \mathtt{Y} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} \\ \mathtt{0} & \mathtt{Y} & \mathtt{Y} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} \\ \mathtt{0} & \mathtt{0} & \mathtt{0} & \mathtt{0} & \mathtt{X} & \mathtt{X} & \mathtt{X} \\ \mathtt{0} & \mathtt{0} & \mathtt{0} & \mathtt{0} & \mathtt{0} & \mathtt{X} & \mathtt{X} \end{bmatrix} \rightarrow \begin{bmatrix} \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} \\ \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} \\ \mathtt{0} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} \\ \mathtt{0} & \mathtt{0} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} \\ \mathtt{0} & \mathtt{0} & \mathtt{Y} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} \\ \mathtt{0} & \mathtt{0} & \mathtt{Y} & \mathtt{Y} & \mathtt{X} & \mathtt{X} & \mathtt{X} \\ \mathtt{0} & \mathtt{0} & \mathtt{0} & \mathtt{0} & \mathtt{0} & \mathtt{X} & \mathtt{X} \end{bmatrix}.$$

In the j^{th} step of the QR method, a bulge is created and is chased down the matrix by similarity transformations, usually Givens transformations,

$$G_k^{-1} A^{(j-1,k)} G_k.$$

The transformations are based on the eigenvalues of 2×2 matrices in the lower right-hand part of the matrix.

There are some variations on the way the chasing occurs. Haag and Watkins (1993) describe an efficient modified QR algorithm that uses both Givens transformations and Gaussian elimination transformations, with or without pivoting. For the $n \times n$ Hessenberg matrix $A^{(0,0)}$, the first step of the Haag-Watkins procedure begins with a 3×3 Householder reflection matrix, $\widetilde{G}_0$, whose first column is

$$(A^{(0,0)} - \sigma_1 I)(A^{(0,0)} - \sigma_2 I)e_1,$$

where σ_1 and σ_2 are the eigenvalues of the 2×2 matrix

$$\begin{bmatrix} a_{n-1,n-1} & a_{n-1,n} \\ a_{n-1,n} & a_{n,n} \end{bmatrix},$$

and e_1 is the first unit vector of length n. The $n \times n$ matrix G_0 is $\text{diag}(\widetilde{G}_0, I)$. The initial transformation $G_0^{-1}A^{(0,0)}G_0$ creates a bulge with nonzero elements $a_{31}^{(0,1)}$, $a_{41}^{(0,1)}$, and $a_{42}^{(0,1)}$.

After the initial transformation, the Haag-Watkins procedure makes $n-3$ transformations

$$A^{(0,k+1)} = G_k^{-1}A^{(0,k)}G_k,$$

for $k = 1, 2, \ldots, n-3$, that chase the bulge diagonally down the matrix, so that $A^{(0,k+1)}$ differs from Hessenberg form only by the nonzero elements $a_{k+3,k+1}^{(0,k+1)}$, $a_{k+4,k+1}^{(0,k+1)}$, and $a_{k+4,k+2}^{(0,k+1)}$. To accomplish this, the matrix G_k differs from the identity only in rows and columns $k+1$, $k+2$, and $k+3$. The transformation

$$G_k^{-1}A^{(0,k)}$$

annihilates the entries $a_{k+2,k}^{(0,k)}$ and $a_{k+3,k}^{(0,k)}$, and the transformation

$$(G_k^{-1}A^{(0,k)})G_k$$

produces $A^{(0,k+1)}$ with two new nonzero elements, $a_{k+4,k+1}^{(0,k+1)}$ and $a_{k+4,k+2}^{(0,k+1)}$. The final transformation in the first step, for $k = n-2$, annihilates $a_{n,n-2}^{(0,k)}$. The transformation matrix G_{n-2} differs from the identity only in rows and columns $n-1$ and n. These steps are iterated until the matrix becomes triangular. As the subdiagonal elements converge to zero, the shifts for use in the first transformation of a step (corresponding to σ_1 and σ_2) are determined by 2×2 submatrices higher on the diagonal. Special consideration must be given to situations in which these submatrices contain zero elements. For this, the reader is referred to Watkins (2002) or Golub and Van Loan (1996).

This description has just indicated the general flavor of the QR method. There are different variations on the overall procedure and then many computational details that must be observed. In the Haag-Watkins procedure, for example, the G_ks are not unique, and their form can affect the efficiency and the stability of the algorithm. Haag and Watkins (1993) describe criteria for the selection of the G_ks. They also discuss some of the details of programming the algorithm. A very careful description of the basic algorithm and various modifications is provided in Trefethen and Bau (1997), pages 196 through 224.

7.5 Krylov Methods

In the power method, we encountered the sequence

$$x,\ Ax,\ A^2x, \ldots .$$

This sequence is a finite Krylov space generating set. As we mentioned on page 284, several methods for computing eigenvalues are often based on a Krylov space,

$$\mathcal{K}_k = \mathcal{V}(\{v, Av, A^2v, \ldots, A^{k-1}v\}).$$

(Aleksei Krylov used these vectors to construct the characteristic polynomial.)

The two most important Krylov methods are the Lanczos tridiagonalization algorithm and the Arnoldi orthogonalization algorithm. We will not discuss these methods here but rather refer the interested reader to Golub and Van Loan (1996).

7.6 Generalized Eigenvalues

In Sect. 3.8.12, we defined the generalized eigenvalues and eigenvectors by replacing the identity in the definition of ordinary eigenvalues and eigenvectors by a general (square) matrix B:

$$|A - cB| = 0. \tag{7.17}$$

If there exists a finite c such that this determinant is zero, then there is some nonzero, finite vector v such that

$$Av = cBv. \tag{7.18}$$

As we have seen in the case of ordinary eigenvalues, symmetry of the matrix, because of diagonalizability, allows for simpler methods to evaluate the eigenvalues. In the case of generalized eigenvalues, symmetry together with positive definiteness allows us to reformulate the problem to be much simpler. If A and B are symmetric and B is positive definite, we refer to the pair (A, B) as *symmetric*.

If A and B are a symmetric pair, B has a Cholesky decomposition, $B = T^{\mathrm{T}}T$, where T is an upper triangular matrix with positive diagonal elements. We can therefore rewrite equation (7.18) as

$$T^{-\mathrm{T}}AT^{-1}u = cu, \tag{7.19}$$

where $u = Tv$. Note that because A is symmetric, $T^{-\mathrm{T}}AT^{-1}$ is symmetric, and since c is an eigenvalue of this matrix, it is real. Its associated eigenvector (with respect to $T^{-\mathrm{T}}AT^{-1}$) is likewise real, and therefore so is the generalized eigenvector v. Because $T^{-\mathrm{T}}AT^{-1}$ is symmetric, the ordinary eigenvectors can

be chosen to be orthogonal. (Recall from page 153 that eigenvectors corresponding to distinct eigenvalues *are* orthogonal, and those corresponding to a multiple eigenvalue can be chosen to be orthogonal.) This implies that the generalized eigenvectors of the symmetric pair (A, B) can be chosen to be B-conjugate.

Because of the equivalence of a generalized eigenproblem for a symmetric pair to an ordinary eigenproblem for a symmetric matrix, any of the methods discussed in this chapter can be used to evaluate the generalized eigenpairs of a symmetric pair. The matrices in statistical applications for which the generalized eigenvalues are required are often symmetric pairs. For example, Roy's maximum root statistic, which is used in multivariate analysis, is a generalized eigenvalue of two Wishart matrices.

The generalized eigenvalues of a pair that is not symmetric are more difficult to evaluate. The approach of forming upper Hessenberg matrices, as in the QR method, is also used for generalized eigenvalues. We will not discuss this method here but instead refer the reader to Watkins (2002) for a description of the method, which is called the QZ algorithm.

7.7 Singular Value Decomposition

The standard algorithm for computing the singular value decomposition

$$A = UDV^{\mathrm{T}}$$

is due to Golub and Reinsch (1970) and is built on ideas of Golub and Kahan (1965). The first step in the Golub-Reinsch algorithm for the singular value decomposition of the $n \times m$ matrix A is to reduce A to upper bidiagonal form:

$$A^{(0)} = \begin{bmatrix} \mathtt{X} & \mathtt{X} & 0 & \cdots & 0 & 0 \\ 0 & \mathtt{X} & \mathtt{X} & \cdots & 0 & 0 \\ 0 & 0 & \mathtt{X} & \cdots & 0 & 0 \\ & & & \ddots & \ddots & \\ 0 & 0 & 0 & \cdots & \mathtt{X} & \mathtt{X} \\ 0 & 0 & 0 & \cdots & 0 & \mathtt{X} \\ 0 & 0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 0 \end{bmatrix}.$$

We assume $n \geq m$. (If this is not the case, we merely use A^{T}.) This algorithm is basically a factored form of the QR algorithm for the eigenvalues of $A^{(0)\mathrm{T}}A^{(0)}$, which would be symmetric and tridiagonal.

The Golub-Reinsch method produces a sequence of upper bidiagonal matrices, $A^{(0)}, A^{(1)}, A^{(2)}, \ldots$, which converges to the diagonal matrix D. (Each of these has a zero submatrix below the square submatrix.) Similar to the QR method for eigenvalues, the transformation from $A^{(j)}$ to $A^{(j+1)}$ is effected by a sequence of orthogonal transformations,

$$\begin{aligned} A^{(j+1)} &= R_{m-2}^{\mathrm{T}} R_{m-3}^{\mathrm{T}} \cdots R_0^{\mathrm{T}} A^{(j)} T_0 T_1 \cdots T_{m-2} \\ &= R^{\mathrm{T}} A^{(j)} T, \end{aligned}$$

which first introduces a nonzero entry below the diagonal (T_0 does this) and then chases it down the diagonal. After T_0 introduces a nonzero entry in the $(2,1)$ position, R_0^{T} annihilates it and produces a nonzero entry in the $(1,3)$ position; T_1 annihilates the $(1,3)$ entry and produces a nonzero entry in the $(3,2)$ position, which R_1^{T} annihilates, and so on. Each of the R_ks and T_ks are Givens transformations, and, except for T_0, it should be clear how to form them.

If none of the elements along the main diagonal or the diagonal above the main diagonal is zero, then T_0 is chosen as the Givens transformation such that T_0^{T} will annihilate the second element in the vector

$$(a_{11}^2 - \sigma_1,\ a_{11}a_{12},\ 0,\ \cdots,\ 0),$$

where σ_1 is the eigenvalue of the lower right-hand 2×2 submatrix of $A^{(0)\mathrm{T}} A^{(0)}$ that is closest in value to the (m,m) element of $A^{(0)\mathrm{T}} A^{(0)}$. This is easy to compute (see Exercise 7.6).

If an element along the main diagonal or the diagonal above the main diagonal is zero, we must proceed slightly differently. (Remember that for purposes of computations "zero" generally means "near zero"; that is, to within some set tolerance.)

If an element above the main diagonal is zero, the bidiagonal matrix is separated at that value into a block diagonal matrix, and each block (which is bidiagonal) is treated separately.

If an element on the main diagonal, say a_{kk}, is zero, then a singular value is zero. In this case, we apply a set of Givens transformations from the left. We first use G_1, which differs from the identity only in rows and columns k and $k+1$, to annihilate the $(k, k+1)$ entry and introduce a nonzero in the $(k, k+2)$ position. We then use G_2, which differs from the identity only in rows and columns k and $k+2$, to annihilate the $(k, k+2)$ entry and introduce a nonzero in the $(k, k+3)$ position. Continuing this process, we form a matrix of the form

$$\begin{bmatrix} \mathtt{X} & \mathtt{X} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \mathtt{X} & \mathtt{X} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathtt{X} & \mathtt{Y} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathtt{X} & \mathtt{X} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathtt{X} & \mathtt{X} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \mathtt{X} & \mathtt{X} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathtt{X} \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

The Y in this matrix (in position $(k-1, k)$) is then chased up the upper block consisting of the first k rows and columns of the original matrix by using Givens transformations applied from the right. This then yields two block bidiagonal matrices (and a 1×1 0 matrix). We operate on the individual blocks as before.

After the steps have converged to yield a diagonal matrix, $\widetilde{D}$, all of the Givens matrices applied from the left are accumulated into a single matrix and all from the right are accumulated into a single matrix to yield a decomposition

$$A = \widetilde{U}\widetilde{D}\widetilde{V}^{\mathrm{T}}.$$

There is one last thing to do. The elements of $\widetilde{D}$ may not be nonnegative. This is easily remedied by postmultiplying by a diagonal matrix G that is the same as the identity except for having a -1 in any position corresponding to a negative value in $\widetilde{D}$. In addition, we generally form the singular value decomposition is such a way that the elements in D are nonincreasing. The entries in $\widetilde{D}$ can be rearranged by a permutation matrix $E_{(\pi)}$ so they are in nonincreasing order. So we have

$$D = E_{(\pi)}^{\mathrm{T}}\widetilde{D}GE_{(\pi)},$$

and the final decomposition is

$$\begin{aligned} A &= \widetilde{U}E_{(\pi)}GDE_{(\pi)}^{\mathrm{T}}\widetilde{V}^{\mathrm{T}} \\ &= UDV^{\mathrm{T}}. \end{aligned}$$

If $n \geq \frac{5}{3}m$, a modification of this algorithm by Chan (1982a,b) is more efficient than the standard Golub-Reinsch method.

Exercises

7.1. Simple matrices and the power method.

a) Let A be an $n \times n$ matrix whose elements are generated independently (but not necessarily identically) from real-valued continuous distributions. What is the probability that A is simple?

b) Under the same conditions as in Exercise 7.1a, and with $n \geq 3$, what is the probability that $|c_{n-2}| < |c_{n-1}| < |c_n|$, where c_{n-2}, c_{n-1}, and c_n are the three eigenvalues with the largest absolute values?

c) Prove that the power method converges linearly if $|c_{n-2}| < |c_{n-1}| < |c_n|$, $b_{n-1} \neq 0$, and $b_n \neq 0$. (The bs are the coefficients in the expansion of $x^{(0)}$.)
Hint: Substitute the expansion in equation (7.11) on page 314 into the expression for the convergence ratio in equation (7.12).

d) Suppose A is simple and the elements of $x^{(0)}$ are generated independently (but not necessarily identically) from continuous distributions. What is the probability that the power method will converge linearly?

7.2. Consider the matrix

$$\begin{bmatrix} 4 & 1 & 2 & 3 \\ 1 & 5 & 3 & 2 \\ 2 & 3 & 6 & 1 \\ 3 & 2 & 1 & 7 \end{bmatrix}.$$

a) Use the power method to determine the largest eigenvalue and an associated eigenvector of this matrix.
b) Find a 3×3 matrix, as in equation (7.7), that has the same eigenvalues as the remaining eigenvalues of the matrix above.
c) Using Givens transformations, reduce the matrix to upper Hessenberg form.

7.3. In the matrix

$$\begin{bmatrix} 2 & 1 & 0 & 0 \\ 1 & 5 & 2 & 0 \\ 3 & 2 & 6 & 1 \\ 0 & 0 & 1 & 8 \end{bmatrix},$$

determine the Givens transformations to chase the 3 in the $(3, 1)$ position out of the matrix.

7.4. In the matrix

$$\begin{bmatrix} 2 & 1 & 0 & 0 \\ 3 & 5 & 2 & 0 \\ 0 & 0 & 6 & 1 \\ 0 & 0 & 0 & 8 \\ 0 & 0 & 0 & 0 \end{bmatrix},$$

determine the Givens transformations to chase the 3 in the $(2, 1)$ position out of the matrix.

7.5. In the QR methods for eigenvectors and singular values, why can we not just use additional orthogonal transformations to triangularize the given matrix (instead of just forming a similar Hessenberg matrix, as in Sect. 7.4) or to diagonalize the given matrix (instead of just forming the bidiagonal matrix, as in Sect. 7.7)?

7.6. Determine the eigenvalue σ_1 (on page 323) used in forming the matrix T_0 for initiating the chase in the algorithm for the singular value decomposition. Express it in terms of $a_{m,m}$, $a_{m-1,m-1}$, $a_{m-1,m}$, and $a_{m-1,m-2}$.

Part II

Applications in Data Analysis

8

Special Matrices and Operations Useful in Modeling and Data Analysis

In previous chapters, we encountered a number of special matrices, such as symmetric matrices, banded matrices, elementary operator matrices, and so on. In this chapter, we will discuss some of these matrices in more detail and also introduce some other special matrices and data structures that are useful in statistics.

There are a number of special kinds of matrices that are useful in statistical applications. In statistical applications in which data analysis is the objective, the initial step is the representation of observational data in some convenient form, which often is a matrix. The matrices for operating on observational data or summarizing the data often have special structures and properties. We discuss the representation of observations using matrices in Sect. 8.1.

In Sect. 8.2, we review and discuss some of the properties of symmetric matrices.

One of the most important properties of many matrices occurring in statistical data analysis is nonnegative or positive definiteness; this is the subject of Sects. 8.3 and 8.4.

Fitted values of a response variable that are associated with given values of covariates in linear models are often projections of the observations onto a subspace determined by the covariates. Projection matrices and Gramian matrices useful in linear models are considered in Sects. 8.5 and 8.6.

Another important property of many matrices occurring in statistical modeling is irreducible nonnegativeness or positiveness; this is the subject of Sect. 8.7.

Many of the defining properties of the special matrices discussed in this chapter are invariant under scalar multiplication; hence, the special matrices are members of cones. Interestingly even further, convex combinations of some types of special matrices yield special matrices of the same type; hence, those special matrices are members of convex cones (see Sect. 2.2.8 beginning on page 43).

J.E. Gentle, *Matrix Algebra*, Springer Texts in Statistics,
DOI 10.1007/978-3-319-64867-5_8

8.1 Data Matrices and Association Matrices

There are several ways that data can be organized for representation in the computer. We distinguish logical structures from computer-storage structures. Data structure in computers is an important concern and can greatly affect the efficiency of computer processing. We discuss some simple aspects of the organization for computer storage in Sect. 11.1, beginning on page 523. In the present section, we consider some general issues of logical organization and structure.

There are two important aspects of data in applications that we will not address here. One is metadata; that is, data about the data. Metadata includes names or labels associated with data, information about how and when the data were collected, information about how the data are stored in the computer, and so on. Another important concern in applications is missing data. In real-world applications it is common to have incomplete data. If the data are stored in some structure that naturally contains a cell or a region for the missing data, the computer representation of the dataset must contain some indication that the cell is empty. For numeric data, the convenient way of doing this is by using "not-available", NA (see page 464), or "not-a-number", NaN (see page 475). The effect on a statistical analysis when some data are missing varies with the type of analysis. We consider some effects of missing data on the estimation of variance-covariance matrices in Sect. 9.5.6, beginning on page 437.

8.1.1 Flat Files

If several features or attributes are observed on each of several entities, a convenient way of organizing the data is as a two-dimensional array with each column corresponding to a specific feature and each row corresponding to a specific observational entity. In the field of statistics, data for the features are stored in "variables", the entities are called "observational units", and a row of the array is called an "observation" (see Fig. 8.1).

	Var 1	Var 2	...	Var m
Obs 1	x	x	...	x
Obs 2	x	x	...	x
⋮	⋮	⋮	...	⋮
Obs n	x	x	...	x

Figure 8.1. Data appropriate for representation in a flat file

The data may be various types of objects, such as names, real numbers, numbers with associated measurement units, sets, vectors, and so on. If the data are represented as real numbers, the data array is a matrix. (Note again our use of the word "matrix"; not just any rectangular array is a matrix in the sense used in this book.) Other types of data can often be made equivalent to a matrix in an intuitive manner.

The flat file arrangement emphasizes the relationships of the data both *within* an observational unit or row and *within* a variable or column. Simple operations on the data matrix may reveal relationships *among* observational units or *among* variables.

Flat files are the appropriate data structure for the analysis of linear models, but statistics is not just about analysis of linear models anymore. (It never was.)

8.1.2 Graphs and Other Data Structures

If the numbers of measurements on the observational units varies or if the interest is primarily in simple relationships among observational units or among variables, the flat file structure may not be very useful. Sometimes a graph structure can be used advantageously.

A *graph* is a nonempty set V of points, called *vertices*, together with a collection E of unordered pairs of elements of V, called *edges*. (Other definitions of "graph" allow the null set to be a graph.) If we let $\mathcal{G}$ be a graph, we represent it as (V, E). We often represent the set of vertices as $V(\mathcal{G})$ and the collection of edges as $E(\mathcal{G})$. An edge is said to be *incident* on each vertex in the edge. The number of vertices (that is, the cardinality of V) is the *order of the graph*, and the number of edges, the cardinality of E, is the *size of the graph*.

An edge in which the two vertices are the same is called a *loop*. If two or more elements of $E(\mathcal{G})$ contain the same two vertices, those edges are called *multiple edges*, and the graph itself is called a *multigraph*. A graph with no loops and with no multiple edges is called a *simple graph*. In some literature, "graph" means "simple graph", as I have defined it; and a graph as I have defined it that is not simple is called a "pseudograph"

A *path* or *walk* is a sequence of edges, $e_1, \ldots, e_n$, such that for $i \geq 2$ one vertex in e_i is a vertex in edge e_{i-1}. Alternatively, a path or walk is defined as a sequence of vertices with common edges.

A graph such that there is a path that includes any pair of vertices is said to be *connected*.

A graph with more than one vertex such that all possible pairs of vertices occur as edges is a *complete graph*.

A *closed path* or *closed walk* is a path such that a vertex in the first edge (or the first vertex in the alternate definition) is in the last edge (or the last vertex).

A *cycle* is a closed path in which all vertices occur exactly twice (or in the alternate definition, in which all vertices except the first and the last are distinct). A graph with no cycles is said to be *acyclic*. An acyclic graph is also called a *tree*. Trees are used extensively in statistics to represent clusters.

The number of edges that contain a given vertex (that is, the number of edges incident on the vertex v) denoted by $d(v)$ is the *degree of the vertex*.

A vertex with degree 0 is said to be *isolated*.

We see immediately that the sum of the degrees of all vertices equals twice the number of edges, that is,

$$\sum d(v_i) = 2\#(E).$$

The sum of the degrees hence must be an even number.

A *regular graph* is one for which $d(v_i)$ is constant for all vertices v_i; more specifically, a graph is *k-regular* if $d(v_i) = k$ for all vertices v_i.

The natural data structure for a graph is a pair of lists, but a graph is often represented graphically (no pun!) as in Fig. 8.2, which shows a graph with five vertices and seven edges. While a matrix is usually not an appropriate structure for representing raw data from a graph, there are various types of matrices that are useful for studying the data represented by the graph, which we will discuss in Sect. 8.8.9.

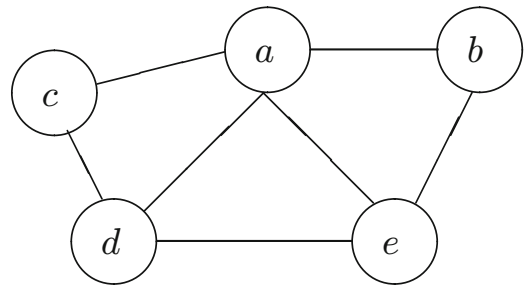


Figure 8.2. A simple graph

If $\mathcal{G}$ is the graph represented in Fig. 8.2, the vertices are

$$V(\mathcal{G}) = \{a, b, c, d, e\}$$

and the edges are

$$E(\mathcal{G}) = \{(a, b), (a, c), (a, d), (a, e), (b, e), (c, d), (d, e)\}.$$

The presence of an edge between two vertices can indicate the existence of a relationship between the objects represented by the vertices. The graph represented in Fig. 8.2 may represent five observational units for which our primary interest is in their relationships with one another. For example, the

observations may be authors of scientific papers, and an edge between two authors may represent the fact that the two have been coauthors on some paper.

	a	b	c	d	e
a		Y	Y	Y	Y
b	Y				Y
c	Y			Y	
d	Y		Y		Y
e	Y	Y		Y	

Figure 8.3. An alternate representation

The same information represented in the 5-order graph of Fig. 8.2 may be represented in a 5×5 rectangular array, as in Fig. 8.3.

In the graph represented in Fig. 8.2, there are no isolated vertices and the graph is connected. (Note that a graph with no isolated vertices is not necessarily connected.) The graph represented in Fig. 8.2 is not complete because, for example, there is no edge that contains vertices c and e. The graph is cyclic because of the closed path (defined by vertices) (c, d, e, b, a, c). Note that the closed path (c, d, a, e, b, a, c) is not a cycle.

This use of a graph immediately suggests various extensions of a basic graph. For example, E may be a multiset, with multiple instances of edges containing the same two vertices, perhaps, in the example above, representing multiple papers in which the two authors are coauthors. As we stated above, a graph in which E is a multiset is called a multigraph. Instead of just the presence or absence of edges between vertices, a *weighted graph* may be more useful; that is, one in which a real number is associated with a pair of vertices to represent the strength of the relationship, not just presence or absence, between the two vertices. A degenerate weighted graph (that is, an unweighted graph as discussed above) has weights of 0 or 1 between all vertices. A multigraph is a weighted graph in which the weights are restricted to nonnegative integers. Although the data in a weighted graph carry much more information than a graph with only its edges, or even a multigraph that allows strength to be represented by multiple edges, the simplicity of a graph sometimes recommends its use even when there are varying degrees of strength of relationships. A standard approach in applications is to set a threshold for the strength of relationship and to define an edge only when the threshold is exceeded.

8.1.2.1 Adjacency Matrix: Connectivity Matrix

The connections between vertices in the graphs shown in Fig. 8.2 or in Fig. 8.4 can be represented in an association matrix called an *adjacency matrix*, a *connectivity matrix*, or an *incidence matrix* to represent edges between vertices, as shown in equation (8.1). (The terms "adjacency", "connectivity", and "incidence" are synonymous. "Adjacency" is perhaps the most commonly used term, but I will naturally use both that term and "connectivity" because of the connotative value of the latter term.) The graph, $\mathcal{G}$, represented in Fig. 8.2 has the symmetric adjacency matrix

$$A(\mathcal{G}) = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{bmatrix}. \tag{8.1}$$

As above, we often use this kind of notation; a symbol, such as $\mathcal{G}$, represents a particular graph, and other objects that relate to the graph make use of that symbol.

There is no difference in the connectivity matrix and a table such as in Fig. 8.3 except for the metadata.

A graph as we have described it is "nondirected"; that is, an edge has no direction. The edge (a, b) is the same as (b, a). An adjacency matrix for a nondirected graph is symmetric.

An interesting property of an adjacency matrix, which we will discuss further on page 393, is that if A is the adjacency matrix of a graph, then A^k_{ij} is the number of paths of length k between nodes i and j in that graph. (See Exercise 8.20.)

The adjacency matrix for graph with no loops is hollow; that is, all diagonal elements are 0s. Another common way of representing a hollow adjacency matrix is to use -1s in place of the off-diagonal zeroes; that is, the absence of a connection between two different vertices is denoted by -1 instead of by 0. Such a matrix is called a *Seidel adjacency matrix*. (This matrix has no relationship to the Gauss-Seidel method discussed in Chap. 6.)

The relationship can obviously be defined in the other direction; that is, given an $n \times n$ symmetric matrix A, we define the *graph of the matrix* as the graph with n vertices and edges between vertices i and j if $a_{ij} \neq 0$. We often denote the graph of the matrix A by $\mathcal{G}(A)$.

Generally we restrict the elements of the connectivity matrix to be 1 or 0 to indicate only presence or absence of a connection, but not to indicate strength of the connection. In this case, a connectivity matrix is a nonnegative matrix; that is, all of its elements are nonnegative. We indicate that a matrix A is nonnegative by

$$A \geq 0.$$

We discuss the notation and properties of nonnegative (and positive) matrices in Sect. 8.7.

8.1.2.2 Digraphs

Another extension of a basic graph is one in which the relationship may not be the same in both directions. This yields a *digraph*, or "directed graph", in which the edges are ordered pairs called directed edges. The vertices in a digraph have two kinds of degree, an *indegree* and an *outdegree*, with the obvious meanings.

The simplest applications of digraphs are for representing networks. Consider, for example, the digraph represented by the network in Fig. 8.4. This is a network with five vertices, perhaps representing cities, and directed edges between some of the vertices. The edges could represent airline connections between the cities; for example, there are flights from x to u and from u to x, and from y to z, but not from z to y.

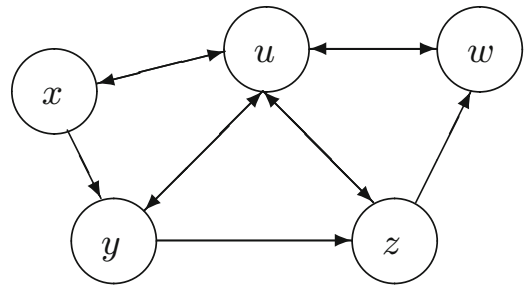


Figure 8.4. A simple digraph

Figure 8.4 represents a digraph with order 5 (there are five vertices) and size 11 (eleven directed edges). A sequence of edges, $e_1, \ldots, e_n$, constituting a path in a digraph must be such that for $i \geq 2$ the first vertex in e_i is the second vertex in edge e_{i-1}. For example, the sequence x, y, z, w, u, x in the graph of Fig. 8.4 is a path (in fact, a cycle) but the sequence x, u, w, z, y, x is not a path.

The connectivity matrix for the digraph in Fig. 8.4 with nodes ordered as u, w, x, y, z is

$$C = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \end{bmatrix}. \tag{8.2}$$

A connectivity matrix for a (nondirected) graph is symmetric, but for a digraph it is not necessarily symmetric. Given an $n \times n$ matrix A, we define the *digraph of the matrix* as the digraph with n vertices and edges from vertex i to j if $a_{ij} \neq 0$. We use the same notation for a digraph as we used above for a graph, $\mathcal{G}(A)$.

In statistical applications, graphs are used for representing symmetric associations. Digraphs are used for representing asymmetric associations or one-way processes such as a stochastic process.

In a simple digraph, the edges only indicate the presence or absence of a relationship, but just as in the case of a simple graph, we can define a weighted digraph by associating nonnegative numbers with each directed edge.

Graphical modeling is useful for analyzing relationships between elements of a collection of sets. For example, in an analysis of internet traffic, profiles of users may be constructed based on the set of web sites each user visits in relation to the sets visited by other users. For this kind of application, an intersection graph may be useful. An *intersection graph*, for a given collection of sets $\mathcal{S}$, is a graph whose vertices correspond to the sets in $\mathcal{S}$ and whose edges between any two sets have a common element.

The word "graph" is often used without qualification to mean any of these types.

8.1.2.3 Connectivity of Digraphs

There are two kinds of connected digraphs. A digraph such that there is a (directed) path that includes any pair of vertices is said to be *strongly connected*. A digraph such that there is a path without regard to the direction of any edge that includes any pair of vertices is said to be *weakly connected*. The digraph shown in Fig. 8.4 is strongly connected. The digraph shown in Fig. 8.5 is weakly connected but not strongly connected.

A digraph that is not weakly connected must have two sets of nodes with no edges between any nodes in one set and any nodes in the other set.

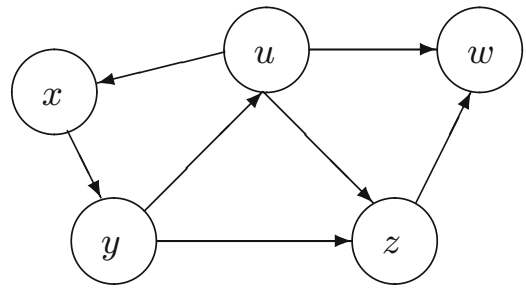


Figure 8.5. A digraph that is not strongly connected

The connectivity matrix of the digraph in Fig. 8.5 is

$$C = \begin{bmatrix} 0\,1\,1\,0\,1 \\ 0\,0\,0\,0\,0 \\ 0\,0\,0\,1\,0 \\ 1\,0\,0\,0\,1 \\ 0\,1\,0\,0\,0 \end{bmatrix}. \tag{8.3}$$

The matrix of a digraph that is not strongly connected can always be reduced to a special block upper triangular form by row and column permutations; that is, if the digraph $\mathcal{G}$ is not strongly connected, then there exists a permutation matrix $E_{(\pi)}$ such that

$$E_{(\pi)}A(\mathcal{G})E_{(\pi)} = \begin{bmatrix} B_{11} & B_{12} \\ 0 & B_{22} \end{bmatrix}, \tag{8.4}$$

where B_{11} and B_{22} are square. Such a transformation is called a *symmetric permutation*.

Later we will formally prove this relationship between strong connectivity and this reduced form of the matrix, but first we consider the matrix in equation (8.3). If we interchange the second and fourth columns and rows, we get the reduced form

$$E_{24}CE_{24} = \left[\begin{array}{ccc|cc} 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{array}\right].$$

8.1.2.4 Irreducible Matrices

Any nonnegative square matrix that can be permuted into the form in equation (8.4) with square diagonal submatrices is said to be *reducible*; a matrix that cannot be put into that form is *irreducible*. We also use the terms reducible and irreducible to refer to the graph itself.

Irreducible matrices have many interesting properties, some of which we will discuss in Sect. 8.7.3, beginning on page 375. The implication (8.77) in that section provides a simple characterization of irreducibility.

8.1.2.5 Strong Connectivity of Digraphs and Irreducibility of Matrices

A nonnegative matrix is irreducible if and only if its digraph is strongly connected. Stated another way, a digraph is not strongly connected if and only if its matrix is reducible.

To see this, first consider a reducible matrix. In its reduced form of equation (8.4), none of the nodes corresponding to the last rows have directed edges leading to any of the nodes corresponding to the first rows; hence, the digraph is not strongly connected.

Now, assume that a given digraph $\mathcal{G}$ is not strongly connected. In that case, there is some node, say the i^{th} node, from which there is no directed path to some other node. Assume that there are $m-1$ nodes that *can* be reached from node i. If $m = 1$, then we have a trivial partitioning of the $n \times n$ connectivity in which B_{11} of equation (8.4) is $(n-1) \times (n-1)$ and B_{22} is

a 1×1 0 matrix (that is, 0_1). If $m \geq 1$, perform symmetric permutations so that the row corresponding to node i and all other $m-1$ nodes are the last m rows of the permuted connectivity matrix. In this case, the first $n-m$ elements in each of those rows must be 0. To see that this must be the case, let $k > n-m$ and $j \leq n-m$ and assume that the element in the $(k,j)^{\text{th}}$ position is nonzero. In that case, there is a path from node i to node k to node j, which is in the set of nodes not reachable from node i; hence the $(k,j)^{\text{th}}$ element (in the permuted matrix) must be 0. The submatrix corresponding to B_{11} is $n-m \times n-m$, and that corresponding to B_{22} is $m \times m$. These properties also hold for connectivity matrices with simple loops (with 1s on the diagonal) and for an augmented connectivity matrix (see page 393).

Reducibility plays an important role in the analysis of Markov chains (see Sect. 9.8.1).

8.1.3 Term-by-Document Matrices

An interesting area of statistical application is in clustering and classifying documents. In the simpler cases, the documents consist of text only, and much recent research has been devoted to "text data-mining". (The problem is not a new one; Mosteller and Wallace 1963, studied a related problem.)

We have a set of text documents (often called a "corpus"). A basic set of data to use in studying the text documents is the *term-document matrix*, which is a matrix whose columns correspond to documents and whose rows correspond to the various terms used in the documents. The terms are usually just words. The entries in the term-document matrix are measures of the importance of the term in the document. Importance may be measured simply by the number of times the word occurs in the document, possibly weighted by some measure of the total number of words in the document. In other measures of the importance, the relative frequency of the word in the given document may be adjusted for the relative frequency of the word in the corpus. (Such a measure is called term frequency-inverse document frequency or tf-idf.) Certain common words, such as "the", called "stop-words", may be excluded from the data. Also, words may be "stemmed"; that is, they may be associated with a root that ignores modifying letters such as an "s" in a plural or an "ed" in a past-tense verb. Other variations include accounting for sequences of terms (called "collocation").

There are several interesting problems that arise in text analysis. Term-document matrices can be quite large. Terms are often misspelled. The documents may be in different languages. Some terms may have multiple meanings. The documents themselves may be stored on different computer systems.

Two types of manipulation of the term-document matrix are commonly employed in the analysis. One is singular-value decomposition (SVD), and the other is nonnegative matrix factorization (NMF).

SVD is used in what is called "latent semantic analysis" ("LSA"), or "latent semantic indexing" ("LSI"). A variation of latent semantic analysis is

"probabilistic" latent semantic analysis, in which a probability distribution is assumed for the words. In a specific instance of probabilistic latent semantic analysis, the probability distribution is a Dirichlet distribution. (Most users of this latter method are not statisticians; they call the method "LDA", for "latent Dirichlet allocation".)

NMF is often used in determining which documents are similar to each other or, alternatively, which terms are clustered in documents. If A is the term-document matrix, and $A = WH$ is a nonnegative factorization, then the element w_{ij} can be interpreted as the degree to which term i belongs to cluster j, while element h_{ij} can be interpreted as the degree to which document j belongs to cluster i. A method of clustering the documents is to assign document j (corresponding to the j^{th} column of A) to the k^{th} cluster if h_{kj} is the maximum element in the column h_{*j}.

There is a wealth of literature on text data-mining, but we will not discuss the analysis methods further.

8.1.4 Probability Distribution Models

Probability models in statistical data analysis are often multivariate distributions, and hence, matrices arise in the model. In the analysis itself, matrices that represent associations are computed from the observational data. In this section we mention some matrices in the models, and in the next section we refer to some association matrices that are computed in the analysis.

Data in rows of flat files are often assumed to be realizations of vector random variables, some elements of which may have a degenerate distribution (that is, the elements in some columns of the data matrix may be considered to be fixed rather than random). The data in one row are often considered independent of the data in another row. Statistical data analysis is generally concerned with studying various models of relationships among the elements of the vector random variables. For example, the familiar linear regression model relates one variable (one column) to a linear combination of other variables plus a translation and random noise.

A *random graph* of fixed order is a discrete probability space over all possible graphs of that order. For a graph of order n, there are $2^{\binom{n}{2}}$ possible graphs. Asymptotic properties of the probability distribution refer to the increase of the order without limit. Occasionally it is useful to consider the order of the graph to be random also. If the order is unrestricted, the sample space for a random graph of random order is infinite but countable. The number of digraphs of order n is $4^{\binom{n}{2}}$.

Random graphs have many uses in the analysis of large systems of interacting objects; for example, a random intersection graph may be used to make inferences about the clustering of internet users based on the web sites they visit.

8.1.5 Derived Association Matrices

In data analysis, the interesting questions usually involve the relationships among the variables or among the observational units. Matrices formed from the original data matrix for the purpose of measuring these relationships are called *association matrices*. There are basically two types: similarity and dissimilarity matrices. The variance-covariance matrix, which we discuss in Sect. 8.6.3, is an example of an association matrix that measures similarity. We discuss dissimilarity matrices in Sect. 8.6.6 and in Sect. 8.8.9 discuss a type of similarity matrix for data represented in graphs.

In addition to the distinction between similarity and dissimilarity association matrices, we may identify two types of association matrices based on whether the relationships of interest are among the rows (observations) or among the columns (variables or features). In applications, dissimilarity relationships among rows tend to be of more interest, and similarity relationships among columns are usually of more interest. (The applied statistician may think of clustering, multidimensional scaling, or Q factor analysis for the former and correlation analysis, principal components analysis, or factor analysis for the latter.)

8.2 Symmetric Matrices and Other Unitarily Diagonalizable Matrices

Most association matrices encountered in applications are real and symmetric. Because real symmetric matrices occur so frequently in statistical applications and because such matrices have so many interesting properties, it is useful to review some of those properties that we have already encountered and to state some additional properties.

First, perhaps, we should iterate a trivial but important fact: the product of symmetric matrices is not, in general, symmetric. A power of a symmetric matrix, however, is symmetric.

We should also emphasize that some of the special matrices we have discussed are assumed to be symmetric because, if they were not, we could define equivalent symmetric matrices. This includes positive definite matrices and more generally the matrices in quadratic forms.

8.2.1 Some Important Properties of Symmetric Matrices

For convenience, here we list some of the important properties of symmetric matrices, many of which concern their eigenvalues. In the following, let A be a real symmetric matrix with eigenvalues c_i and corresponding eigenvectors v_i.

- If k is any positive integer, A^k is symmetric.
- AB is not necessarily symmetric even if B is a symmetric matrix.

- $A \otimes B$ is symmetric if B is symmetric.
- If A is nonsingular, then A^{-1} is also symmetric because $(A^{-1})^{\mathrm{T}} = (A^{\mathrm{T}})^{-1} = A^{-1}$.
- If A is nonsingular (so that A^k is defined for nonpositive integers), A^k is symmetric and nonsingular for any integer k.
- All eigenvalues of A are real (see page 140).
- A is diagonalizable (or simple), and in fact A is orthogonally diagonalizable; that is, it has an orthogonally similar canonical factorization, $A = VCV^{\mathrm{T}}$ (see page 154).
- A has the spectral decomposition $A = \sum_i c_i v_i v_i^{\mathrm{T}}$, where the c_i are the eigenvalues and v_i are the corresponding eigenvectors (see page 155).
- A power of A has the spectral decomposition $A^k = \sum_i c_i^k v_i v_i^{\mathrm{T}}$.
- Any quadratic form $x^{\mathrm{T}}Ax$ can be expressed as $\sum_i b_i^2 c_i$, where the b_i are elements in the vector $V^{-1}x$.
- We have

$$\max_{x \neq 0} \frac{x^{\mathrm{T}}Ax}{x^{\mathrm{T}}x} = \max\{c_i\}$$

  (see page 156). If A is nonnegative definite, this is the spectral radius $\rho(A)$.
- For the L_2 norm of the symmetric matrix A, we have

$$\|A\|_2 = \rho(A).$$

- For the Frobenius norm of the symmetric matrix A, we have

$$\|A\|_F = \sqrt{\sum c_i^2}.$$

  This follows immediately from the fact that A is diagonalizable, as do the following properties.
-
$$\mathrm{tr}(A) = \sum c_i$$

-
$$|A| = \prod c_i$$

  (see equations (3.227) and (3.228) on page 140).

8.2.2 Approximation of Symmetric Matrices and an Important Inequality

In Sect. 3.10, we considered the problem of approximating a given matrix by another matrix of lower rank. There are other situations in statistics in which we need to approximate one matrix by another one. In data analysis, this may be because our given matrix arises from poor observations and we know the "true" matrix has some special properties not possessed by the given matrix computed from the data. A familiar example is a sample variance-covariance

matrix computed from incomplete data (see Sect. 9.5.6). Other examples in statistical applications occur in the simulation of random matrices (see Gentle 2003; Section 5.3.3). In most cases of interest, the matrix to be approximated is a symmetric matrix.

Consider the difference of two symmetric $n \times n$ matrices, A and $\widetilde{A}$; that is,

$$E = A - \widetilde{A}. \tag{8.5}$$

The matrix of the differences, E, is also symmetric. We measure the "closeness" of A and $\widetilde{A}$ by some norm of E.

The Hoffman-Wielandt theorem gives a lower bound on the Frobenius norm of E in terms of the differences of the eigenvalues of A and $\widetilde{A}$: if the eigenvalues of A are $c_1, \ldots c_n$ and the eigenvalues of $\widetilde{A}$ are $\tilde{c}_1, \ldots \tilde{c}_n$, each set being arranged in nonincreasing order, we have

$$\sum_{i=1}^{n}(c_i - \tilde{c}_i)^2 \leq \|E\|_{\mathrm{F}}^2. \tag{8.6}$$

This fact was proved by Hoffman and Wielandt (1953) using techniques from linear programming. Wilkinson (1965) gives a simpler proof (which he attributes to Wallace Givens) along the following lines.

Because A, $\widetilde{A}$, and E are symmetric, they are all orthogonally diagonalizable. Let the diagonal factorizations of A and E, respectively, be VCV^{T} and $U\mathrm{diag}((e_1, \ldots, e_n))U^{\mathrm{T}}$, where $e_1, \ldots e_n$ are the eigenvalues of E in nonincreasing order. Hence, we have

$$\begin{aligned} U\mathrm{diag}((e_1, \ldots, e_n))U^{\mathrm{T}} &= U(A - \widetilde{A})U^{\mathrm{T}} \\ &= U(VCV^{\mathrm{T}} - \widetilde{A})U^{\mathrm{T}} \\ &= UV(C - V^{\mathrm{T}}\widetilde{A}V)V^{\mathrm{T}}U^{\mathrm{T}}. \end{aligned}$$

Taking norms of both sides, we have

$$\sum_{i=1}^{n} e_i^2 = \|C - V^{\mathrm{T}}\widetilde{A}V\|^2. \tag{8.7}$$

(All norms in the remainder of this section will be the Frobenius norm.) Now, let

$$f(Q) = \|C - Q^{\mathrm{T}}\widetilde{A}Q\|^2 \tag{8.8}$$

be a function of any $n \times n$ orthogonal matrix, Q. (Equation (8.7) yields $f(V) = \sum e_i^2$.) To arrive at inequality (8.6), we show that this function is bounded below by the sum of the differences in the squares of the elements of C (which are the eigenvalues of A) and the eigenvalues of $Q^{\mathrm{T}}\widetilde{A}Q$ (which are the eigenvalues of the matrix approximating A).

Because the elements of Q are bounded, $f(\cdot)$ is bounded, and because the set of orthogonal matrices is compact (see page 133) and $f(\cdot)$ is continuous, $f(\cdot)$ must attain its lower bound, say l. To simplify the notation, let

$$X = Q^{\mathrm{T}} \widetilde{A} Q.$$

Now suppose that there are r distinct eigenvalues of A (that is, the diagonal elements in C):

$$d_1 > \cdots > d_r.$$

We can write C as $\mathrm{diag}(d_i I_{m_i})$, where m_i is the multiplicity of d_i. We now partition $Q^{\mathrm{T}} \widetilde{A} Q$ to correspond to the partitioning of C represented by $\mathrm{diag}(d_i I_{m_i})$:

$$X = \begin{bmatrix} X_{11} & \cdots & X_{1r} \\ \vdots & \vdots & \vdots \\ X_{r1} & \cdots & X_{rr} \end{bmatrix}. \tag{8.9}$$

In this partitioning, the diagonal blocks, X_{ii}, are $m_i \times m_i$ symmetric matrices. The submatrix X_{ij}, is an $m_i \times m_j$ matrix.

We now proceed in two steps to show that in order for $f(Q)$ to attain its lower bound l, X must be diagonal. First we will show that when $f(Q) = l$, the submatrix X_{ij} in equation (8.9) must be null if $i \neq j$. To this end, let Q_∇ be such that $f(Q_\nabla) = l$, and assume the contrary regarding the corresponding $X_\nabla = Q_\nabla^{\mathrm{T}} \widetilde{A} Q_\nabla$; that is, assume that in some submatrix X_{ij_∇} where $i \neq j$, there is a nonzero element, say x_∇. We arrive at a contradiction by showing that in this case there is another X_0 of the form $Q_0^{\mathrm{T}} \widetilde{A} Q_0$, where Q_0 is orthogonal and such that $f(Q_0) < f(Q_\nabla)$.

To establish some useful notation, let p and q be the row and column, respectively, of X_∇ where this nonzero element x_∇ occurs; that is, $x_{pq} = x_\nabla \neq 0$ and $p \neq q$ because x_{pq} is in X_{ij_∇}. (Note the distinction between uppercase letters, which represent submatrices, and lowercase letters, which represent elements of matrices.) Also, because X_∇ is symmetric, $x_{qp} = x_\nabla$. Now let $a_\nabla = x_{pp}$ and $b_\nabla = x_{qq}$. We form Q_0 as $Q_\nabla R$, where R is an orthogonal rotation matrix of the form G_{pq} in equation (5.12) on page 239. We have, therefore, $\|Q_0^{\mathrm{T}} \widetilde{A} Q_0\|^2 = \|R^{\mathrm{T}} Q_\nabla^{\mathrm{T}} \widetilde{A} Q_\nabla R\|^2 = \|Q_\nabla^{\mathrm{T}} \widetilde{A} Q_\nabla\|^2$. Let a_0, b_0, and x_0 represent the elements of $Q_0^{\mathrm{T}} \widetilde{A} Q_0$ that correspond to a_∇, b_∇, and x_∇ in $Q_\nabla^{\mathrm{T}} \widetilde{A} Q_\nabla$.

From the definition of the Frobenius norm, we have

$$f(Q_0) - f(Q_\nabla) = 2(a_\nabla - a_0)d_i + 2(b_\nabla - b_0)d_j$$

because all other terms cancel. If the angle of rotation is θ, then

$$\begin{aligned} a_0 &= a_\nabla \cos^2\theta - 2x_\nabla \cos\theta \sin\theta + b_\nabla \sin^2\theta, \\ b_0 &= a_\nabla \sin^2\theta - 2x_\nabla \cos\theta \sin\theta + b_\nabla \cos^2\theta, \end{aligned}$$

and so for a function h of θ we can write

$$\begin{aligned} h(\theta) &= f(Q_0) - f(Q_\nabla) \\ &= 2d_i((a_\nabla - b_\nabla)\sin^2\theta + x_\nabla \sin 2\theta) + 2d_j((b_\nabla - b_0)\sin^2\theta - x_\nabla \sin 2\theta) \\ &= 2d_i((a_\nabla - b_\nabla) + 2d_j(b_\nabla - b_0))\sin^2\theta + 2x_\nabla(d_i - d_j)\sin 2\theta, \end{aligned}$$

and so

$$\frac{\mathrm{d}}{\mathrm{d}\theta}h(\theta) = 2d_i((a_\triangledown - b_\triangledown) + 2d_j(b_\triangledown - b_0))\sin 2\theta + 4x_\triangledown(d_i - d_j)\cos 2\theta.$$

The coefficient of $\cos 2\theta$, $4x_\triangledown(d_i - d_j)$, is nonzero because d_i and d_j are distinct, and $x_\triangledown$ is nonzero by the second assumption to be contradicted, and so the derivative at $\theta = 0$ is nonzero. Hence, by the proper choice of a direction of rotation (which effectively interchanges the roles of d_i and d_j), we can make $f(Q_0) - f(Q_\triangledown)$ positive or negative, showing that $f(Q_\triangledown)$ cannot be a minimum if some X_{ij} in equation (8.9) with $i \neq j$ is nonnull; that is, if $Q_\triangledown$ is a matrix such that $f(Q_\triangledown)$ is the minimum of $f(Q)$, then in the partition of $Q_\triangledown^{\mathrm{T}}\widetilde{A}Q_\triangledown$ only the diagonal submatrices $X_{ii_\triangledown}$ can be nonnull:

$$Q_\triangledown^{\mathrm{T}}\widetilde{A}Q_\triangledown = \mathrm{diag}(X_{11_\triangledown}, \ldots, X_{rr_\triangledown}).$$

The next step is to show that each $X_{ii_\triangledown}$ must be diagonal. Because it is symmetric, we can diagonalize it with an orthogonal matrix P_i as

$$P_i^{\mathrm{T}} X_{ii_\triangledown} P_i = G_i.$$

Now let P be the direct sum of the P_i and form

$$\begin{aligned} P^{\mathrm{T}}CP - P^{\mathrm{T}}Q_\triangledown^{\mathrm{T}}\widetilde{A}Q_\triangledown P &= \mathrm{diag}(d_1 I, \ldots, d_r I) - \mathrm{diag}(G_1, \ldots, G_r) \\ &= C - P^{\mathrm{T}}Q_\triangledown^{\mathrm{T}}\widetilde{A}Q_\triangledown P. \end{aligned}$$

Hence,

$$f(Q_\triangledown P) = f(Q_\triangledown),$$

and so the minimum occurs for a matrix $Q_\triangledown P$ that reduces $\widetilde{A}$ to a diagonal form. The elements of the G_i must be the $\tilde{c}_i$ in some order, so the minimum of $f(Q)$, which we have denoted by $f(Q_\triangledown)$, is $\sum(c_i - \tilde{c}_{p_i})^2$, where the p_i are a permutation of $1, \ldots, n$. As the final step, we show $p_i = i$. We begin with p_1. Suppose $p_1 \neq 1$ but $p_s = 1$; that is, $\tilde{c}_1 \geq \tilde{c}_{p_1}$. Interchange p_1 and p_s in the permutation. The change in the sum $\sum(c_i - \tilde{c}_{p_i})^2$ is

$$\begin{aligned} (c_1 - \tilde{c}_1)^2 + (c_s - \tilde{c}_{p_s})^2 - (c_1 - \tilde{c}_{p_s})^2 - (c_s - \tilde{c}_1)^2 &= -2(c_s - c_1)(\tilde{c}_{p_1} - \tilde{c}_1) \\ &\leq 0; \end{aligned}$$

that is, the interchange reduces the value of the sum. Similarly, we proceed through the p_i to p_n, getting $p_i = i$.

We have shown, therefore, that the minimum of $f(Q)$ is $\sum_{i=1}^n (c_i - \tilde{c}_i)^2$, where both sets of eigenvalues are ordered in nonincreasing value. From equation (8.7), which is $f(V)$, we have the inequality (8.6).

While an upper bound may be of more interest in the approximation problem, the lower bound in the Hoffman-Wielandt theorem gives us a measure of the goodness of the approximation of one matrix by another matrix. There are various extensions and other applications of the Hoffman-Wielandt theorem, see Chu (1991).

8.2.3 Normal Matrices

A real square matrix A is said to be *normal* if $A^{\mathrm{T}}A = AA^{\mathrm{T}}$. (In general, a square matrix is normal if $A^{\mathrm{H}}A = AA^{\mathrm{H}}$.) The Gramian matrix formed from a normal matrix is the same as the Gramian formed from the transpose (or conjugate transpose) of the matrix. Normal matrices include symmetric (and Hermitian), skew symmetric (and Hermitian), square orthogonal (and unitary) matrices, and circulant matrices. The identity is also obviously a normal matrix.

There are a number of interesting properties possessed by a normal matrix, but the most important property is that it can be diagonalized by a unitary matrix. Recall from page 154 that a matrix can be orthogonally diagonalized if and only if the matrix is symmetric. Not all normal matrices can be orthogonally diagonalized, but all can be diagonalized by a unitary matrix ("unitarily diagonalized"). In fact, a matrix can be unitarily diagonalized if and only if the matrix is normal. (This is the reason the word "normal" is used to describe these matrices; and a matrix that is unitarily diagonalizable is an alternate, and more meaningful way of defining a normal matrix.)

It is easy to see that a matrix A is unitarily diagonalizable if and only if $A^{\mathrm{H}}A = AA^{\mathrm{H}}$ (and for real A, $A^{\mathrm{T}}A = AA^{\mathrm{T}}$ implies $A^{\mathrm{H}}A = AA^{\mathrm{H}}$).

First suppose A is unitarily diagonalizable. Let $A = P\,D\,P^{\mathrm{H}}$, where P is unitary and D is diagonal. In that case, the elements of D are the eigenvalues of A, as we see by considering each column in AP. Now consider AA^{H}:

$$\begin{aligned} AA^{\mathrm{H}} &= P\,D\,P^{\mathrm{H}}P\,D^{\mathrm{H}}\,P^{\mathrm{H}} = P\,D\,D^{\mathrm{H}}\,P^{\mathrm{H}} = \\ &\quad P\,D^{\mathrm{H}}D\,P^{\mathrm{H}} = P\,D^{\mathrm{H}}\,P^{\mathrm{H}}P\,D\,P^{\mathrm{H}} = A^{\mathrm{H}}A. \end{aligned}$$

Next, suppose $A^{\mathrm{H}}A = AA^{\mathrm{H}}$. To see that A is unitarily diagonalizable, form the Schur factorization, $A = UTU^{\mathrm{H}}$ (see Sect. 3.8.7 on page 147). We have

$$A^{\mathrm{H}}A = UT^{\mathrm{H}}U^{\mathrm{H}}UTU^{\mathrm{H}} = UT^{\mathrm{H}}TU^{\mathrm{H}}$$

and

$$AA^{\mathrm{H}} = UTU^{\mathrm{H}}UT^{\mathrm{H}}U^{\mathrm{H}} = UTT^{\mathrm{H}}U^{\mathrm{H}}.$$

Now under the assumption that $A^{\mathrm{H}}A = AA^{\mathrm{H}}$,

$$UT^{\mathrm{H}}TU^{\mathrm{H}} = UTT^{\mathrm{H}}U^{\mathrm{H}},$$

which implies $T^{\mathrm{H}}T = TT^{\mathrm{H}}$, in which case

$$|t_{ii}|^2 = \sum_{j=1}^{n} |t_{ij}|^2,$$

that is, $t_{ij} = 0$ unless $j = i$. We conclude that T is diagonal, and hence, the Schur factorization, $A = UTU^{\mathrm{H}}$ a is unitary diagonalization of A, so A is unitarily diagonalizable.

Spectral methods, based on the unitary diagonalization, are useful in many areas of applied mathematics. The spectra of nonnormal matrices, however, are quite different (see Trefethen and Embree (2005)).

8.3 Nonnegative Definite Matrices: Cholesky Factorization

We defined nonnegative definite and positive definite matrices on page 91, and discussed some of their properties, particularly in Sect. 3.8.11. We have seen that these matrices have useful factorizations, in particular, the square root and the Cholesky factorization. In this section, we recall those definitions, properties, and factorizations.

A symmetric matrix A such that any quadratic form involving the matrix is nonnegative is called a *nonnegative definite matrix*. That is, a symmetric matrix A is a nonnegative definite matrix if, for any (conformable) vector x,

$$x^{\mathrm{T}}Ax \geq 0. \tag{8.10}$$

(We remind the reader that there is a related term, *positive semidefinite matrix*, that is not used consistently in the literature. We will generally avoid the term "semidefinite".)

We denote the fact that A is nonnegative definite by

$$A \succeq 0. \tag{8.11}$$

(Some people use the notation $A \geq 0$ to denote a nonnegative definite matrix, but we have decided to use this notation to indicate that each element of A is nonnegative; see page 64.)

There are several properties that follow immediately from the definition.

- The sum of two (conformable) nonnegative matrices is nonnegative definite.
- All diagonal elements of a nonnegative definite matrix are nonnegative. Hence, if A is nonnegative definite, $\mathrm{tr}(A) \geq 0$.
- Any square submatrix whose principal diagonal is a subset of the principal diagonal of a nonnegative definite matrix is nonnegative definite. In particular, any square principal submatrix of a nonnegative definite matrix is nonnegative definite.

It is easy to show that the latter two facts follow from the definition by considering a vector x with zeros in all positions except those corresponding to the submatrix in question. For example, to see that all diagonal elements of a nonnegative definite matrix are nonnegative, assume the (i, i) element is negative, and then consider the vector x to consist of all zeros except for a 1 in the i^{th} position. It is easy to see that the quadratic form is negative, so the assumption that the (i, i) element is negative leads to a contradiction.

- A diagonal matrix is nonnegative definite if and only if all of the diagonal elements are nonnegative.

This must be true because a quadratic form in a diagonal matrix is the sum of the diagonal elements times the squares of the elements of the vector.

We can also form other submatrices that are nonnegative definite:

- If A is nonnegative definite, then $A_{-(i_1,\ldots,i_k)(i_1,\ldots,i_k)}$ is nonnegative definite. (See page 599 for notation.)

Again, we can see this by selecting an x in the defining inequality (8.10) consisting of 1s in the positions corresponding to the rows and columns of A that are retained and 0s elsewhere.

By considering $x^{\mathrm{T}}C^{\mathrm{T}}ACx$ and $y = Cx$, we see that

- if A is nonnegative definite, and C is conformable for the multiplication, then $C^{\mathrm{T}}AC$ is nonnegative definite.

From equation (3.252) and the fact that the determinant of a product is the product of the determinants, we have that

- the determinant of a nonnegative definite matrix is nonnegative.

Finally, for the nonnegative definite matrix A, we have

$$a_{ij}^2 \leq a_{ii}a_{jj}, \tag{8.12}$$

as we see from the definition $x^{\mathrm{T}}Ax \geq 0$ and choosing the vector x to have a variable y in position i, a 1 in position j, and 0s in all other positions. For a symmetric matrix A, this yields the quadratic $a_{ii}y^2 + 2a_{ij}y + a_{jj}$. If this quadratic is to be nonnegative for all y, then the discriminant $4a_{ij}^2 - 4a_{ii}a_{jj}$ must be nonpositive; that is, inequality (8.12) must be true.

8.3.1 Eigenvalues of Nonnegative Definite Matrices

We have seen on page 159 that a real symmetric matrix is nonnegative (positive) definite if and only if all of its eigenvalues are nonnegative (positive).

This fact allows a generalization of the statement above: a triangular matrix is nonnegative (positive) definite if and only if all of the diagonal elements are nonnegative (positive).

8.3.2 The Square Root and the Cholesky Factorization

Two important factorizations of nonnegative definite matrices are the square root,

$$A = (A^{\frac{1}{2}})^2, \tag{8.13}$$

discussed in Sect. 5.9.1, and the Cholesky factorization,

$$A = T^{\mathrm{T}}T, \tag{8.14}$$

discussed in Sect. 5.9.2. If T is as in equation (8.14), the symmetric matrix $T + T^{\mathrm{T}}$ is also nonnegative definite, or positive definite if A is.

The square root matrix is used often in theoretical developments, such as Exercise 4.7b for example, but the Cholesky factor is more useful in practice. The Cholesky factorization also has a prominent role in multivariate analysis, where it appears in the *Bartlett decomposition*. If W is a Wishart matrix with variance-covariance matrix Σ (see Exercise 4.12 on page 224), then the Bartlett decomposition of W is

$$W = (TU)^{\mathrm{T}}TU,$$

where U is the Cholesky factor of Σ, and T is an upper triangular matrix with positive diagonal elements. The diagonal elements of T have independent chi-squared distributions and the off-diagonal elements of T have independent standard normal distributions.

8.3.3 The Convex Cone of Nonnegative Definite Matrices

The class of all $n \times n$ nonnegative definite matrices is a *cone* because if X is a nonnegative definite matrix and $a > 0$, then aX is a nonnegative definite matrix (see page 43). Furthermore, it is *convex cone* in $\mathbb{R}^{n\times n}$, because if X_1 and X_2 are $n \times n$ nonnegative definite matrices and $a, b \geq 0$, then $aX_1 + bX_2$ is nonnegative definite so long as either $a > 0$ or $b > 0$.

This class is not closed under Cayley multiplication (that is, in particular, it is not a group with respect to that operation). The product of two nonnegative definite matrices might not even be symmetric.

The convex cone of nonnegative definite matrices is an important object in a common optimization problem called convex cone programming ("programming" here means "optimization"). A special case of convex cone programming is called "semidefinite programming", or "SDP" (where "semidefinite" comes from the alternative terminology for nonnegative definite). The canonical SDP problem for given $n \times n$ symmetric matrices C and A_i, and real numbers b_i, for $i = 1, \ldots, m$, is

$$\begin{aligned} &\text{minimize } \langle C, X\rangle \\ &\text{subject to } \langle A_i, X\rangle = b_i \quad i = 1, \ldots, m \\ &\qquad\qquad X \succeq 0. \end{aligned}$$

The notation $\langle C, X\rangle$ here means the matrix inner product (see page 97). SDP includes linear programming as a simple special case, in which C and X are vectors. See Vandenberghe and Boyd (1996) for further discussion of the SDP problem and applications.

8.4 Positive Definite Matrices

An important class of nonnegative definite matrices are those that satisfy strict inequalities in the definition involving $x^{\mathrm{T}}Ax$. These matrices are called

positive definite matrices and they have all of the properties discussed above for nonnegative definite matrices as well as some additional useful properties.

A symmetric matrix A is called a *positive definite matrix* if, for any (conformable) vector $x \neq 0$, the quadratic form is positive; that is,

$$x^{\mathrm{T}}Ax > 0. \tag{8.15}$$

We denote the fact that A is positive definite by

$$A \succ 0. \tag{8.16}$$

(Some people use the notation $A > 0$ to denote a positive definite matrix, but we have decided to use this notation to indicate that each element of A is positive.)

The properties of nonnegative definite matrices noted above hold also for positive definite matrices, generally with strict inequalities. It is obvious that all diagonal elements of a positive definite matrix are positive. Hence, if A is positive definite, $\mathrm{tr}(A) > 0$. Furthermore, as above and for the same reasons, if A is positive definite, then $A_{-(i_1,\ldots,i_k)(i_1,\ldots,i_k)}$ is positive definite. In particular,

- Any square principal submatrix of a positive definite matrix is positive definite.

Because a quadratic form in a diagonal matrix is the sum of the diagonal elements times the squares of the elements of the vector, a diagonal matrix is positive definite if and only if all of the diagonal elements are positive.

From equation (3.252) and the fact that the determinant of a product is the product of the determinants, we have

- The determinant of a positive definite matrix is positive.
- If A is positive definite,
$$a_{ij}^2 < a_{ii}a_{jj}, \tag{8.17}$$
which we see using the same argument as for inequality (8.12).

We have a slightly stronger statement regarding sums involving positive definite matrices than what we could conclude about nonnegative definite matrices:

- The sum of a positive definite matrix and a (conformable) nonnegative definite matrix is positive definite.

That is,

$$x^{\mathrm{T}}Ax > 0\ \forall x \neq 0 \quad \text{and} \quad y^{\mathrm{T}}By \geq 0\ \forall y \Longrightarrow z^{\mathrm{T}}(A+B)z > 0\ \forall z \neq 0. \tag{8.18}$$

- A positive definite matrix is necessarily nonsingular. (We see this from the fact that no nonzero combination of the columns, or rows, can be 0.) Furthermore, if A is positive definite, then A^{-1} is positive definite. (We showed this is Sect. 3.8.11, but we can see it in another way: because for any $y \neq 0$ and $x = A^{-1}y$, we have $y^{\mathrm{T}}A^{-1}y = x^{\mathrm{T}}y = x^{\mathrm{T}}Ax > 0$.)

- A (strictly) diagonally dominant symmetric matrix with positive diagonals is positive definite. The proof of this is Exercise 8.3.
- A positive definite matrix is orthogonally diagonalizable.
- A positive definite matrix has a square root.
- A positive definite matrix Cholesky factorization.

We cannot conclude that the product of two positive definite matrices is positive definite, but we do have the useful fact:

- If A is positive definite, and C is of full rank and conformable for the multiplication AC, then $C^{\mathrm{T}}AC$ is positive definite (see page 114).

We have seen from the definition of positive definiteness and the distribution of multiplication over addition that the sum of a positive definite matrix and a nonnegative definite matrix is positive definite. We can define an ordinal relationship between positive definite and nonnegative definite matrices of the same size. If A is positive definite and B is nonnegative definite of the same size, we say A is *strictly greater than* B and write

$$A \succ B \tag{8.19}$$

if $A - B$ is positive definite; that is, if $A - B \succ 0$.

We can form a *partial ordering* of nonnegative definite matrices of the same order based on this additive property. We say A is *greater than* B and write

$$A \succeq B \tag{8.20}$$

if $A - B$ is either the 0 matrix or is nonnegative definite; that is, if $A - B \succeq 0$ (see Exercise 8.2a). The "strictly greater than" relation implies the "greater than" relation. These relations are *partial* in the sense that they do not apply to all pairs of nonnegative matrices; that is, there are pairs of matrices A and B for which neither $A \succeq B$ nor $B \succeq A$.

If $A \succ B$, we also write $B \prec A$; and if $A \succeq B$, we may write $B \preceq A$.

8.4.1 Leading Principal Submatrices of Positive Definite Matrices

A sufficient condition for a symmetric matrix to be positive definite is that the determinant of each of the leading principal submatrices be positive. To see this, first let the $n \times n$ symmetric matrix A be partitioned as

$$A = \begin{bmatrix} A_{n-1} & a \\ a^{\mathrm{T}} & a_{nn} \end{bmatrix},$$

and assume that A_{n-1} is positive definite and that $|A| > 0$. (This is not the same notation that we have used for these submatrices, but the notation is convenient in this context.) From equation (3.192) on page 123,

$$|A| = |A_{n-1}|(a_{nn} - a^{\mathrm{T}} A_{n-1}^{-1} a).$$

Because A_{n-1} is positive definite, $|A_{n-1}| > 0$, and so $(a_{nn} - a^{\mathrm{T}} A_{n-1}^{-1} a) > 0$; hence, the 1×1 matrix $(a_{nn} - a^{\mathrm{T}} A_{n-1}^{-1} a)$ is positive definite. That any matrix whose leading principal submatrices have positive determinants is positive definite follows from this by induction, beginning with a 2×2 matrix.

8.4.2 The Convex Cone of Positive Definite Matrices

The class of all $n \times n$ positive definite matrices is a *cone* because if X is a positive definite matrix and $a > 0$, then aX is a positive definite matrix (see page 43). Furthermore, this class is a *convex cone* in $\mathbb{R}^{n \times n}$ because if X_1 and X_2 are $n \times n$ positive definite matrices and $a, b \geq 0$, then $aX_1 + bX_2$ is positive definite so long as either $a \neq 0$ or $b \neq 0$.

As with the cone of nonnegative definite matrices this class is not closed under Cayley multiplication.

8.4.3 Inequalities Involving Positive Definite Matrices

Quadratic forms of positive definite matrices and nonnegative matrices occur often in data analysis. There are several useful inequalities involving such quadratic forms.

On page 156, we showed that if $x \neq 0$, for any symmetric matrix A with eigenvalues c_i,

$$\frac{x^{\mathrm{T}} A x}{x^{\mathrm{T}} x} \leq \max\{c_i\}. \tag{8.21}$$

If A is nonnegative definite, by our convention of labeling the eigenvalues, we have $\max\{c_i\} = c_1$. If the rank of A is r, the minimum nonzero eigenvalue is denoted c_r. Letting the eigenvectors associated with $c_1, \ldots, c_r$ be $v_1, \ldots, v_r$ (and recalling that these choices may be arbitrary in the case where some eigenvalues are not simple), by an argument similar to that used on page 156, we have that if A is nonnegative definite of rank r,

$$\frac{v_i^{\mathrm{T}} A v_i}{v_i^{\mathrm{T}} v_i} \geq c_r, \tag{8.22}$$

for $1 \leq i \leq r$.

If A is positive definite and x and y are conformable nonzero vectors, we see that

$$x^{\mathrm{T}} A^{-1} x \geq \frac{(y^{\mathrm{T}} x)^2}{y^{\mathrm{T}} A y} \tag{8.23}$$

by using the same argument as used in establishing the Cauchy-Schwarz inequality (2.26). We first obtain the Cholesky factor T of A (which is, of course, of full rank) and then observe that for every real number t

$$\left(tTy + T^{-\mathrm{T}}x\right)^{\mathrm{T}}\left(tTy + T^{-\mathrm{T}}x\right) \geq 0,$$

and hence the discriminant of the quadratic equation in t must be nonnegative:

$$4\left((Ty)^{\mathrm{T}}T^{-\mathrm{T}}x\right)^2 - 4\left(T^{-\mathrm{T}}x\right)^{\mathrm{T}}\left(T^{-\mathrm{T}} - x\right)(Ty)^{\mathrm{T}}Ty \leq 0.$$

The inequality (8.23) is used in constructing Scheffé simultaneous confidence intervals in linear models.

The Kantorovich inequality for positive numbers has an immediate extension to an inequality that involves positive definite matrices. The Kantorovich inequality, which finds many uses in optimization problems, states, for positive numbers $c_1 \geq c_2 \geq \cdots \geq c_n$ and nonnegative numbers $y_1, \ldots, y_n$ such that $\sum y_i = 1$, that

$$\left(\sum_{i=1}^{n} y_i c_i\right)\left(\sum_{i=1}^{n} y_i c_i^{-1}\right) \leq \frac{(c_1 + c_2)^2}{4c_1 c_2}.$$

Now let A be an $n \times n$ positive definite matrix with eigenvalues $c_1 \geq c_2 \geq \cdots \geq c_n > 0$. We substitute x^2 for y, thus removing the nonnegativity restriction, and incorporate the restriction on the sum directly into the inequality. Then, using the similar canonical factorization of A and A^{-1}, we have

$$\frac{\left(x^{\mathrm{T}}Ax\right)\left(x^{\mathrm{T}}A^{-1}x\right)}{(x^{\mathrm{T}}x)^2} \leq \frac{(c_1 + c_n)^2}{4c_1 c_n}. \tag{8.24}$$

This Kantorovich matrix inequality likewise has applications in optimization; in particular, for assessing convergence of iterative algorithms.

The left-hand side of the Kantorovich matrix inequality also has a lower bound,

$$\frac{\left(x^{\mathrm{T}}Ax\right)\left(x^{\mathrm{T}}A^{-1}x\right)}{(x^{\mathrm{T}}x)^2} \geq 1, \tag{8.25}$$

which can be seen in a variety of ways, perhaps most easily by using the inequality (8.23). (You were asked to prove this directly in Exercise 3.31.)

All of the inequalities (8.21) through (8.25) are sharp. We know that (8.21) and (8.22) are sharp by using the appropriate eigenvectors. We can see the others are sharp by using $A = I$.

There are several variations on these inequalities and other similar inequalities that are reviewed by Marshall and Olkin (1990) and Liu and Neudecker (1996).

8.5 Idempotent and Projection Matrices

An important class of matrices are those that, like the identity, have the property that raising them to a power leaves them unchanged. A matrix A such that

$$AA = A \tag{8.26}$$

is called an *idempotent matrix*. An idempotent matrix is square, and it is either singular or the identity matrix. (It must be square in order to be conformable for the indicated multiplication. If it is not singular, we have $A = (A^{-1}A)A = A^{-1}(AA) = A^{-1}A = I$; hence, an idempotent matrix is either singular or the identity matrix.)

From the definition, it is clear that an idempotent matrix is its own Drazin inverse: $A^{\mathrm{D}} = A$ (see page 129).

An idempotent matrix that is symmetric is called a *projection matrix*.

8.5.1 Idempotent Matrices

Many matrices encountered in the statistical analysis of linear models are idempotent. One such matrix is $X^{-}X$ (see page 124 and Sect. 9.3.2). This matrix exists for any $n \times m$ matrix X, and it is square. (It is $m \times m$.)

Because the eigenvalues of A^2 are the squares of the eigenvalues of A, all eigenvalues of an idempotent matrix must be either 0 or 1.

Any vector in the column space of an idempotent matrix A is an eigenvector of A. (This follows immediately from $AA = A$.) More generally, if x and y are vectors in $\mathrm{span}(A)$ and a is a scalar, then

$$A(ax + y) = ax + y. \tag{8.27}$$

(To see this, we merely represent x and y as linear combinations of columns (or rows) of A and substitute in the equation.)

The number of eigenvalues that are 1 is the rank of an idempotent matrix. (Exercise 8.4 asks why this is the case.) We therefore have, for an idempotent matrix A,

$$\mathrm{tr}(A) = \mathrm{rank}(A). \tag{8.28}$$

Because the eigenvalues of an idempotent matrix are either 0 or 1, a symmetric idempotent matrix is nonnegative definite.

If A is idempotent and $n \times n$, then

$$\mathrm{rank}(I - A) = n - \mathrm{rank}(A). \tag{8.29}$$

We showed this in equation (3.200) on page 125. (Although there we were considering the special matrix $A^{-}A$, the only properties used were the idempotency of $A^{-}A$ and the fact that $\mathrm{rank}(A^{-}A) = \mathrm{rank}(A)$.)

Equation (8.29) together with the diagonalizability theorem (equation (3.248)) implies that an idempotent matrix is diagonalizable.

If A is idempotent and V is an orthogonal matrix of the same size, then $V^{\mathrm{T}}AV$ is idempotent (whether or not V is a matrix that diagonalizes A) because

$$(V^{\mathrm{T}}AV)(V^{\mathrm{T}}AV) = V^{\mathrm{T}}AAV = V^{\mathrm{T}}AV. \tag{8.30}$$

If A is idempotent, then $(I - A)$ is also idempotent, as we see by multiplication. This fact and equation (8.29) have generalizations for sums of idempotent matrices that are parts of Cochran's theorem, which we consider below.

Although if A is idempotent so $(I - A)$ is also idempotent and hence is not of full rank (unless $A = 0$), for any scalar $a \neq -1$, $(I + aA)$ is of full rank, and

$$(I + aA)^{-1} = I - \frac{a}{a+1}A, \tag{8.31}$$

as we see by multiplication.

On page 146, we saw that similar matrices are equivalent (have the same rank). For idempotent matrices, we have the converse: idempotent matrices of the same rank (and size) are similar (see Exercise 8.5).

If A_1 and A_2 are matrices conformable for addition, then $A_1 + A_2$ is idempotent if and only if $A_1A_2 = A_2A_1 = 0$. It is easy to see that this condition is sufficient by multiplication:

$$(A_1 + A_2)(A_1 + A_2) = A_1A_1 + A_1A_2 + A_2A_1 + A_2A_2 = A_1 + A_2.$$

To see that it is necessary, we first observe from the expansion above that $A_1 + A_2$ is idempotent only if $A_1A_2 + A_2A_1 = 0$. Multiplying this necessary condition on the left by A_1 yields

$$A_1A_1A_2 + A_1A_2A_1 = A_1A_2 + A_1A_2A_1 = 0,$$

and multiplying on the right by A_1 yields

$$A_1A_2A_1 + A_2A_1A_1 = A_1A_2A_1 + A_2A_1 = 0.$$

Subtracting these two equations yields

$$A_1A_2 = A_2A_1,$$

and since $A_1A_2 + A_2A_1 = 0$, we must have $A_1A_2 = A_2A_1 = 0$.

8.5.1.1 Symmetric Idempotent Matrices

Many of the idempotent matrices in statistical applications are symmetric, and such matrices have some useful properties.

Because the eigenvalues of an idempotent matrix are either 0 or 1, the spectral decomposition of a symmetric idempotent matrix A can be written as

$$V^{\mathrm{T}}AV = \mathrm{diag}(I_r, 0), \tag{8.32}$$

where V is a square orthogonal matrix and $r = \mathrm{rank}(A)$. (This is from equation (3.253) on page 154.)

For symmetric matrices, there is a converse to the fact that all eigenvalues of an idempotent matrix are either 0 or 1. If A is a symmetric matrix all of whose eigenvalues are either 0 or 1, then A is idempotent. We see this from the spectral decomposition of A, $A = V\text{diag}(I_r, 0)V^{\mathrm{T}}$, and, with $C = \text{diag}(I_r, 0)$, by observing

$$AA = VCV^{\mathrm{T}}VCV^{\mathrm{T}} = VCCV^{\mathrm{T}} = VCV^{\mathrm{T}} = A,$$

because the diagonal matrix of eigenvalues C contains only 0s and 1s.

If A is symmetric and p is any positive integer,

$$A^{p+1} = A^p \Longrightarrow A \text{ is idempotent.} \tag{8.33}$$

This follows by considering the eigenvalues of A, $c_1, \ldots, c_n$. The eigenvalues of A^{p+1} are $c_1^{p+1}, \ldots, c_n^{p+1}$ and the eigenvalues of A^p are $c_1^p, \ldots, c_n^p$, but since $A^{p+1} = A^p$, it must be the case that $c_i^{p+1} = c_i^p$ for each $i = 1, \ldots, n$. The only way this is possible is for each eigenvalue to be 0 or 1, and in this case the symmetric matrix must be idempotent.

There are bounds on the elements of a symmetric idempotent matrix. Because A is symmetric and $A^{\mathrm{T}}A = A$,

$$a_{ii} = \sum_{j=1}^{n} a_{ij}^2; \tag{8.34}$$

hence, $0 \leq a_{ii}$. Rearranging equation (8.34), we have

$$a_{ii} = a_{ii}^2 + \sum_{j \neq i} a_{ij}^2, \tag{8.35}$$

so $a_{ii}^2 \leq a_{ii}$ or $0 \leq a_{ii}(1 - a_{ii})$; that is, $a_{ii} \leq 1$. Now, if $a_{ii} = 0$ or $a_{ii} = 1$, then equation (8.35) implies

$$\sum_{j \neq i} a_{ij}^2 = 0,$$

and the only way this can happen is if $a_{ij} = 0$ for all $j \neq i$. So, in summary, if A is an $n \times n$ symmetric idempotent matrix, then

$$0 \leq a_{ii} \leq 1 \text{ for } i = 1, \ldots, m, \tag{8.36}$$

and

$$\text{if } a_{ii} = 0 \text{ or } a_{ii} = 1, \text{ then } a_{ij} = a_{ji} = 0 \text{ for all } j \neq i. \tag{8.37}$$

8.5.1.2 Cochran's Theorem

There are various facts that are sometimes called *Cochran's theorem*. The simplest one concerns k symmetric idempotent $n \times n$ matrices, $A_1, \ldots, A_k$, such that

$$I_n = A_1 + \cdots + A_k. \tag{8.38}$$

Under these conditions, we have

$$A_i A_j = 0 \text{ for all } i \neq j. \tag{8.39}$$

We see this by the following argument. For an arbitrary j, as in equation (8.32), for some matrix V, we have

$$V^{\mathrm{T}} A_j V = \mathrm{diag}(I_r, 0),$$

where $r = \mathrm{rank}(A_j)$. Now

$$\begin{aligned} I_n &= V^{\mathrm{T}} I_n V \\ &= \sum_{i=1}^{k} V^{\mathrm{T}} A_i V \\ &= \mathrm{diag}(I_r, 0) + \sum_{i \neq j} V^{\mathrm{T}} A_i V, \end{aligned}$$

which implies

$$\sum_{i \neq j} V^{\mathrm{T}} A_i V = \mathrm{diag}(0, I_{n-r}). \tag{8.40}$$

Now, from equation (8.30), for each i, $V^{\mathrm{T}} A_i V$ is idempotent, and so from equation (8.36) the diagonal elements are all nonnegative, and hence equation (8.40) implies that for each $i \neq j$, the first r diagonal elements are 0. Furthermore, since these diagonal elements are 0, equation (8.37) implies that all elements in the first r rows and columns are 0. We have, therefore, for each $i \neq j$,

$$V^{\mathrm{T}} A_i V = \mathrm{diag}(0, B_i)$$

for some $(n - r) \times (n - r)$ symmetric idempotent matrix B_i. Now, for any $i \neq j$, consider $A_i A_j$ and form $V^{\mathrm{T}} A_i A_j V$. We have

$$\begin{aligned} V^{\mathrm{T}} A_i A_j V &= (V^{\mathrm{T}} A_i V)(V^{\mathrm{T}} A_j V) \\ &= \mathrm{diag}(0, B_i)\mathrm{diag}(I_r, 0) \\ &= 0. \end{aligned}$$

Because V is nonsingular, this implies the desired conclusion; that is, that $A_i A_j = 0$ for any $i \neq j$.

We can now extend this result to an idempotent matrix in place of I; that is, for an idempotent matrix A with $A = A_1 + \cdots + A_k$. Rather than stating it simply as in equation (8.39), however, we will state the implications differently.

Let $A_1, \ldots, A_k$ be $n \times n$ symmetric matrices and let

$$A = A_1 + \cdots + A_k. \tag{8.41}$$

Then any two of the following conditions imply the third one:

(a). A is idempotent.
(b). A_i is idempotent for $i = 1, \ldots, k$.
(c). $A_i A_j = 0$ for all $i \neq j$.

This is also called *Cochran's theorem*. (The theorem also applies to non-symmetric matrices if condition (c) is augmented with the requirement that $\text{rank}(A_i^2) = \text{rank}(A_i)$ for all i. We will restrict our attention to symmetric matrices, however, because in most applications of these results, the matrices are symmetric.)

First, if we assume properties (a) and (b), we can show that property (c) follows using an argument similar to that used to establish equation (8.39) for the special case $A = I$. The formal steps are left as an exercise.

Now, let us assume properties (b) and (c) and show that property (a) holds. With properties (b) and (c), we have

$$\begin{aligned} AA &= (A_1 + \cdots + A_k)(A_1 + \cdots + A_k) \\ &= \sum_{i=1}^{k} A_i A_i + \sum_{i \neq j} \sum_{j=1}^{k} A_i A_j \\ &= \sum_{i=1}^{k} A_i \\ &= A. \end{aligned}$$

Hence, we have property (a); that is, A is idempotent.

Finally, let us assume properties (a) and (c). Property (b) follows immediately from

$$A_i^2 = A_i A_i = A_i A = A_i A A = A_i^2 A = A_i^3$$

and the implication (8.33).

Any two of the properties (a) through (c) also imply a fourth property for $A = A_1 + \cdots + A_k$ when the A_i are symmetric:

(d). $\text{rank}(A) = \text{rank}(A_1) + \cdots + \text{rank}(A_k)$.

We first note that any two of properties (a) through (c) imply the third one, so we will just use properties (a) and (b). Property (a) gives

$$\text{rank}(A) = \text{tr}(A) = \text{tr}(A_1 + \cdots + A_k) = \text{tr}(A_1) + \cdots + \text{tr}(A_k),$$

and property (b) states that the latter expression is $\text{rank}(A_1) + \cdots + \text{rank}(A_k)$, thus yielding property (d).

There is also a partial converse: properties (a) and (d) imply the other properties.

One of the most important special cases of Cochran's theorem is when $A = I$ in the sum (8.41):

$$I_n = A_1 + \cdots + A_k.$$

The identity matrix is idempotent, so if $\text{rank}(A_1) + \cdots + \text{rank}(A_k) = n$, all the properties above hold.

The most important statistical application of Cochran's theorem is for the distribution of quadratic forms of normally distributed random vectors. These distribution results are also called Cochran's theorem. We briefly discuss it in Sect. 9.2.3.

8.5.2 Projection Matrices: Symmetric Idempotent Matrices

For a given vector space $\mathcal{V}$, a symmetric idempotent matrix A whose columns span $\mathcal{V}$ is said to be a *projection matrix* onto $\mathcal{V}$; in other words, a matrix A is a projection matrix onto $\text{span}(A)$ if and only if A is symmetric and idempotent. (Some authors do not require a projection matrix to be symmetric. In that case, the terms "idempotent" and "projection" are synonymous.)

It is easy to see that, for any vector x, if A is a projection matrix onto $\mathcal{V}$, the vector Ax is in $\mathcal{V}$, and the vector $x - Ax$ is in $\mathcal{V}^\perp$ (the vectors Ax and $x - Ax$ are orthogonal). For this reason, a projection matrix is sometimes called an "orthogonal projection matrix". Note that an orthogonal projection matrix is not an orthogonal matrix, however, unless it is the identity matrix. Stating this in alternative notation, if A is a projection matrix and $A \in \mathbb{R}^{n\times n}$, then A maps $\mathbb{R}^n$ onto $\mathcal{V}(A)$ and $I - A$ is also a projection matrix (called the *complementary projection matrix* of A), and it maps $\mathbb{R}^n$ onto the orthogonal complement, $\mathcal{N}(A)$. These spaces are such that $\mathcal{V}(A) \oplus \mathcal{N}(A) = \mathbb{R}^n$.

In this text, we use the term "projection" to mean "orthogonal projection", but we should note that in some literature "projection" can include "oblique projection". In the less restrictive definition, for vector spaces $\mathcal{V}$, $\mathcal{X}$, and $\mathcal{Y}$, if $\mathcal{V} = \mathcal{X} \oplus \mathcal{Y}$ and $v = x + y$ with $x \in \mathcal{X}$ and $y \in \mathcal{Y}$, then the vector x is called the projection of v onto $\mathcal{X}$ along $\mathcal{Y}$. In this text, to use the unqualified term "projection", we require that $\mathcal{X}$ and $\mathcal{Y}$ be orthogonal; if they are not, then we call x the *oblique projection* of v onto $\mathcal{X}$ along $\mathcal{Y}$. The choice of the more restrictive definition is because of the overwhelming importance of orthogonal projections in statistical applications. The restriction is also consistent with the definition in equation (2.51) of the projection of a vector onto another vector (as opposed to the projection onto a vector space).

Because a projection matrix is idempotent, the matrix projects any of its columns onto itself, and of course it projects the full matrix onto itself: $AA = A$ (see equation (8.27)).

If x is a general vector in $\mathbb{R}^n$, that is, if x has order n and belongs to an n-dimensional space, and A is a projection matrix of rank $r \leq n$, then Ax has order n and belongs to $\text{span}(A)$, which is an r-dimensional space. Thus, we say projections are *dimension reductions*.

Useful projection matrices often encountered in statistical linear models are X^+X and XX^+. (Recall that for any generalized inverse X^-X is an idempotent matrix.) The matrix X^+ exists for any $n \times m$ matrix X, and X^+X is square ($m \times m$) and symmetric.

8.5.2.1 Projections onto Linear Combinations of Vectors

On page 36, we gave the projection of a vector y onto a vector x as

$$\frac{x^{\mathrm{T}}y}{x^{\mathrm{T}}x}x.$$

The projection matrix to accomplish this is the "outer/inner products matrix",

$$\frac{1}{x^{\mathrm{T}}x}xx^{\mathrm{T}}. \tag{8.42}$$

The outer/inner products matrix has rank 1. It is useful in a variety of matrix transformations. If x is normalized, the projection matrix for projecting a vector on x is just xx^{T}. The projection matrix for projecting a vector onto a unit vector e_i is $e_ie_i^{\mathrm{T}}$, and $e_ie_i^{\mathrm{T}}y = (0, \ldots, y_i, \ldots, 0)$.

This idea can be used to project y onto the plane formed by two vectors, x_1 and x_2, by forming a projection matrix in a similar manner and replacing x in equation (8.42) with the matrix $X = [x_1|x_2]$. On page 409, we will view linear regression fitting as a projection onto the space spanned by the independent variables.

The angle between vectors we defined on page 37 can be generalized to the angle between a vector and a plane or any linear subspace by defining it as the angle between the vector and the projection of the vector onto the subspace. By applying the definition (2.54) to the projection, we see that the *angle* θ between the vector y and the subspace spanned by the columns of a projection matrix A is determined by the cosine

$$\cos(\theta) = \frac{y^{\mathrm{T}}Ay}{y^{\mathrm{T}}y}. \tag{8.43}$$

8.6 Special Matrices Occurring in Data Analysis

Some of the most useful applications of matrices are in the representation of observational data, as in Fig. 8.1 on page 330. If the data are represented as real numbers, the array is a matrix, say X. The rows of the $n \times m$ data matrix X are "observations" and correspond to a vector of measurements on a single *observational unit*, and the columns of X correspond to n measurements of a single variable or feature. In data analysis we may form various association

matrices that measure relationships among the variables or the observations that correspond to the columns or the rows of X. Many summary statistics arise from a matrix of the form $X^{\mathrm{T}}X$. (If the data in X are incomplete—that is, if some elements are missing—problems may arise in the analysis. We discuss some of these issues in Sect. 9.5.6.)

8.6.1 Gramian Matrices

A (real) matrix A such that for some (real) matrix B, $A = B^{\mathrm{T}}B$, is called a *Gramian matrix*. Any nonnegative definite matrix is Gramian (from equation (8.14) and Sect. 5.9.2 on page 256). (This is not a definition of "Gramian" or "Gram" matrix; these terms have more general meanings, but they do include any matrix expressible as $B^{\mathrm{T}}B$.)

8.6.1.1 Sums of Squares and Cross Products

Although the properties of Gramian matrices are of interest, our starting point is usually the data matrix X, which we may analyze by forming a Gramian matrix $X^{\mathrm{T}}X$ or XX^{T} (or a related matrix). These Gramian matrices are also called *sums of squares and cross products matrices*. (The term "cross product" does not refer to the cross product of vectors defined on page 47, but rather to the presence of sums over i of the products $x_{ij}x_{ik}$ along with sums of squares x_{ij}^2.) These matrices and other similar ones are useful association matrices in statistical applications.

8.6.1.2 Some Immediate Properties of Gramian Matrices

Some interesting properties of a Gramian matrix $X^{\mathrm{T}}X$ that we have discussed are:

- $X^{\mathrm{T}}X$ is symmetric.
- $\mathrm{rank}(X^{\mathrm{T}}X) = \mathrm{rank}(X)$.
- $X^{\mathrm{T}}X$ is of full rank if and only if X is of full column rank.
- $X^{\mathrm{T}}X$ is nonnegative definite.
- $X^{\mathrm{T}}X$ is positive definite if and only if X is of full column rank.
- $X^{\mathrm{T}}X = 0 \iff X = 0$.
- $BX^{\mathrm{T}}X = CX^{\mathrm{T}}X \iff BX^{\mathrm{T}} = CX^{\mathrm{T}}$.
- $X^{\mathrm{T}}XB = X^{\mathrm{T}}XC \iff XB = XC$.
- If d is a singular value of X, then $c = d^2$ is an eigenvalue of $X^{\mathrm{T}}X$; or, expressed another way,
  if c is a nonzero eigenvalue of $X^{\mathrm{T}}X$, then there is a singular value d of X such that $d^2 = c$.

These properties were shown in Sect. 3.3.10, beginning on page 115, except for the last one, which was shown on page 162.

Each element of a Gramian matrix is the dot product of columns of the constituent matrix. If x_{*i} and x_{*j} are the i^{th} and j^{th} columns of the matrix X, then

$$(X^{\mathrm{T}}X)_{ij} = x_{*i}^{\mathrm{T}}x_{*j}. \tag{8.44}$$

A Gramian matrix is also the sum of the outer products of the rows of the constituent matrix. If x_{i*} is the i^{th} row of the $n \times m$ matrix X, then

$$X^{\mathrm{T}}X = \sum_{i=1}^{n} x_{i*}x_{i*}^{\mathrm{T}}. \tag{8.45}$$

This is generally the way a Gramian matrix is computed.

By equation (8.14), we see that any Gramian matrix formed from a general matrix X is the same as a Gramian matrix formed from a square upper triangular matrix T:

$$X^{\mathrm{T}}X = T^{\mathrm{T}}T.$$

8.6.1.3 Generalized Inverses of Gramian Matrices

The generalized inverses of $X^{\mathrm{T}}X$ have useful properties. First, we see from the definition, for any generalized inverse $(X^{\mathrm{T}}X)^-$, that $((X^{\mathrm{T}}X)^-)^{\mathrm{T}}$ is also a generalized inverse of $X^{\mathrm{T}}X$. (Note that $(X^{\mathrm{T}}X)^-$ is not necessarily symmetric.) Also, we have, from equation (3.162),

$$X(X^{\mathrm{T}}X)^-X^{\mathrm{T}}X = X. \tag{8.46}$$

This means that $(X^{\mathrm{T}}X)^-X^{\mathrm{T}}$ is a generalized inverse of X.

The Moore-Penrose inverse of X has an interesting relationship with a generalized inverse of $X^{\mathrm{T}}X$:

$$XX^+ = X(X^{\mathrm{T}}X)^-X^{\mathrm{T}}. \tag{8.47}$$

This can be established directly from the definition of the Moore-Penrose inverse.

An important property of $X(X^{\mathrm{T}}X)^-X^{\mathrm{T}}$ is its invariance to the choice of the generalized inverse of $X^{\mathrm{T}}X$. Suppose G is any generalized inverse of $X^{\mathrm{T}}X$. Then, from equation (8.46), we have $X(X^{\mathrm{T}}X)^-X^{\mathrm{T}}X = XGX^{\mathrm{T}}X$, and from the implication (3.162), we have

$$XGX^{\mathrm{T}} = X(X^{\mathrm{T}}X)^-X^{\mathrm{T}}; \tag{8.48}$$

that is, $X(X^{\mathrm{T}}X)^-X^{\mathrm{T}}$ is invariant to the choice of generalized inverse.

8.6.1.4 Eigenvalues of Gramian Matrices

The nonzero eigenvalues of $X^{\mathrm{T}}X$ are the same as the nonzero eigenvalues of XX^{T} (property 14 on page 140).

If the singular value decomposition of X is UDV^{T} (page 161), then the similar canonical factorization of $X^{\mathrm{T}}X$ (equation (3.252)) is $VD^{\mathrm{T}}DV^{\mathrm{T}}$. Hence, we see that the nonzero singular values of X are the square roots of the nonzero eigenvalues of the symmetric matrix $X^{\mathrm{T}}X$. By using DD^{T} similarly, we see that they are also the square roots of the nonzero eigenvalues of XX^{T}.

8.6.2 Projection and Smoothing Matrices

It is often of interest to approximate an arbitrary n-vector in a given m-dimensional vector space, where $m < n$. An $n \times n$ projection matrix of rank m clearly does this.

8.6.2.1 A Projection Matrix Formed from a Gramian Matrix

An important matrix that arises in analysis of a linear model of the form

$$y = X\beta + \epsilon \tag{8.49}$$

is

$$H = X(X^{\mathrm{T}}X)^{-}X^{\mathrm{T}}, \tag{8.50}$$

where $(X^{\mathrm{T}}X)^{-}$ is any generalized inverse. From equation (8.48), H is invariant to the choice of generalized inverse. By equation (8.47), this matrix can be obtained from the pseudoinverse and so

$$H = XX^{+}. \tag{8.51}$$

In the full rank case, this is uniquely

$$H = X(X^{\mathrm{T}}X)^{-1}X^{\mathrm{T}}. \tag{8.52}$$

Whether or not X is of full rank, H is a projection matrix onto $\mathrm{span}(X)$. It is called the "hat matrix" because it projects the observed response vector, often denoted by y, onto a *predicted* response vector, often denoted by $\widehat{y}$ in $\mathrm{span}(X)$:

$$\widehat{y} = Hy. \tag{8.53}$$

Because H is invariant, this projection is invariant to the choice of generalized inverse. (In the nonfull rank case, however, we generally refrain from referring to the vector Hy as the "predicted response"; rather, we may call it the "fitted response".)

The rank of H is the same as the rank of X, and its trace is the same as its rank (because it is idempotent). When X is of full column rank, we have

$$\mathrm{tr}(H) = \text{number of columns of } X. \tag{8.54}$$

(This can also be seen by using the invariance of the trace to permutations of the factors in a product as in equation (3.79).)

In linear models, $\mathrm{tr}(H)$ is the model degrees of freedom, and the sum of squares due to the model is just $y^{\mathrm{T}}Hy$.

The complementary projection matrix,

$$I - H, \tag{8.55}$$

also has interesting properties that relate to linear regression analysis. In geometrical terms, this matrix projects a vector onto $\mathcal{N}(X^{\mathrm{T}})$, the orthogonal complement of $\mathrm{span}(X)$. We have

$$\begin{aligned} y &= Hy + (I - H)y \\ &= \widehat{y} + r, \end{aligned} \tag{8.56}$$

where $r = (I - H)y \in \mathcal{N}(X^{\mathrm{T}})$. The orthogonal complement is called the residual vector space, and r is called the residual vector. Both the rank and the trace of the orthogonal complement are the number of rows in X (that is, the number of observations) minus the regression degrees of freedom. This quantity is the "residual degrees of freedom" (unadjusted).

These two projection matrices (8.50) or (8.52) and (8.55) partition the total sum of squares:

$$y^{\mathrm{T}}y = y^{\mathrm{T}}Hy \; + \; y^{\mathrm{T}}(I - H)y. \tag{8.57}$$

This partitioning yields the total sum of squares into a sum of squares due to the fitted relationship between y and X and a "residual" sum of squares. The analysis of these two sums of squares is one of the most fundamental and important techniques in statistics. Note that the second term in this partitioning is the Schur complement of $X^{\mathrm{T}}X$ in $[X\, y]^{\mathrm{T}}\,[X\, y]$ (see equation (3.191) on page 122).

8.6.2.2 Smoothing Matrices

The hat matrix, either from a full rank X as in equation (8.52) or formed by a generalized inverse as in equation (8.50), *smoothes* the vector y onto the hyperplane defined by the column space of X. It is therefore a *smoothing* matrix. (Note that the rank of the column space of X is the same as the rank of $X^{\mathrm{T}}X$.)

A useful variation of the cross products matrix $X^{\mathrm{T}}X$ is the matrix formed by adding a nonnegative (positive) definite matrix A to it. Because $X^{\mathrm{T}}X$ is

nonnegative (positive) definite, $X^{\mathrm{T}}X + A$ is nonnegative definite, as we have seen (page 349), and hence $X^{\mathrm{T}}X + A$ is a Gramian matrix.

Because the square root of the nonnegative definite A exists, we can express the sum of the matrices as

$$X^{\mathrm{T}}X + A = \begin{bmatrix} X \\ A^{\frac{1}{2}} \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} X \\ A^{\frac{1}{2}} \end{bmatrix}. \tag{8.58}$$

In a common application, a positive definite matrix λI, with $\lambda > 0$, is added to $X^{\mathrm{T}}X$, and this new matrix is used as a smoothing matrix. The analogue of the hat matrix (8.52) is

$$H_\lambda = X(X^{\mathrm{T}}X + \lambda I)^{-1}X^{\mathrm{T}}, \tag{8.59}$$

and the analogue of the fitted response is

$$\widehat{y}_\lambda = H_\lambda y. \tag{8.60}$$

This has the effect of shrinking the $\widehat{y}$ of equation (8.53) toward 0. (In regression analysis, this is called "ridge regression"; see page 364 or 431.)

Any matrix such as H_λ that is used to transform the observed vector y onto a given subspace is called a smoothing matrix.

8.6.2.3 Effective Degrees of Freedom

Because of the shrinkage in ridge regression (that is, because the fitted model is less dependent just on the data in X) we say the "effective" degrees of freedom of a ridge regression model decreases with increasing λ. We can formally define the *effective model degrees of freedom* of any linear fit $\widehat{y} = H_\lambda y$ as

$$\mathrm{tr}(H_\lambda), \tag{8.61}$$

analogous to the model degrees of freedom in linear regression above. This definition of effective degrees of freedom applies generally in data smoothing. In fact, many smoothing matrices used in applications depend on a single smoothing parameter such as the λ in ridge regression, and so the same notation H_λ is often used for a general smoothing matrix.

To evaluate the effective degrees of freedom in the ridge regression model for a given λ and X, for example, using the singular value decomposition of X, $X = UDV^{\mathrm{T}}$, we have

$$\begin{aligned} &\mathrm{tr}(X(X^{\mathrm{T}}X + \lambda I)^{-1}X^{\mathrm{T}}) \\ &\quad = \mathrm{tr}\left(UDV^{\mathrm{T}}(VD^2V^{\mathrm{T}} + \lambda VV^{\mathrm{T}})^{-1}VDU^{\mathrm{T}}\right) \\ &\quad = \mathrm{tr}\left(UDV^{\mathrm{T}}(V(D^2 + \lambda I)V^{\mathrm{T}})^{-1}VDU^{\mathrm{T}}\right) \\ &\quad = \mathrm{tr}\left(UD(D^2 + \lambda I)^{-1}DU^{\mathrm{T}}\right) \\ &\quad = \mathrm{tr}\left(D^2(D^2 + \lambda I)^{-1}\right) \\ &\quad = \sum \frac{d_i^2}{d_i^2 + \lambda}, \end{aligned} \tag{8.62}$$

where the d_i are the singular values of X.

When $\lambda = 0$, this is the same as the ordinary model degrees of freedom, and when λ is positive, this quantity is smaller, as we would want it to be by the argument above. The $d_i^2/(d_i^2 + \lambda)$ are called shrinkage factors.

If $X^{\mathrm{T}}X$ is not of full rank, the addition of λI to it also has the effect of yielding a full rank matrix, if $\lambda > 0$, and so the inverse of $X^{\mathrm{T}}X + \lambda I$ exists even when that of $X^{\mathrm{T}}X$ does not. In any event, the addition of λI to $X^{\mathrm{T}}X$ yields a matrix with a better condition number, which we discussed in Sect. 6.1. (On page 272, we showed that the condition number of $X^{\mathrm{T}}X + \lambda I$ is better than that of $X^{\mathrm{T}}X$.)

8.6.2.4 Residuals from Smoothed Data

Just as in equation (8.56), we can write

$$y = \widehat{y}_\lambda + r_\lambda. \tag{8.63}$$

Notice, however, that in $\widehat{y}_\lambda = H_\lambda y$, H_λ is not in general a projection matrix. Unless H_λ is a projection matrix, $\widehat{y}_\lambda$ and r_λ are not orthogonal in general as are $\widehat{y}$ and r, and we do not have the additive partitioning of the sum of squares as in equation (8.57).

The rank of H_λ is the same as the number of columns of X, but the trace, and hence the model degrees of freedom, is less than this number.

8.6.3 Centered Matrices and Variance-Covariance Matrices

In Sect. 2.3, we defined the variance of a vector and the covariance of two vectors. These are the same as the "sample variance" and "sample covariance" in statistical data analysis and are related to the variance and covariance of random variables in probability theory. We now consider the variance-covariance matrix associated with a data matrix. We occasionally refer to the variance-covariance matrix simply as the "variance matrix" or just as the "variance".

First, we consider centering and scaling data matrices.

8.6.3.1 Centering and Scaling of Data Matrices

When the elements in a vector represent similar measurements or observational data on a given phenomenon, summing or averaging the elements in the vector may yield meaningful statistics. In statistical applications, the columns in a matrix often represent measurements on the same feature or on the same variable over different observational units as in Fig. 8.1, and so the mean of a column may be of interest.

We may center the column by subtracting its mean from each element in the same manner as we centered vectors on page 48. The matrix formed by centering all of the columns of a given matrix is called a centered matrix,

and if the original matrix is X, we represent the centered matrix as X_c in a notation analogous to what we introduced for centered vectors. If we represent the matrix whose i^{th} column is the constant mean of the i^{th} column of X as $\overline{X}$,

$$X_c = X - \overline{X}. \tag{8.64}$$

Here is an R statement to compute this:

```
Xc <- X-rep(1,n)%*%t(apply(X,2,mean))
```

If the unit of a measurement is changed, all elements in a column of the data matrix in which the measurement is used will change. The amount of variation of elements within a column or the relative variation among different columns ideally should not be measured in terms of the basic units of measurement, which can differ irreconcilably from one column to another. (One column could represent scores on an exam and another column could represent weight, for example.)

In analyzing data, it is usually important to scale the variables so that their variations are comparable. We do this by using the standard deviation of the column. If we have also centered the columns, the column vectors are the centered and scaled vectors of the form of those in equation (2.74),

$$x_{cs} = \frac{x_c}{s_x},$$

where s_x is the standard deviation of x,

$$s_x = \frac{\|x_c\|}{\sqrt{n-1}}.$$

If all columns of the data matrix X are centered and scaled, we denote the resulting matrix as X_{cs}. If s_i represents the standard deviation of the i^{th} column, this matrix is formed as

$$X_{cs} = X_c \text{diag}(1/s_i). \tag{8.65}$$

Here is an R statement to compute this:

```
Xcs <- Xc%*%diag(1/apply(X,2,sd))
```

If the rows of X are taken as representative of a population of similar vectors, it is often useful to center and scale any vector from that population in the manner of equation (8.65):

$$\tilde{x} = \text{diag}(1/s_i)x_c. \tag{8.66}$$

(Note that x_c is a vector of the same order as a row of X_c.)

8.6.3.2 Gramian Matrices Formed from Centered Matrices: Covariance Matrices

An important Gramian matrix is formed as the sums of squares and cross products matrix from a centered matrix and scaled by $(n-1)$, where n is the number of rows of the original matrix:

$$\begin{aligned} S_X &= \frac{1}{n-1} X_{\mathrm{c}}^{\mathrm{T}} X_{\mathrm{c}} \\ &= (s_{ij}). \end{aligned} \tag{8.67}$$

This matrix is called the *variance-covariance matrix* associated with the given matrix X, or the *sample variance-covariance matrix*, to distinguish it from the variance-covariance matrix of the distribution given in equation (4.71) on page 218. We denote it by S_X or just S. If x_{*i} and x_{*j} are the vectors corresponding to the i^{th} and j^{th} columns of X, then $s_{ij} = \mathrm{Cov}(x_{*i}, x_{*j})$; that is, the off-diagonal elements are the covariances between the column vectors, as in equation (2.78), and the diagonal elements are *variances* of the column vectors.

This matrix and others formed from it, such as R_X in equation (8.69) below, are called *association matrices* because they are based on measures of association (covariance or correlation) among the columns of X. We could likewise define a Gramian association matrix based on measures of association among the rows of X.

A transformation using the Cholesky factor of S_X or the square root of S_X (assuming S_X is full rank) results in a matrix whose associated variance-covariance is the identity. We call this a sphered matrix:

$$X_{\mathrm{sphered}} = X_{\mathrm{c}} S_X^{-\frac{1}{2}}. \tag{8.68}$$

The matrix S_X is a measure of the anisometry of the space of vectors represented by the rows of X as mentioned in Sect. 3.2.9. The inverse, S_X^{-1}, in some sense evens out the anisometry. Properties of vectors in the space represented by the rows of X are best assessed following a transformation as in equation (8.66). For example, rather than orthogonality of two vectors u and v, a more interesting relationship would be S_X^{-1}-conjugacy (see equation (3.93)):

$$u^{\mathrm{T}} S_X^{-1} v = 0.$$

Also, the Mahalanobis distance, $\sqrt{(u-v)^{\mathrm{T}} S_X^{-1} (u-v)}$, may be more relevant for measuring the difference in two vectors than the standard Euclidean distance.

8.6.3.3 Gramian Matrices Formed from Scaled Centered Matrices: Correlation Matrices

If the columns of a centered matrix are standardized (that is, divided by their standard deviations, assuming that each is nonconstant, so that the standard deviation is positive), the scaled cross products matrix is the *correlation matrix*, often denoted by R_X or just R,

$$\begin{aligned} R_X &= \frac{1}{n-1} X_{\mathrm{cs}}^{\mathrm{T}} X_{\mathrm{cs}} \\ &= (r_{ij}), \end{aligned} \tag{8.69}$$

where if x_{*i} and x_{*j} are the vectors corresponding to the i^{th} and j^{th} columns of X, $r_{ij} = \mathrm{Cor}(x_{*i}, x_{*j})$. The correlation matrix can also be expressed as $R_X = X_{\mathrm{c}}^{\mathrm{T}} D X_{\mathrm{c}}$, where D is the diagonal matrix whose k^{th} diagonal is $1/\sqrt{\mathrm{V}(x_{*k})}$, where $\mathrm{V}(x_{*k})$ is the sample variance of the k^{th} column; that is, $\mathrm{V}(x_{*k}) = \sum_i (x_{ik} - \bar{x}_{*k})^2/(n-1)$. This Gramian matrix R_X is based on measures of association among the columns of X.

The elements along the diagonal of the correlation matrix are all 1, and the off-diagonals are between -1 and 1, each being the correlation between a pair of column vectors, as in equation (2.80). The correlation matrix is nonnegative definite because it is a Gramian matrix.

The trace of an $n \times n$ correlation matrix is n, and therefore the eigenvalues, which are all nonnegative, sum to n.

Without reference to a data matrix, any nonnegative definite matrix with 1s on the diagonal and with all elements less than or equal to 1 in absolute value is called a *correlation matrix*.

8.6.4 The Generalized Variance

The diagonal elements of the variance-covariance matrix S associated with the $n \times m$ data matrix X are the second moments of the centered columns of X, and the off-diagonal elements are pairwise second central moments of the columns. Each element of the matrix provides some information about the spread of a single column or of two columns of X. The determinant of S provides a single overall measure of the spread of the columns of X. This measure, $|S|$, is called the *generalized variance*, or generalized *sample* variance, to distinguish it from an analogous measure of a distributional model.

On page 74, we discussed the equivalence of a determinant and the volume of a parallelotope. The generalized variance captures this, and when the columns or rows of S are more orthogonal to each other, the volume of the parallelotope determined by the columns or rows of S is greater, as shown in Fig. 8.6 for $m = 3$.

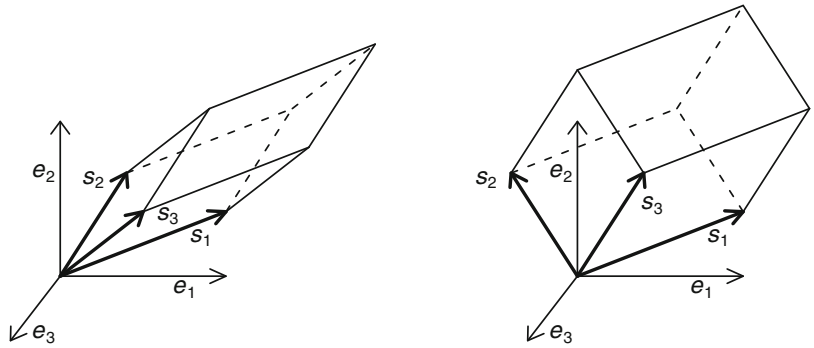


Figure 8.6. Generalized variances in terms of the columns of S

The columns or rows of S are generally not of much interest in themselves. Our interest is in the relationship of the centered columns of the $n \times m$ data matrix X. Let us consider the case of $m = 2$. Let z_{*1} and z_{*2} represent the centered column vectors of X; that is, for z_{*1}, we have $z_{*1_i} = x_{*1_i} - \bar{x_1}$. Now, as in equation (3.42), consider the parallelogram formed by z_{*1} and z_{*2}. For computing the area, consider z_{*1} as forming the base. The length of the base is

$$\|z_{*1}\| = \sqrt{(n-1)s_{11}},$$

and the height is

$$\|z_{*2}\| |\sin(\theta)| = \sqrt{(n-1)s_{22}(1-r_{12}^2)}.$$

(Recall the relationship between the angle and the correlation from equation (2.81).)

The area of the parallelogram therefore is

$$\text{area} = (n-1)\sqrt{s_{11}s_{22}(1-r_{12}^2)}.$$

Now, consider S:

$$\begin{aligned} S &= \begin{bmatrix} s_{11} & s_{21} \\ s_{12} & s_{22} \end{bmatrix} \\ &= \begin{bmatrix} s_{11} & \sqrt{s_{11}s_{22}}r_{12} \\ \sqrt{s_{11}s_{22}}r_{12} & s_{22} \end{bmatrix}. \end{aligned}$$

The determinant of S is therefore

$$s_{11}s_{22}(1-r_{12}^2),$$

that is,

$$|S| = \frac{1}{(n-1)^m}\text{volume}^2. \tag{8.70}$$

Although we considered only the case $m = 2$, equation (8.70) holds generally, as can be seen by induction on m (see Anderson, 2003).

8.6.4.1 Comparing Variance-Covariance Matrices

Many standard statistical procedures for comparing groups of data rely on the assumption that the population variance-covariance matrices of the groups are all the same. (The simplest example of this is the two-sample t-test, in which the concern is just that the population variances of the two groups be equal.) Occasionally, the data analyst wishes to test this assumption of homogeneity of variances.

On page 175, we considered the problem of comparing two matrices of the same size. There we defined a metric based on a matrix norm. For the problem of comparing variance-covariance matrices, a measure based on the generalized variances is more commonly used.

In the typical situation, we have an $n \times m$ data matrix X in which the first n_1 rows represent observations from one group, the next n_2 rows represent observations from another group, and the last n_g rows represent observations from the g^{th} group. For each group, we form a sample variance-covariance matrix S_i as in equation (8.67) using the i^{th} submatrix of X. Whenever we have individual variance-covariance matrices in situations similar to this, we define the *pooled variance-covariance matrix*:

$$S_{\text{p}} = \frac{1}{(n-g)}\sum_{i=1}^{g}(n_i - 1)S_i. \tag{8.71}$$

Bartlett suggested a test based on the determinants of $(n_i - 1)S_i$. (From equation (3.36), $|(n_i - 1)S_i| = (n_i - 1)^m|S_i|$.) A similar test suggested by Box also uses the generalized variances. One form of the Box M statistic for testing for homogeneity of variances is

$$M = (n-g)\log(|S_{\text{p}}|) - \sum_{i=1}^{g}(n_i - 1)S_i. \tag{8.72}$$

8.6.5 Similarity Matrices

Covariance and correlation matrices are examples of *similarity association matrices*: they measure the similarity or closeness of the columns of the matrices from which they are formed.

The cosine of the angle between two vectors is related to the correlation between the vectors, so a matrix of the cosine of the angle between the columns

of a given matrix would also be a similarity matrix. (The angle is exactly the same as the correlation if the vectors are centered; see equation (2.80).)

Similarity matrices can be formed in many ways, and some are more useful in particular applications than in others. They may not even arise from a standard dataset in the familiar form in statistical applications. For example, we may be interested in comparing text documents. We might form a vector-like object whose elements are the words in the document. The similarity between two such ordered tuples, generally of different lengths, may be a count of the number of words, word pairs, or more general phrases in common between the two documents.

It is generally reasonable that similarity between two objects be symmetric; that is, the first object is as close to the second as the second is to the first. We reserve the term similarity matrix for matrices formed from such measures and, hence, that themselves are symmetric. Occasionally, for example in psychometric applications, the similarities are measured relative to rank order closeness within a set. In such a case, the measure of closeness may not be symmetric. A matrix formed from such measures is called a *directed similarity matrix*.

8.6.6 Dissimilarity Matrices

The elements of similarity generally increase with increasing "closeness". We may also be interested in the dissimilarity. Clearly, any decreasing function of similarity could be taken as a reasonable measure of dissimilarity. There are, however, other measures of dissimilarity that often seem more appropriate. In particular, the properties of a metric (see page 32) may suggest that a dissimilarity be defined in terms of a metric.

In considering either similarity or dissimilarity for a data matrix, we could work with either rows or columns, but the common applications make one or the other more natural for the specific application. Because of the types of the common applications, we will discuss dissimilarities and distances in terms of rows instead of columns; thus, in this section, we will consider the dissimilarity of x_{i*} and x_{j*}, the vectors corresponding to the i^{th} and j^{th} rows of X.

Dissimilarity matrices are also *association matrices* in the general sense of Sect. 8.1.5.

Dissimilarity or distance can be measured in various ways. A metric is the most obvious measure, although in certain applications other measures are appropriate. The measures may be based on some kind of ranking, for example. If the dissimilarity is based on a metric, the association matrix is often called a *distance matrix*. In most applications, the Euclidean distance, $\|x_{i*} - x_{j*}\|_2$, is the most commonly used metric, but others, especially ones based on L_1 or L_∞ norms, are often useful.

Any hollow matrix with nonnegative elements is a *directed dissimilarity matrix*. A directed dissimilarity matrix is also called a *cost matrix*. If the matrix is symmetric, it is a *dissimilarity matrix*.

An $n \times n$ matrix $D = (d_{ij})$ is an m-dimensional *distance matrix* if there exists m-vectors $x_1 \ldots, x_n$ such that, for some metric Δ, $d_{ij} = \Delta(x_i, x_j)$. A distance matrix is necessarily a dissimilarity matrix. If the metric is the Euclidean distance, the matrix D is called a *Euclidean distance matrix*.

The matrix whose rows are the vectors $x_1^{\mathrm{T}}, \ldots, x_n^{\mathrm{T}}$ is called an associated *configuration matrix* of the given distance matrix. In metric multidimensional scaling, we are given a dissimilarity matrix and seek to find a configuration matrix whose associated distance matrix is closest to the dissimilarity matrix, usually in terms of the Frobenius norm of the difference of the matrices (see Trosset 2002; for basic definitions and extensions).

8.7 Nonnegative and Positive Matrices

A *nonnegative matrix*, as the name suggests, is a real matrix all of whose elements are nonnegative, and a *positive matrix* is a real matrix all of whose elements are positive. In some other literature, the latter type of matrix is called strictly positive, and a nonnegative matrix with a positive element is called positive.

Many useful matrices are nonnegative, and we have already considered various kinds of nonnegative matrices. The adjacency or connectivity matrix of a graph is nonnegative. Dissimilarity matrices, including distance matrices, are nonnegative. Matrices used in modeling stochastic processes are nonnegative. While many of these matrices are square, we do not restrict the definitions to square matrices.

If A is nonnegative, we write

$$A \geq 0, \tag{8.73}$$

and if it is positive, we write

$$A > 0. \tag{8.74}$$

Notice that $A \geq 0$ and $A \neq 0$ together do not imply $A > 0$.

We write

$$A \geq B$$

to mean $(A - B) \geq 0$ and

$$A > B$$

to mean $(A - B) > 0$. (Recall the definitions of nonnegative definite and positive definite matrices, and, from equations (8.11) and (8.16), the notation used to indicate those properties, $A \succeq 0$ and $A \succ 0$. Furthermore, notice that these definitions and this notation for nonnegative and positive matrices are consistent with analogous definitions and notation involving vectors on page 16.

Some authors, however, use the notation of equations (8.73) and (8.74) to mean "nonnegative definite" and "positive definite". We should also note that some authors use somewhat different terms for these and related properties. "Positive" for these authors means nonnegative with at least one positive element, and "strictly positive" means positive as we have defined it.)

Notice that positiveness (nonnegativeness) has nothing to do with positive (nonnegative) definiteness. A positive or nonnegative matrix need not be symmetric or even square, although most such matrices useful in applications are square. A square positive matrix, unlike a positive definite matrix, need not be of full rank.

The following properties are easily verified.

1. If $A \geq 0$ and $u \geq v \geq 0$, then $Au \geq Av$.
2. If $A \geq 0$, $A \neq 0$, and $u > v > 0$, then $Au > Av$.
3. If $A > 0$ and $v \geq 0$, then $Av \geq 0$.
4. If $A > 0$ and A is square, then $\rho(A) > 0$.

Whereas most of the important matrices arising in the analysis of linear models are symmetric, and thus have the properties listed in Sect. 8.2.1 beginning on page 340, many important nonnegative matrices, such as those used in studying stochastic processes, are not necessarily symmetric. The eigenvalues of real symmetric matrices are real, but *the eigenvalues of real nonsymmetric matrices may have imaginary components*. In the following discussion, we must be careful to remember the meaning of the spectral radius. The definition in equation (3.233) for the spectral radius of the matrix A with eigenvalues c_i,

$$\rho(A) = \max |c_i|,$$

is still correct, but the operator "$|\cdot|$" must be interpreted as the modulus of a complex number.

8.7.1 The Convex Cones of Nonnegative and Positive Matrices

The class of all $n \times m$ nonnegative (positive) matrices is a *cone* because if X is a nonnegative (positive) matrix and $a > 0$, then aX is a nonnegative (positive) matrix (see page 43). Furthermore, it is a *convex cone* in $\mathbb{R}^{n\times m}$, because if X_1 and X_2 are $n \times m$ nonnegative (positive) matrices and $a, b \geq 0$, then $aX_1 + bX_2$ is nonnegative (positive) so long as either $a \neq 0$ or $b \neq 0$.

Both of these classes are closed under Cayley multiplication, but they may not have inverses in the class (that is, in particular, they are not groups with respect to that operation).

8.7.2 Properties of Square Positive Matrices

We have the following important properties for square positive matrices. These properties collectively are the conclusions of the *Perron theorem*.

Let A be a square positive matrix and let $r = \rho(A)$. Then:

1. r is an eigenvalue of A. The eigenvalue r is called the *Perron root*. Note that the Perron root is real and positive (although other eigenvalues of A may not be).
2. There is an eigenvector v associated with r such that $v > 0$.
3. The Perron root is simple. (That is, the algebraic multiplicity of the Perron root is 1.)
4. The dimension of the eigenspace of the Perron root is 1. (That is, the geometric multiplicity of $\rho(A)$ is 1.) Hence, if v is an eigenvector associated with r, it is unique except for scaling. This associated eigenvector is called the *Perron vector*. Note that the Perron vector is real (although other eigenvectors of A may not be). The elements of the Perron vector all have the same sign, which we usually take to be positive; that is, $v > 0$.
5. If c_i is any other eigenvalue of A, then $|c_i| < r$. (That is, r is the only eigenvalue on the spectral circle of A.)

We will give proofs only of properties 1 and 2 as examples. Proofs of all of these facts are available in Bapat and Raghavan (1997).

To see properties 1 and 2, first observe that a positive matrix must have at least one nonzero eigenvalue because the coefficients and the constant in the characteristic equation must all be positive. Now scale the matrix so that its spectral radius is 1 (see page 142). So without loss of generality, let A be a scaled positive matrix with $\rho(A) = 1$. Now let (c, x) be some eigenpair of A such that $|c| = 1$. First, we want to show, for some such c, that $c = \rho(A)$.

Because all elements of A are positive,

$$|x| = |Ax| \leq A|x|,$$

and so

$$A|x| - |x| \geq 0. \tag{8.75}$$

An eigenvector must be nonzero, so we also have

$$A|x| > 0.$$

Now we want to show that $A|x| - |x| = 0$. To that end, suppose the contrary; that is, suppose $A|x| - |x| \neq 0$. In that case, $A(A|x| - |x|) > 0$ from equation (8.75), and so there must be a positive number ϵ such that

$$\frac{A}{1+\epsilon} A|x| > A|x|$$

or

$$By > y,$$

where $B = A/(1+\epsilon)$ and $y = A|x|$. Now successively multiplying both sides of this inequality by the positive matrix B, we have

$$B^k y > y \quad \text{for all } k = 1, 2, \ldots.$$

Because $\rho(B) = \rho(A)/(1+\epsilon) < 1$, from equation (3.314) on page 173, we have $\lim_{k\to\infty} B^k = 0$; that is, $\lim_{k\to\infty} B^k y = 0 > y$. This contradicts the fact that $y > 0$. Because the supposition $A|x| - |x| \neq 0$ led to this contradiction, we must have $A|x| - |x| = 0$. Therefore $1 = \rho(A)$ must be an eigenvalue of A, and $|x|$ must be an associated eigenvector; hence, with $v = |x|$, $(\rho(A), v)$ is an eigenpair of A and $v > 0$, and this is the statement made in properties 1 and 2.

The Perron-Frobenius theorem, which we consider below, extends these results to a special class of square nonnegative matrices. (This class includes all positive matrices, so the Perron-Frobenius theorem is an extension of the Perron theorem.)

8.7.3 Irreducible Square Nonnegative Matrices

Nonnegativity of a matrix is not a very strong property. First of all, note that it includes the zero matrix; hence, clearly none of the properties of the Perron theorem can hold. Even a nondegenerate, full rank nonnegative matrix does not necessarily possess those properties. A small full rank nonnegative matrix provides a counterexample for properties 2, 3, and 5:

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

The eigenvalues are 1 and 1; that is, 1 with an algebraic multiplicity of 2 (so property 3 does not hold). There is only one nonnull eigenvector, $(1, -1)$, (so property 2 does not hold, but property 4 holds), but the eigenvector is not positive (or even nonnegative). Of course property 5 cannot hold if property 3 does not hold.

We now consider irreducible square nonnegative matrices. This class includes positive matrices, and irreducibility yields some of the properties of positivity to matrices with some zero elements. (Heuristically, irreducibility puts some restrictions on the nonpositive elements of the matrix.) On page 337, we defined reducibility of a nonnegative square matrix and we saw that a matrix is irreducible if and only if its digraph is strongly connected.

To recall the definition, a nonnegative matrix is said to be *reducible* if by symmetric permutations it can be put into a block upper triangular matrix with square blocks along the diagonal; that is, the nonnegative matrix A is reducible if and only if there is a permutation matrix $E_{(\pi)}$ such that

$$E_{(\pi)} A E_{(\pi)} = \begin{bmatrix} B_{11} & B_{12} \\ 0 & B_{22} \end{bmatrix}, \tag{8.76}$$

where B_{11} and B_{22} are square. A matrix that cannot be put into that form is *irreducible*. An alternate term for reducible is *decomposable*, with the associated term *indecomposable*. (There is an alternate meaning for the term

"reducible" applied to a matrix. This alternate use of the term means that the matrix is capable of being expressed by a similarity transformation as the sum of two matrices whose columns are mutually orthogonal.)

We see from the definition in equation (8.76) that a positive matrix is irreducible.

Irreducible matrices have several interesting properties. An $n \times n$ nonnegative matrix A is irreducible if and only if $(I + A)^{n-1}$ is a positive matrix; that is,

$$A \text{ is irreducible} \iff (I + A)^{n-1} > 0. \tag{8.77}$$

To see this, first assume $(I+A)^{n-1} > 0$; thus, $(I+A)^{n-1}$ clearly is irreducible. If A is reducible, then there exists a permutation matrix $E_{(\pi)}$ such that

$$E_{(\pi)} A E_{(\pi)} = \begin{bmatrix} B_{11} & B_{12} \\ 0 & B_{22} \end{bmatrix},$$

and so

$$\begin{aligned} E_{(\pi)}(I + A)^{n-1} E_{(\pi)} &= \left(E_{(\pi)}(I + A)E_{(\pi)}\right)^{n-1} \\ &= \left(I + E_{(\pi)} A E_{(\pi)}\right)^{n-1} \\ &= \begin{bmatrix} I_{n_1} + B_{11} & B_{12} \\ 0 & I_{n_2} + B_{22} \end{bmatrix}. \end{aligned}$$

This decomposition of $(I+A)^{n-1}$ cannot exist because it is irreducible; hence we conclude A is irreducible if $(I + A)^{n-1} > 0$.

Now, if A is irreducible, we can see that $(I + A)^{n-1}$ must be a positive matrix either by a strictly linear-algebraic approach or by couching the argument in terms of the digraph $\mathcal{G}(A)$ formed by the matrix, as in the discussion on page 337 that showed that a digraph is strongly connected if (and only if) it is irreducible. We will use the latter approach in the spirit of applications of irreducibility in stochastic processes.

For either approach, we first observe that the $(i, j)^{\text{th}}$ element of $(I+A)^{n-1}$ can be expressed as

$$\left((I + A)^{n-1}\right)_{ij} = \left(\sum_{k=0}^{n-1} \binom{n-1}{k} A^k\right)_{ij}. \tag{8.78}$$

Hence, for $k = 1, \ldots, n-1$, we consider the $(i, j)^{\text{th}}$ entry of A^k. Let $a_{ij}^{(k)}$ represent this quantity.

Given any pair (i, j), for some $l_1, l_2, \ldots, l_{k-1}$, we have

$$a_{ij}^{(k)} = \sum_{l_1, l_2, \ldots, l_{k-1}} a_{1l_1} a_{l_1 l_2} \cdots a_{l_{k-1} j}.$$

Now $a_{ij}^{(k)} > 0$ if and only if $a_{1l_1}, a_{l_1 l_2}, \ldots, a_{l_{k-1} j}$ are all positive; that is, if there is a path $v_1, v_{l_1}, \ldots, v_{l_{k-1}}, v_j$ in $\mathcal{G}(A)$. If A is irreducible, then $\mathcal{G}(A)$ is

strongly connected, and hence the path exists. So, for any pair (i, j), we have from equation (8.78) $\left((I + A)^{n-1}\right)_{ij} > 0$; that is, $(I + A)^{n-1} > 0$.

The positivity of $(I + A)^{n-1}$ for an irreducible nonnegative matrix A is a very useful property because it allows us to extend some conclusions of the Perron theorem to irreducible nonnegative matrices.

8.7.3.1 Properties of Square Irreducible Nonnegative Matrices: The Perron-Frobenius Theorem

If A is a square irreducible nonnegative matrix, then we have the following properties, which are similar to properties 1 through 4 on page 374 for positive matrices. These following properties are the conclusions of the *Perron-Frobenius theorem*.

1. $\rho(A)$ is an eigenvalue of A. This eigenvalue is called the *Perron root*, as before (and, as before, it is real and positive).
2. The Perron root $\rho(A)$ is simple. (That is, the algebraic multiplicity of the Perron root is 1.)
3. The dimension of the eigenspace of the Perron root is 1. (That is, the geometric multiplicity of $\rho(A)$ is 1.)
4. The eigenvector associated with $\rho(A)$ is positive. This eigenvector is called the *Perron vector*, as before.

The relationship (8.77) allows us to prove properties 1 and 4 in a method similar to the proofs of properties 1 and 2 for positive matrices. (This is Exercise 8.9.) Complete proofs of all of these facts are available in Bapat and Raghavan (1997). See also the solution to Exercise 8.10b on page 611 for a special case.

The one property of square positive matrices that does not carry over to square irreducible nonnegative matrices is property 5: $r = \rho(A)$ is the only eigenvalue on the spectral circle of A. For example, the small irreducible nonnegative matrix

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

has eigenvalues 1 and -1, and so both are on the spectral circle.

8.7.3.2 Primitive Matrices

Square irreducible nonnegative matrices that have only one eigenvalue on the spectral circle, that is, that do satisfy property 5 on page 374, also have other interesting properties that are important, for example, in Markov chains. We therefore give a name to matrices with that property:

> A square irreducible nonnegative matrix is said to be *primitive* if it has only one eigenvalue on the spectral circle.

8.7.3.3 Limiting Behavior of Primitive Matrices

In modeling with Markov chains and other applications, the limiting behavior of A^k is an important property.

On page 172, we saw that $\lim_{k\to\infty} A^k = 0$ if $\rho(A) < 1$. For a primitive matrix, we also have a limit for A^k if $\rho(A) = 1$. (As we have done above, we can scale any matrix with a nonzero eigenvalue to a matrix with a spectral radius of 1.)

If A is a primitive matrix, then we have the useful result

$$\lim_{k\to\infty} \left(\frac{A}{\rho(A)}\right)^k = vw^{\mathrm{T}}, \tag{8.79}$$

where v is an eigenvector of A associated with $\rho(A)$ and w is an eigenvector of A^{T} associated with $\rho(A)$, and w and v are scaled so that $w^{\mathrm{T}}v = 1$. (As we mentioned on page 158, such eigenvectors exist because $\rho(A)$ is a simple eigenvalue. They also exist because of property 4; they are both positive. Note that A is not necessarily symmetric, and so its eigenvectors may include imaginary components; however, the eigenvectors associated with $\rho(A)$ are real, and so we can write w^{T} instead of w^{H}.)

To see equation (8.79), we consider $\left(A - \rho(A)vw^{\mathrm{T}}\right)$. First, if (c_i, v_i) is an eigenpair of $\left(A - \rho(A)vw^{\mathrm{T}}\right)$ and $c_i \neq 0$, then (c_i, v_i) is an eigenpair of A. We can see this by multiplying both sides of the eigen-equation by vw^{T}:

$$\begin{aligned} c_i vw^{\mathrm{T}}v_i &= vw^{\mathrm{T}}\left(A - \rho(A)vw^{\mathrm{T}}\right)v_i \\ &= \left(vw^{\mathrm{T}}A - \rho(A)vw^{\mathrm{T}}vw^{\mathrm{T}}\right)v_i \\ &= \left(\rho(A)vw^{\mathrm{T}} - \rho(A)vw^{\mathrm{T}}\right)v_i \\ &= 0; \end{aligned}$$

hence,

$$\begin{aligned} Av_i &= \left(A - \rho(A)vw^{\mathrm{T}}\right)v_i \\ &= c_i v_i. \end{aligned}$$

Next, we show that

$$\rho\left(A - \rho(A)vw^{\mathrm{T}}\right) < \rho(A). \tag{8.80}$$

If $\rho(A)$ were an eigenvalue of $\left(A - \rho(A)vw^{\mathrm{T}}\right)$, then its associated eigenvector, say w, would also have to be an eigenvector of A, as we saw above. But since as an eigenvalue of A the geometric multiplicity of $\rho(A)$ is 1, for some scalar s, $w = sv$. But this is impossible because that would yield

$$\begin{aligned} \rho(A)sv &= \left(A - \rho(A)vw^{\mathrm{T}}\right)sv \\ &= sAv - s\rho(A)v \\ &= 0, \end{aligned}$$

and neither $\rho(A)$ nor sv is zero. But as we saw above, any eigenvalue of $\left(A - \rho(A)vw^{\mathrm{T}}\right)$ is an eigenvalue of A and no eigenvalue of $\left(A - \rho(A)vw^{\mathrm{T}}\right)$ can be as large as $\rho(A)$ in modulus; therefore we have inequality (8.80).

Finally, we recall equation (3.269), with w and v as defined above, and with the eigenvalue $\rho(A)$,

$$\left(A - \rho(A)vw^{\mathrm{T}}\right)^k = A^k - (\rho(A))^k vw^{\mathrm{T}}, \tag{8.81}$$

for $k = 1, 2, \ldots$.

Dividing both sides of equation (8.81) by $(\rho(A))^k$ and rearranging terms, we have

$$\left(\frac{A}{\rho(A)}\right)^k = vw^{\mathrm{T}} + \frac{\left(A - \rho(A)vw^{\mathrm{T}}\right)}{\rho(A)}. \tag{8.82}$$

Now

$$\rho\left(\frac{\left(A - \rho(A)vw^{\mathrm{T}}\right)}{\rho(A)}\right) = \frac{\rho\left(A - \rho(A)vw^{\mathrm{T}}\right)}{\rho(A)},$$

which is less than 1; hence, from equation (3.312) on page 172, we have

$$\lim_{k\to\infty}\left(\frac{\left(A - \rho(A)vw^{\mathrm{T}}\right)}{\rho(A)}\right)^k = 0;$$

so, taking the limit in equation (8.82), we have equation (8.79).

Applications of the Perron-Frobenius theorem are far-ranging. It has implications for the convergence of some iterative algorithms, such as the power method discussed in Sect. 7.2. The most important applications in statistics are in the analysis of Markov chains, which we discuss in Sect. 9.8.1.

8.7.4 Stochastic Matrices

A nonnegative matrix P such that

$$P1 = 1 \tag{8.83}$$

is called a *stochastic matrix*. The definition means that $(1, 1)$ is an eigenpair of any stochastic matrix. It is also clear that if P is a stochastic matrix, then $\|P\|_\infty = 1$ (see page 166), and because $\rho(P) \leq \|P\|$ for any norm (see page 171) and 1 is an eigenvalue of P, we have $\rho(P) = 1$.

A stochastic matrix may not be positive, and it may be reducible or irreducible. (Hence, $(1, 1)$ may not be the Perron root and Perron eigenvector.)

If P is a stochastic matrix such that

$$1^{\mathrm{T}}P = 1^{\mathrm{T}}, \tag{8.84}$$

it is called a *doubly stochastic matrix*. If P is a doubly stochastic matrix, $\|P\|_1 = 1$, and, of course, $\|P\|_\infty = 1$ and $\rho(P) = 1$.

A permutation matrix is a doubly stochastic matrix; in fact, it is the simplest and one of the most commonly encountered doubly stochastic matrices. A permutation matrix is clearly reducible.

Stochastic matrices are particularly interesting because of their use in defining a discrete homogeneous Markov chain. In that application, a stochastic matrix and *distribution vectors* play key roles. A distribution vector is a nonnegative matrix whose elements sum to 1; that is, a vector v such that $1^{\mathrm{T}}v = 1$. In Markov chain models, the stochastic matrix is a probability transition matrix from a distribution at time t, π_t, to the distribution at time $t+1$,

$$\pi_{t+1} = P\pi_t.$$

In Sect. 9.8.1, we define some basic properties of Markov chains. Those properties depend in large measure on whether the transition matrix is reducible or not.

8.7.5 Leslie Matrices

Another type of nonnegative transition matrix, often used in population studies, is a *Leslie matrix*, after P. H. Leslie, who used it in models in demography. A Leslie matrix is a matrix of the form

$$\begin{bmatrix} \alpha_1 & \alpha_2 & \cdots & \alpha_{m-1} & \alpha_m \\ \sigma_1 & 0 & \cdots & 0 & 0 \\ 0 & \sigma_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_{m-1} & 0 \end{bmatrix}, \tag{8.85}$$

where all elements are nonnegative, and additionally $\sigma_i \leq 1$.

A Leslie matrix is clearly reducible. Furthermore, a Leslie matrix has a single unique positive eigenvalue (see Exercise 8.10), which leads to some interesting properties (see Sect. 9.8.2).

8.8 Other Matrices with Special Structures

Matrices of a variety of special forms arise in statistical analyses and other applications. For some matrices with special structure, specialized algorithms can increase the speed of performing a given task considerably. Many tasks involving matrices require a number of computations of the order of n^3, where n is the number of rows or columns of the matrix. For some of the matrices discussed in this section, because of their special structure, the order of computations may be n^2. The improvement from $\mathrm{O}(n^3)$ to $\mathrm{O}(n^2)$ is enough to

make some tasks feasible that would otherwise be infeasible because of the time required to complete them. The collection of papers in Olshevsky (2003) describe various specialized algorithms for the kinds of matrices discussed in this section.

8.8.1 Helmert Matrices

A *Helmert matrix* is a square orthogonal matrix that partitions sums of squares. Its main use in statistics is in defining contrasts in general linear models to compare the second level of a factor with the first level, the third level with the average of the first two, and so on. (There is another meaning of "Helmert matrix" that arises from so-called Helmert transformations used in geodesy.)

For example, a partition of the sum $\sum_{i=1}^{n} y_i^2$ into orthogonal sums each involving $\bar{y}_k^2$ and $\sum_{i=1}^{k}(y_i - \bar{y}_k)^2$ is

$$\begin{aligned} \tilde{y}_i &= (i(i+1))^{-1/2}\left(\sum_{j=1}^{i+1} y_j - (i+1)y_{i+1}\right) \quad \text{for } i = 1, \ldots n-1, \\ \tilde{y}_n &= n^{-1/2}\sum_{j=1}^{n} y_j. \end{aligned} \tag{8.86}$$

These expressions lead to a computationally stable one-pass algorithm for computing the sample variance (see equation (10.8) on page 504).

The Helmert matrix that corresponds to this partitioning has the form

$$\begin{aligned} H_n &= \begin{bmatrix} 1/\sqrt{n} & 1/\sqrt{n} & 1/\sqrt{n} & \cdots & 1/\sqrt{n} \\ 1/\sqrt{2} & -1/\sqrt{2} & 0 & \cdots & 0 \\ 1/\sqrt{6} & 1/\sqrt{6} & -2/\sqrt{6} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{1}{\sqrt{n(n-1)}} & \frac{1}{\sqrt{n(n-1)}} & \frac{1}{\sqrt{n(n-1)}} & \cdots & -\frac{(n-1)}{\sqrt{n(n-1)}} \end{bmatrix} \\ &= \begin{bmatrix} 1/\sqrt{n}\,1_n^{\mathrm{T}} \\ K_{n-1} \end{bmatrix}, \end{aligned} \tag{8.87}$$

where K_{n-1} is the $(n-1) \times n$ matrix below the first row. For the full n-vector y, we have

$$\begin{aligned} y^{\mathrm{T}} K_{n-1}^{\mathrm{T}} K_{n-1} y &= \sum (y_i - \bar{y})^2 \\ &= \sum (y_i - \bar{y})^2 \\ &= (n-1)s_y^2. \end{aligned}$$

The rows of the matrix in equation (8.87) correspond to *orthogonal contrasts* in the analysis of linear models (see Sect. 9.3.2).

Obviously, the sums of squares are never computed by forming the Helmert matrix explicitly and then computing the quadratic form, but the computations in partitioned Helmert matrices are performed indirectly in analysis of variance, and representation of the computations in terms of the matrix is often useful in the analysis of the computations.

8.8.2 Vandermonde Matrices

A *Vandermonde matrix* is an $n \times m$ matrix with columns that are defined by monomials,

$$V_{n\times m} = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{m-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{m-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^{m-1} \end{bmatrix}, \tag{8.88}$$

where $x_i \neq x_j$ if $i \neq j$. The Vandermonde matrix arises in polynomial regression analysis. For the model equation $y_i = \beta_0 + \beta_1 x_i + \cdots + \beta_p x_i^p + \epsilon_i$, given observations on y and x, a Vandermonde matrix is the matrix in the standard representation $y = X\beta + \epsilon$.

Because of the relationships among the columns of a Vandermonde matrix, computations for polynomial regression analysis can be subject to numerical errors, and so sometimes we make transformations based on orthogonal polynomials. The condition number (see Sect. 6.1, page 266) for a Vandermonde matrix is large. A Vandermonde matrix, however, can be used to form simple orthogonal vectors that correspond to orthogonal polynomials. For example, if the xs are chosen over a grid on $[-1, 1]$, a QR factorization (see Sect. 5.8 on page 248) yields orthogonal vectors that correspond to Legendre polynomials. These vectors are called discrete Legendre polynomials. Although not used in regression analysis so often now, orthogonal vectors are useful in selecting settings in designed experiments.

Vandermonde matrices also arise in the representation or approximation of a probability distribution in terms of its moments.

The determinant of a square Vandermonde matrix has a particularly simple form (see Exercise 8.11).

8.8.3 Hadamard Matrices and Orthogonal Arrays

In a wide range of applications, including experimental design, cryptology, and other areas of combinatorics, we often encounter matrices whose elements are chosen from a set of only a few different elements. In experimental design, the elements may correspond to the levels of the factors; in cryptology, they may represent the letters of an alphabet. In two-level factorial designs, the entries may be either 0 or 1. Matrices all of whose entries are either 1 or

-1 can represent the same layouts, and such matrices may have interesting mathematical properties.

An $n \times n$ matrix with $-1, 1$ entries whose determinant is $n^{n/2}$ is called a *Hadamard matrix*. Hadamard's name is associated with this matrix because of the bound derived by Hadamard for the determinant of any matrix A with $|a_{ij}| \leq 1$ for all i, j: $|\det(A)| \leq n^{n/2}$. A Hadamard matrix achieves this upper bound. A maximal determinant is often used as a criterion for a good experimental design.

We often denote an $n \times n$ Hadamard matrix by H_n, which is the same notation often used for a Helmert matrix, but in the case of Hadamard matrices, the matrix is not unique. All rows are orthogonal and so are all columns. The norm of each row or column is n, so

$$H_n^{\mathrm{T}} H_n = H_n H_n^{\mathrm{T}} = nI. \tag{8.89}$$

It is clear that if H_n is a Hadamard matrix then so is H_n^{T}, and a Hadamard matrix is a normal matrix (page 345). Symmetric Hadamard matrices are often of special interest. In a special type of $n \times n$ Hadamard matrix, one row and one column consist of all 1s; all $n - 1$ other rows and columns consist of $n/2$ 1s and $n/2$ -1s. Such a matrix is called a *normalized* Hadamard matrix. Most Hadamard matrices occurring in statistical applications are normalized, and also most have all 1s on the diagonal.

A Hadamard matrix is often represented as a mosaic of black and white squares, as in Fig. 8.7.

$$\begin{bmatrix} 1 & -1 & -1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

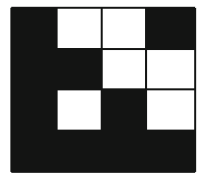


Figure 8.7. A 4×4 Hadamard matrix

Hadamard matrices do not exist for all n. Clearly, n must be even because $|H_n| = n^{n/2}$, but some experimentation (or an exhaustive search) quickly shows that there is no Hadamard matrix for $n = 6$. It has been conjectured, but not proven, that Hadamard matrices exist for any n divisible by 4. Given any $n \times n$ Hadamard matrix, H_n, and any $m \times m$ Hadamard matrix, H_m, an $nm \times nm$ Hadamard matrix can be formed as a partitioned matrix in which each 1 in H_n is replaced by the block submatrix H_m and each -1 is replaced by the block submatrix $-H_m$. For example, the 4×4 Hadamard matrix shown in Fig. 8.7 is formed using the 2×2 Hadamard matrix

$$\begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}$$

as both H_n and H_m. Since an H_2 exists, this means that for any $n = 2^k$, an $n \times n$ Hadamard matrix exists. Not all Hadamard matrices can be formed from other Hadamard matrices in this way, however; that is, they are not necessarily $2^k \times 2^k$. A Hadamard matrix exists for $n = 12$, for example.

A related type of orthogonal matrix, called a *conference matrix*, is a hollow matrix C_n with $-1, 1$ entries on the off-diagonals, and such that

$$C_n^{\mathrm{T}} C_n = (n-1)I. \tag{8.90}$$

A conference matrix is said to be *normalized* if the first row and the first column consist of all 1s, except for the $(1,1)$ element. Conference matrices arise in circuit design and other applications of graph theory. They are related to the adjacency matrices occurring in such applications. The $(n-1) \times (n-1)$ matrix formed by removing the first row and first column of a symmetric conference matrix is a Seidel adjacency matrix (page 334).

A somewhat more general type of matrix corresponds to an $n \times m$ array with the elements in the j^{th} column being members of a set of k_j elements and such that, for some fixed $p \leq m$, in every $n \times p$ submatrix all possible combinations of the elements of the m sets occur equally often as a row. (I make a distinction between the *matrix* and the *array* because often in applications the elements in the array are treated merely as symbols without the assumptions of an algebra of a field. A terminology for orthogonal arrays has evolved that is different from the terminology for matrices; for example, a *symmetric* orthogonal array is one in which $k_1 = \cdots = k_m$. On the other hand, treating the orthogonal arrays as matrices with real elements may provide solutions to combinatorial problems such as may arise in optimal design.)

The 4×4 Hadamard matrix shown in Fig. 8.7 is a symmetric orthogonal array with $k_1 = \cdots = k_4 = 2$ and $p = 4$, so in the array each of the possible combinations of elements occurs exactly once. This array is a member of a simple class of symmetric orthogonal arrays that has the property that in any two rows each ordered pair of elements occurs exactly once.

Orthogonal arrays are particularly useful in developing fractional factorial plans. (The robust designs of Taguchi correspond to orthogonal arrays.) Dey and Mukerjee (1999) discuss orthogonal arrays with an emphasis on the applications in experimental design, and Hedayat et al. (1999) provide extensive an discussion of the properties of orthogonal arrays.

8.8.4 Toeplitz Matrices

A *Toeplitz matrix* is a square matrix with constant codiagonals:

$$\begin{bmatrix} d & u_1 & u_2 & \cdots & u_{n-1} \\ l_1 & d & u_1 & \cdots & u_{n-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{n-2} & l_{n-3} & l_{n-4} & \ddots & u_1 \\ l_{n-1} & l_{n-2} & l_{n-3} & \cdots & d \end{bmatrix}. \tag{8.91}$$

An $n \times n$ Toeplitz matrix is characterized by a diagonal element d and two $(n-1)$-vectors, u and l, as indicated in the expression above. If $u = 0$, the Toeplitz matrix is lower triangular; if $l = 0$, the matrix is upper triangular; and if $u = l$, the matrix is symmetric. A Toeplitz matrix is a banded matrix, but it may or may not be a "band matrix", that is, one with many 0 codiagonals, which we discuss in Chaps. 3 and 12.

Banded Toeplitz matrices arise frequently in time series studies. The covariance matrix in an ARMA(p, q) process, for example, is a symmetric Toeplitz matrix with $2\max(p, q)$ nonzero off-diagonal bands. See page 451 for an example and further discussion.

8.8.4.1 Inverses of Certain Toeplitz Matrices and Other Banded Matrices

A Toeplitz matrix that occurs often in stationary time series is the $n \times n$ variance-covariance matrix of the form

$$V = \sigma^2 \begin{bmatrix} 1 & \rho & \rho^2 & \cdots & \rho^{n-1} \\ \rho & 1 & \rho & \cdots & \rho^{n-2} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \rho^{n-1} & \rho^{n-2} & \rho^{n-3} & \cdots & 1 \end{bmatrix}.$$

It is easy to see that V^{-1} exists if $\sigma \neq 0$ and $|\rho| < 1$, and that it is the type 2 matrix

$$V^{-1} = \frac{1}{(1-\rho^2)\sigma^2} \begin{bmatrix} 1 & -\rho & 0 & \cdots & 0 \\ -\rho & 1+\rho^2 & -\rho & \cdots & 0 \\ 0 & -\rho & 1+\rho^2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix}. \tag{8.92}$$

Type 2 matrices also occur as the inverses of other matrices with special patterns that arise in other common statistical applications (see Graybill 1983; for examples).

The inverses of all banded invertible matrices have some off-diagonal submatrices that are zero or have low rank, depending on the bandwidth of the original matrix (see Strang and Nguyen 2004, for further discussion and examples).

8.8.5 Circulant Matrices

A Toeplitz matrix having the form

$$\begin{bmatrix} c_1 & c_2 & c_3 & \cdots & c_{n-1} & c_n \\ c_n & c_1 & c_2 & \cdots & c_{n-2} & c_{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ c_3 & c_4 & c_5 & \cdots & c_1 & c_2 \\ c_2 & c_3 & c_4 & \cdots & c_n & c_1 \end{bmatrix} \tag{8.93}$$

is called a *circulant* matrix. Beginning with a fixed first row, each subsequent row is obtained by a right circular shift of the row just above it.

A useful $n \times n$ circulant matrix is the permutation matrix

$$E_{(n,1,2,\ldots,n-1)} = \prod_{k=2}^{n} E_{k,k-1}, \tag{8.94}$$

which is the identity transformed by moving each row downward into the row below it and the last row into the first row. (See page 82.) Let us denote this permutation as π_c; hence, we denote the elementary circulant matrix in (8.94) as $E_{(\pi_c)}$.

By first recalling that, for any permutation matrix, $E_{(\pi)}^{-1} = E_{(\pi)}^{\mathrm{T}}$, and then considering the effects of the multiplications $AE_{(\pi)}$ and $E_{(\pi)}A$, it is easy to see that A is circulant if and only if

$$A = E_{(\pi_c)} A E_{(\pi_c)}^{\mathrm{T}}. \tag{8.95}$$

Circulant matrices have several straightforward properties. If A is circulant, then

- A^{T} is circulant
- A^2 is circulant
- if A is nonsingular, then A^{-1} is circulant

You are asked to prove these simple results in Exercise 8.13.

Any linear combination of two circulant matrices of the same order is circulant (that is, they form a vector space, see Exercise 8.14).

If A and B are circulant matrices of the same order, then AB is circulant (Exercise 8.15).

Another important property of a circulant matrix is that it is normal, as we can see by writing the (i, j) element of AA^{T} and of $A^{\mathrm{T}}A$ as a sum of products of elements of A (Exercise 8.16). This has an important implication: a circulant matrix is unitarily diagonalizable.

8.8.6 Fourier Matrices and the Discrete Fourier Transform

A special Vandermonde matrix (equation (8.88)) is an $n \times n$ matrix whose entries are the n^{th} roots of unity, that is $\{1, \omega, \omega^2, \ldots, \omega^{n-1}\}$, where

$$\omega = \mathrm{e}^{2\pi \mathrm{i}/n} = \cos\left(\frac{2\pi}{n}\right) + \mathrm{i}\sin\left(\frac{2\pi}{n}\right). \tag{8.96}$$

The matrix is called a *Fourier matrix*. The (j,k) entry of a Fourier matrix is $\omega^{(j-1)(k-1)}/\sqrt{n}$:

$$F_n = \frac{1}{\sqrt{n}} \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^1 & \omega^2 & \omega^3 & \cdots & \omega^{(n-1)} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \omega^{3(n-1)} & \cdots & \omega^{(n-1)(n-1)} \end{bmatrix}, \tag{8.97}$$

(The Fourier matrix is sometimes defined with a negative sign in the exponents; that is, such that the $(j,k)^{\text{th}}$ entry is $\omega^{-(j-1)(k-1)}/\sqrt{n}$. The normalizing factor $1/\sqrt{n}$ is also sometimes omitted. In fact, in many applications of Fourier matrices and various Fourier forms, there is inconsequential, but possibly annoying, variation in the notation.)

Notice that the Fourier matrix is symmetric, and any entry raised to the n^{th} power is 1. Although the Fourier matrix is symmetric, its eigenvalues are not necessarily real, because it itself is not a real matrix.

Fourier matrices whose order is a power of 2 tend to have a propensity of elements that are either ± 1 or $\pm \mathrm{i}$. For example, the 4×4 Fourier matrix is

$$F_4 = \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \mathrm{i} & -1 & -\mathrm{i} \\ 1 & -1 & 1 & -1 \\ 1 & -\mathrm{i} & -1 & \mathrm{i} \end{bmatrix}.$$

(Recall that there are different definitions of the elements of a Fourier matrix; for the 4×4, they all are as shown above, but the patterns of positive and negative values may be different. Most of the elements of the 8×8 Fourier matrix are either ± 1 or $\pm \mathrm{i}$. The 16 that are not are $\pm 1/\sqrt{2} \pm \mathrm{i}/\sqrt{2}$.

The Fourier matrix has many useful properties, such as being unitary; that is, row and columns are orthonormal: $f_{*j}^{\mathrm{H}} f_{*k} = f_{j*}^{\mathrm{H}} f_{k*} = 0$ for $j \neq k$, and $f_{*j}^{\mathrm{H}} f_{*j} = 1$ (Exercise 8.17).

The most interesting feature of the Fourier matrix is its relationship to the Fourier transform. For an integrable function $f(x)$, the Fourier transform is

$$\mathcal{F}f(s) = \int_{-\infty}^{\infty} \mathrm{e}^{-2\pi \mathrm{i} s x} f(x)\, \mathrm{d}x. \tag{8.98}$$

(Note that the characteristic function in probability theory is this same transform applied to a probability density function, with argument $t = -2\pi s$.)

8.8.6.1 Fourier Matrices and Elementary Circulant Matrices

The Fourier matrix and the elementary circulant matrix $E_{(\pi_c)}$ of corresponding order are closely related. Being a normal matrix, $E_{(\pi_c)}$ is unitarily diagonalizable, and the Fourier matrix and its conjugate transpose are the diagonalizing matrices, and the diagonal matrix itself, that is, the eigenvalues, are elements of the Fourier matrix:

$$E_{(\pi_c)} = F_n^{\mathrm{H}} \mathrm{diag}((1, \omega, \omega^2, \ldots, \omega^{n-1}))F_n. \tag{8.99}$$

This is easy to see by performing the multiplications on the right side of the equation, and you are asked to do this in Exercise 8.18.

We see that the eigenvalues of $E_{(\pi_c)}$ are what we would expect if we continued the development from page 137 where we determined the eigenvalues of an elementary permutation matrix. (An elementary permutation matrix of order 2 has the two eigenvalues $\sqrt{1}$ and $-\sqrt{1}$.)

Notice that the modulus of all eigenvalues of $E_{(\pi_c)}$ is the same, 1. Hence, all eigenvalues of $E_{(\pi_c)}$ lie on its spectral circle.

8.8.6.2 The Discrete Fourier Transform

Fourier transforms are invertible transformations of functions that often allow operations on those functions to be performed more easily. The Fourier transform shown in equation (8.98) may allow for simpler expressions of convolutions, for example. An n-vector is equivalent to a function whose domain is $\{1, \ldots, n\}$, and Fourier transforms of vectors are useful in various operations on the vectors. More importantly, if the vector represents observational data, certain properties of the data may become immediately apparent in the Fourier transform of the data vector.

The Fourier transform, at a given value s, of the function as shown in equation (8.98) is an integral. The argument of the transform s may or may not range over the same domain as x the argument of the function. The analogue of the Fourier transform of a vector is a sum, again at some given value, which is effectively the argument of the transform. For an n-vector x, the discrete Fourier transform at n points is

$$\mathcal{F}x = F_n x. \tag{8.100}$$

Transformations of this form are widely used in time series, where the vector x contains observations at equally spaced points in time. While the elements of x represent measurements at distinct points in time, the elements of the vector $\mathcal{F}x$ represent values at different points in the period of a sine and/or a cosine curve, as we see from the relation of the roots of unity given in equation (8.96). Many time series, especially those relating to measurement of waves propagating through matter or of vibrating objects, exhibit periodicities, which can be used to distinguish different kinds of waves and possibly

to locate their source. Many waves and other time series have multiple periodicities, at different frequencies. The Fourier transform is sometimes called a frequency filter.

Our purpose here is just to indicate the relation of Fourier transforms to matrices, and to suggest how this might be useful in applications. There is a wealth of literature on Fourier transforms and their applications, but we will not pursue those topics here.

As often when writing expressions involving matrices, we must emphasize that *the form of a mathematical expression and the way the expression should be evaluated in actual practice may be quite different.* This is particularly true in the case of the discrete Fourier transform. *There would never be a reason to form the Fourier matrix for any computations,* but more importantly, rather than evaluating the elements in the Fourier transform $\mathcal{F}x$ using the right side of equation (8.100), we take advantage of properties of the powers of roots of unity to arrive at a faster method of computing them—so much faster, in fact, the method is called the "fast" Fourier transform, or FFT. The FFT is one of the most important computational methods in all of applied mathematics. I will not discuss it further here, however. Descriptions of it can be found throughout the literature on computational science (including other books that I have written).

An Aside: Complex Matrices

The Fourier matrix, I believe, is the only matrix I discuss in this book that has complex entries. The purpose is just to relate the discrete Fourier transform to matrix multiplication. We have already indicated various differences in operations on vectors and matrices over the complex plane. We often form the *conjugate* of a complex number, which we denote by an overbar: $\bar{z}$. The conjugate of an object such as a vector or a matrix is the result of taking the conjugate of each element individually. Instead of a transpose, we usually work with a *conjugate transpose*, $A^{\mathrm{H}} = \overline{A}^{\mathrm{T}}$. (See the discussion on pages 33 and 60.) We define the *inner product* of vectors x and y as $\langle x, y\rangle = \bar{x}^{\mathrm{T}}y$; instead of symmetric matrices, we focus on *Hermitian* matrices, that is, ones such that the conjugate transpose is the same as the original matrices, and instead of orthogonal matrices, we focus on *unitary* matrices, that is, ones whose product with its conjugate transpose is the identity. All of the general results that we have stated for real matrices of course hold for complex matrices. Some of the analogous operations, however, have different properties. For example, the property of a matrix being unitarily diagonalizable is an analogue of the property of being orthogonally diagonalizable, but, as we have seen, a wider class of matrices (normal matrices) are unitarily diagonalizable than those that are orthogonally diagonalizable (symmetric matrices).

Most scientific software systems support computations with complex numbers. Both R and Matlab, for example, provide full capabilities for

working with complex numbers. The imaginary unit in both is denoted by "`i`", which of course must be distinguished from "`i`" denoting a variable. In R, the function `complex` can be used to initialize a complex number; for example,

```
z<-complex(re=3,im=2)
```

assigns the value $3+2\mathrm{i}$ to the variable `z`. (Another simple way of doing this is to juxtapose a numeric literal in front of the symbol `i`; in R, for example, `z<-3+2i` assigns the same value to `z`. I do not recommend the latter construct because of possible confusion with other expressions.) The j^{th} row of an n^{th} order Fourier matrix can be generated by the R expression

```
c(1,exp(2*pi*complex(im=seq(j,j*(n-1),j)/n)))
```

As mentioned above, there would almost never be a reason to form the Fourier matrix for any computations. It is instructive, however, to note its form and to do some simple manipulations with the Fourier matrix in R or some other software system. The Matlab function `dftmtx` generates a Fourier matrix (with a slightly different definition, resulting in a different pattern of positives and negatives than what I have shown above).
Some additional R code for manipulating complex matrices is given in the hints for Exercise 8.18 on page 611. (Note that there I do form a Fourier matrix and use it multiplications; but it is just for illustration.)

8.8.7 Hankel Matrices

A *Hankel matrix* is a square matrix with constant "anti"-codiagonals:

$$\begin{bmatrix} u_{n-1} & u_{n-2} & \cdots & u_1 & d \\ u_{n-2} & u_{n-3} & \cdots & d & l_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ u_1 & d & l_1 & \ddots & l_{n-2} \\ d & l_1 & l_2 & \cdots & l_{n-1} \end{bmatrix}. \tag{8.101}$$

Notice that a Hankel matrix is similar to a Toeplitz matrix, except that the axial diagonal is not the principal diagonal; rather, for an $n \times n$ Hankel matrix H, it is the "anti-diagonal", consisting of the elements $h_{1,n}, h_{2,n-1}, \ldots, h_{n,1}$. Although this "diagonal" is visually apparent, and symmetries about it are intuitively obvious, there is no commonly-used term for this diagonal or for symmetries about it. We can also observe analogues of lower triangular, upper triangular, and symmetric matrices based on $u = 0$, $l = 0$, and $u = l$; but these are not the kinds of matrices that we have identified with these terms. Words such as "anti" and "skew" are sometimes used to qualify names

of these properties or objects. A triangular counterpart is sometimes called "skew ... diagonal". There is no standard term for symmetry about the anti-diagonal, however. The term "skew symmetric" has already been taken. (It is a matrix A with $a_{ij} = -a_{ji}$.)

As with a Toeplitz matrix, a Hankel matrix is characterized by a "diagonal" element and two vectors, and can be generalized to $n \times m$ matrices based on vectors of different orders. As in the expression (8.101) above, an $n \times n$ Toeplitz matrix can be defined by a scalar d and two $(n-1)$-vectors, u and l. There are other, perhaps more common, ways of putting the elements of a Hankel matrix into two vectors. In Matlab, the function `hankel` produces a Hankel matrix, but the elements are specified in a different way from that in expression (8.101).

A common form of Hankel matrix is an $n \times n$ skew upper triangular matrix, and it is formed from the u vector only. The simplest form of the square skew upper triangular Hankel matrix is formed from the vector $u = (n-1, n-2, \ldots, 1)$ and $d = n$:

$$\begin{bmatrix} 1 & 2 & 3 & \cdots & n \\ 2 & 3 & 4 & \cdots & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ n & 0 & 0 & \cdots & 0 \end{bmatrix}. \tag{8.102}$$

A skew upper triangular Hankel matrix occurs in the spectral analysis of time series. If $x(t)$ is a (discrete) time series, for $t = 0, 1, 2, \ldots$, the Hankel matrix of the time series has as the (i, j) element

$$\begin{array}{ll} x(i+j-2) & \text{if } i+j-1 \leq n, \\ 0 & \text{otherwise.} \end{array}$$

The L_2 norm of the Hankel matrix of the time series is called the *Hankel norm* of the frequency filter response (the Fourier transform).

8.8.8 Cauchy Matrices

Another type of special $n \times m$ matrix whose elements are determined by a few n-vectors and m-vectors is a Cauchy-type matrix. The standard Cauchy matrix is built from two vectors, x and y. The more general form defined below uses two additional vectors.

A *Cauchy matrix* is an $n \times m$ matrix $C(x, y, v, w)$ generated by n-vectors x and v and m-vectors y and w of the form

$$C(x, y, v, w) = \begin{bmatrix} \frac{v_1 w_1}{x_1 - y_1} & \cdots & \frac{v_1 w_m}{x_1 - y_m} \\ \vdots & \cdots & \vdots \\ \frac{v_n w_1}{x_n - y_1} & \cdots & \frac{v_n w_m}{x_n - y_m} \end{bmatrix}. \tag{8.103}$$

Cauchy-type matrices often arise in the numerical solution of partial differential equations (PDEs). For Cauchy matrices, the order of the number of computations for factorization or solutions of linear systems can be reduced from a power of three to a power of two. This is a very significant improvement for large matrices. In the PDE applications, the matrices are generally not large, but nevertheless, even in those applications, it worthwhile to use algorithms that take advantage of the special structure. Fasino and Gemignani (2003) describe such an algorithm.

8.8.9 Matrices Useful in Graph Theory

Many problems in statistics and applied mathematics can be posed as graphs, and various methods of graph theory can be used in their solution.

Graph theory is particularly useful in cluster analysis or classification. These involve the analysis of relationships of objects for the purpose of identifying similar groups of objects. The objects are associated with vertices of the graph, and an edge is generated if the relationship (measured somehow) between two objects is sufficiently great. For example, suppose the question of interest is the authorship of some text documents. Each document is a vertex, and an edge between two vertices exists if there are enough words in common between the two documents. A similar application could be the determination of which computer user is associated with a given computer session. The vertices would correspond to login sessions, and the edges would be established based on the commonality of programs invoked or files accessed. In applications such as these, there would typically be a training dataset consisting of text documents with known authors or consisting of session logs with known users. In both of these types of applications, decisions would have to be made about the extent of commonality of words, phrases, programs invoked, or files accessed in order to establish an edge between two documents or sessions.

Unfortunately, as is often the case for an area of mathematics or statistics that developed from applications in diverse areas or through the efforts of applied mathematicians somewhat outside of the mainstream of mathematics, there are major inconsistencies in the notation and terminology employed in graph theory. Thus, we often find different terms for the same object; for example, adjacency matrix and connectivity matrix. This unpleasant situation, however, is not so disagreeable as a one-to-many inconsistency, such as the designation of the eigenvalues of a graph to be the eigenvalues of one type of matrix in some of the literature and the eigenvalues of different types of matrices in other literature.

Refer to Sect. 8.1.2 beginning on page 331 for terms and notation that we will use in the following discussion.

8.8.9.1 Adjacency Matrix: Connectivity Matrix

We discussed adjacency or connectivity matrices on page 334. A matrix, such as an adjacency matrix, that consists of only 1s and 0s is called a *Boolean matrix*.

Two vertices that are not connected and hence correspond to a 0 in a connectivity matrix are said to be *independent*.

If no edges connect a vertex with itself, the adjacency matrix is a hollow matrix.

Because the 1s in a connectivity matrix indicate a strong association, and we would naturally think of a vertex as having a strong association with itself, we sometimes modify the connectivity matrix so as to have 1s along the diagonal. Such a matrix is sometimes called an *augmented connectivity matrix* or *augmented adjacency matrix*.

The eigenvalues of the adjacency matrix reveal some interesting properties of the graph and are sometimes called the eigenvalues of the graph. The eigenvalues of another matrix, which we discuss below, are more useful, however, and we will refer to those eigenvalues as the eigenvalues of the graph.

8.8.9.2 Digraphs

The digraph represented in Fig. 8.4 on page 335 is a network with five vertices, perhaps representing cities, and directed edges between some of the vertices. The edges could represent airline connections between the cities; for example, there are flights from x to u and from u to x, and from y to z, but not from z to y.

In a digraph, the relationships are directional. (An example of a directional relationship that might be of interest is when each observational unit has a different number of measured features, and a relationship exists from v_i to v_j if a majority of the features of v_i are identical to measured features of v_j.)

8.8.9.3 Use of the Connectivity Matrix

The analysis of a network may begin by identifying which vertices are connected with others; that is, by construction of the connectivity matrix.

The connectivity matrix can then be used to analyze other levels of association among the data represented by the graph or digraph. For example, from the connectivity matrix in equation (8.2) on page 335, we have

$$A^2 = \begin{bmatrix} 4 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 2 \\ 1 & 2 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

In terms of the application suggested on page 335 for airline connections, the matrix A^2 represents the number of connections between the cities that consist of exactly two flights. From A^2 we see that there are two ways to go from city y to city w in just two flights but only one way to go from w to y in two flights.

This property extends to multiple connections. If A is the adjacency matrix of a graph, then A^k_{ij} is the number of paths of length k between nodes i and j in that graph. (See Exercise 8.20.) Important areas of application of this fact are in DNA sequence comparisons and measuring "centrality" of a node within a complex social network.

8.8.9.4 The Laplacian Matrix of a Graph

Spectral graph theory is concerned with the analysis of the eigenvalues of a graph. As mentioned above, there are two different definitions of the eigenvalues of a graph. The more useful definition, and the one we use here, takes the eigenvalues of a graph to be the eigenvalues of a matrix, called the Laplacian matrix, formed from the adjacency matrix and a diagonal matrix consisting of the degrees of the vertices.

Given the graph $\mathcal{G}$, let $D(\mathcal{G})$ be a diagonal matrix consisting of the degrees of the vertices of $\mathcal{G}$ (that is, $D(\mathcal{G}) = \text{diag}(d(\mathcal{G}))$) and let $C(\mathcal{G})$ be the adjacency matrix of $\mathcal{G}$. If there are no isolated vertices (that is if $d(\mathcal{G}) > 0$), then the *Laplacian matrix* of the graph, $L(\mathcal{G})$ is given by

$$L(\mathcal{G}) = I - D(\mathcal{G})^{-\frac{1}{2}} C(\mathcal{G}) D(\mathcal{G})^{-\frac{1}{2}}. \tag{8.104}$$

Some authors define the Laplacian in other ways:

$$L_a(\mathcal{G}) = I - D(\mathcal{G})^{-1} C(\mathcal{G}) \tag{8.105}$$

or

$$L_b(\mathcal{G}) = D(\mathcal{G}) - C(\mathcal{G}). \tag{8.106}$$

The eigenvalues of the Laplacian matrix are the *eigenvalues of a graph*. The definition of the Laplacian matrix given in equation (8.104) seems to be more useful in terms of bounds on the eigenvalues of the graph. The set of unique eigenvalues (the spectrum of the matrix L) is called the spectrum of the graph.

So long as $d(\mathcal{G}) > 0$, $L(\mathcal{G}) = D(\mathcal{G})^{-\frac{1}{2}} L_a(\mathcal{G}) D(\mathcal{G})^{-\frac{1}{2}}$. Unless the graph is regular, the matrix $L_b(\mathcal{G})$ is not symmetric. Note that if $\mathcal{G}$ is k-regular, $L(\mathcal{G}) = I - C(\mathcal{G})/k$, and $L_b(\mathcal{G}) = L(\mathcal{G})$.

For a digraph, the degrees are replaced by either the indegrees or the outdegrees. (Some authors define it one way and others the other way. The essential properties hold either way.)

The Laplacian can be viewed as an operator on the space of functions $f : V(\mathcal{G}) \to \mathbb{R}$ such that for the vertex v

$$L(f(v)) = \frac{1}{\sqrt{d_v}} \sum_{w, w \sim v} \left(\frac{f(v)}{\sqrt{d_v}} - \frac{f(w)}{\sqrt{d_w}} \right),$$

where $w \sim v$ means vertices w and v that are adjacent, and d_u is the degree of the vertex u.

For a symmetric graph, the Laplacian matrix is symmetric, so its eigenvalues are all real. We can see that the eigenvalues are all nonnegative by forming the Rayleigh quotient (equation (3.266)) using an arbitrary vector g, which can be viewed as a real-valued function over the vertices,

$$\begin{aligned} R_L(g) &= \frac{\langle g, Lg \rangle}{\langle g, g \rangle} \\ &= \frac{\langle g, D^{-\frac{1}{2}} L_a D^{-\frac{1}{2}} g \rangle}{\langle g, g \rangle} \\ &= \frac{\langle f, L_a f \rangle}{\langle D^{\frac{1}{2}} f, D^{\frac{1}{2}} f \rangle} \\ &= \frac{\sum_{v \sim w} (f(v) - f(w))^2}{f^{\mathrm{T}} D f}, \end{aligned} \tag{8.107}$$

where $f = D^{-\frac{1}{2}} g$, and $f(u)$ is the element of the vector corresponding to vertex u. Because the Raleigh quotient is nonnegative, all eigenvalues are nonnegative, and because there is an $f \neq 0$ for which the Rayleigh quotient is 0, we see that 0 is an eigenvalue of a graph. Furthermore, using the Cauchy-Schwartz inequality, we see that the spectral radius is less than or equal to 2.

The eigenvalues of a matrix are the basic objects in spectral graph theory. They provide information about the properties of networks and other systems modeled by graphs. We will not explore them further here, and the interested reader is referred to Bollobás (2013) or other general texts on the subject.

If $\mathcal{G}$ is the graph represented in Fig. 8.2 on page 332, with $V(\mathcal{G}) = \{a, b, c, d, e\}$, the degrees of the vertices of the graph are $d(\mathcal{G}) = (4, 2, 2, 3, 3)$. Using the adjacency matrix given in equation (8.1), we have

$$L(\mathcal{G}) = \begin{bmatrix} 1 & -\frac{\sqrt{2}}{4} & -\frac{\sqrt{2}}{4} & -\frac{\sqrt{3}}{6} & -\frac{\sqrt{3}}{6} \\ -\frac{\sqrt{2}}{4} & 1 & 0 & 0 & -\frac{\sqrt{6}}{6} \\ -\frac{\sqrt{2}}{4} & 0 & 1 & -\frac{\sqrt{6}}{6} & 0 \\ -\frac{\sqrt{3}}{6} & 0 & -\frac{\sqrt{6}}{6} & 1 & -\frac{1}{3} \\ -\frac{\sqrt{3}}{6} & -\frac{\sqrt{6}}{6} & 0 & -\frac{1}{3} & 1 \end{bmatrix}. \tag{8.108}$$

This matrix is singular, and the unnormalized eigenvector corresponding to the 0 eigenvalue is $(2\sqrt{14}, 2\sqrt{7}, 2\sqrt{7}, \sqrt{42}, \sqrt{42})$.

8.8.10 Z-Matrices and M-Matrices

In certain applications in physics and in the solution of systems of nonlinear differential equations, a class of matrices called M-matrices is important.

The matrices in these applications have nonpositive off-diagonal elements. A square matrix all of whose off-diagonal elements are nonpositive is called a Z-matrix.

A Z-matrix that is positive stable (see page 159) is called an M-matrix. A real symmetric M-matrix is positive definite.

In addition to the properties that constitute the definition, M-matrices have a number of remarkable properties, which we state here without proof. If A is a real M-matrix, then

- all principal minors of A are positive;
- all diagonal elements of A are positive;
- all diagonal elements of L and U in the LU decomposition of A are positive;
- for some i, $\sum_j a_{ij} \geq 0$; and
- A is nonsingular and $A^{-1} \geq 0$.

Proofs of some of these facts can be found in Horn and Johnson (1991).

Exercises

8.1. Normal matrices.
  a) Show that a skew symmetric matrix is normal.
  b) Show that a skew Hermitian matrix is normal.

8.2. Ordering of nonnegative definite matrices.
  a) A relation $\bowtie$ on a set is a *partial ordering* if, for elements a, b, and c,
    - it is reflexive: $a \bowtie a$;
    - it is antisymmetric: $a \bowtie b \bowtie a \Longrightarrow a = b$; and
    - it is transitive: $a \bowtie b \bowtie c \Longrightarrow a \bowtie c$.

    Show that the relation $\succeq$ (equation (8.19)) is a partial ordering.
  b) Show that the relation $\succ$ (equation (8.20)) is transitive.

8.3. a) Show that a (strictly) diagonally dominant symmetric matrix is positive definite.
  b) Show that if the real $n \times n$ symmetric matrix A is such that

$$a_{ii} \geq \sum_{j \neq i}^{n} |a_{ij}| \quad \text{for each } i = 1, \ldots, n$$

  then $A \succeq 0$.

8.4. Show that the number of positive eigenvalues of an idempotent matrix is the rank of the matrix.

8.5. Show that two idempotent matrices of the same rank are similar.

8.6. Under the given conditions, show that properties (a) and (b) on page 357 imply property (c).

8.7. Projections.

a) Show that the matrix given in equation (8.42) (page 359) is a projection matrix.

b) Write out the projection matrix for projecting a vector onto the plane formed by two vectors, x_1 and x_2, as indicated on page 359, and show that it is the same as the hat matrix of equation (8.52).

8.8. Correlation matrices.

A correlation matrix can be defined in terms of a Gramian matrix formed by a centered and scaled matrix, as in equation (8.69). Sometimes in the development of statistical theory, we are interested in the properties of correlation matrices with given eigenvalues or with given ratios of the largest eigenvalue to other eigenvalues.

Write a program to generate $n \times n$ random correlation matrices R with specified eigenvalues, $c_1, \ldots, c_n$. The only requirements on R are that its diagonals be 1, that it be symmetric, and that its eigenvalues all be positive and sum to n. Use the following method due to Davies and Higham (2000) that uses random orthogonal matrices with the Haar uniform distribution generated using the method described in Exercise 4.10.

0. Generate a random orthogonal matrix Q; set $k = 0$, and form

$$R^{(0)} = Q\text{diag}((c1, \ldots, c_n))Q^{\mathrm{T}}.$$

1. If $r_{ii}^{(k)} = 1$ for all i in $\{1, \ldots, n\}$, go to step 3.
2. Otherwise, choose p and q with $p < j$, such that $r_{pp}^{(k)} < 1 < r_{qq}^{(k)}$ or $r_{pp}^{(k)} > 1 > r_{qq}^{(k)}$, and form $G^{(k)}$ as in equation (5.13), where c and s are as in equations (5.17) and (5.18), with $a = 1$.
   Form $R^{(k+1)} = (G^{(k)})^{\mathrm{T}} R^{(k)} G^{(k)}$.
   Set $k = k + 1$, and go to step 1.
3. Deliver $R = R^{(k)}$.

8.9. Use the relationship (8.77) to prove properties 1 and 4 on page 377.

8.10. Leslie matrices.

a) Write the characteristic polynomial of the Leslie matrix, equation (8.85).

b) Show that the Leslie matrix has a single, unique positive eigenvalue.

8.11. Write out the determinant for an $n \times n$ Vandermonde matrix.

Hint: The determinant of an $n \times n$ Vandermonde matrix as in equation (8.88) is $(x_n - x_1)(x_n - x_2) \cdots (x_n - x_{n-1})$ times the determinant of the $n - 1 \times n - 1$ Vandermonde matrix formed by removing the last row and column. Show this by multiplying the original Vandermonde matrix by $B = I + D$, where D is the matrix with 0s in all positions except for the first supradiagonal, which consists of $-x_n$, replicated $n - 1$ times. Clearly, the determinant of B is 1.

8.12. Consider the 3×3 symmetric Toeplitz matrix with elements a, b, and c; that is, the matrix that looks like this:

$$\begin{bmatrix} 1 & b & c \\ b & 1 & b \\ c & b & 1 \end{bmatrix}.$$

a) Invert this matrix.
See page 385.
b) Determine conditions for which the matrix would be singular.

8.13. If A is circulant, show that
- A^{T} is circulant
- A^2 is circulant
- if A is nonsingular, then A^{-1} is circulant

Hint: Use equation (8.95).

8.14. Show that the set of all $n \times n$ circulant matrices is a vector space along with the axpy operation. (Just show that it is closed with respect to that operation.)

8.15. If A and B are circulant matrices of the same order, show AB is circulant.

8.16. Show that a circulant matrix is normal.

8.17. Show that a Fourier matrix, as in equation (8.97), is unitary by showing $f_{*j}^{\mathrm{H}} f_{*k} = f_{j*}^{\mathrm{H}} f_{k*} = 0$ for $j \neq k$, and $f_{*j}^{\mathrm{H}} f_{*j} = 1$.

8.18. Show that equation (8.99) is correct by performing the multiplications on the right side of the equation.

8.19. Write out the determinant for the $n \times n$ skew upper triangular Hankel matrix in (8.102).

8.20. Graphs. Let A be the adjacency matrix of an undirected graph.

a) Show that A_{ij}^2 is the number of paths of length 2 between nodes i and j.
Hint: Construct a general diagram similar to Fig. 8.2 on page 332, and count the paths between two arbitrary nodes.

b) Show that A_{ij}^k is the number of paths of length k between nodes i and j.
Hint: Use Exercise 8.20a and mathematical induction on k.

9

Selected Applications in Statistics

Data come in many forms. In the broad view, the term "data" embraces all representations of information or knowledge. There is no single structure that can efficiently contain all of these representations. Some data are in free-form text (for example, the Federalist Papers, which was the subject of a famous statistical analysis), other data are in a hierarchical structure (for example, political units and subunits), and still other data are encodings of methods or algorithms. (This broad view is entirely consistent with the concept of a "stored-program computer"; the program is the data.)

Several of the results in this chapter have already been presented in Chap. 8 or even in previous chapters, for example, the smoothing matrix H_λ that we discuss in Sect. 9.3.8 has already been encountered on page 364 in Chap. 8. The purpose of the apparent redundancy is to present the results from a different perspective. (None of the results are new; all are standard in the statistical literature.)

9.1 Structure in Data and Statistical Data Analysis

Data often have a logical structure as described in Sect. 8.1.1; that is, a two-dimensional array in which columns correspond to variables or measurable attributes and rows correspond to an observation on all attributes taken together. A matrix is obviously a convenient object for representing numeric data organized this way. An objective in analyzing data of this form is to uncover relationships among the variables, or to characterize the distribution of the sample over $\mathbb{R}^m$. Interesting relationships and patterns are called "structure" in the data. This is a different meaning from that of the word used in the phrase "logical structure" or in the phrase "data structure" used in computer science.

J.E. Gentle, *Matrix Algebra*, Springer Texts in Statistics,
DOI 10.1007/978-3-319-64867-5_9

Another type of pattern that may be of interest is a temporal pattern; that is, a set of relationships among the data and the time or the sequence in which the data were observed.

The objective of this chapter is to illustrate how some of the properties of matrices and vectors that were covered in previous chapters relate to statistical models and to data analysis procedures. The field of statistics is far too large for a single chapter on "applications" to cover more than just a small part of the area. Similarly, the topics covered previously are too extensive to give examples of applications of all of them.

A probability distribution is a specification of the stochastic structure of random variables, so we begin with a brief discussion of properties of multivariate probability distributions. The emphasis is on the multivariate normal distribution and distributions of linear and quadratic transformations of normal random variables. We then consider an important structure in multivariate data, a linear model. We discuss some of the computational methods used in analyzing the linear model. We then describe some computational method for identifying more general linear structure and patterns in multivariate data. Next we consider approximation of matrices in the absence of complete data. Finally, we discuss some models of stochastic processes. The special matrices discussed in Chap. 8 play an important role in this chapter.

9.2 Multivariate Probability Distributions

Most methods of statistical inference are based on assumptions about some underlying probability distribution of a random variable. In some cases these assumptions completely specify the form of the distribution, and in other cases, especially in nonparametric methods, the assumptions are more general. Many statistical methods in estimation and hypothesis testing rely on the properties of various transformations of a random variable.

In this section, we do not attempt to develop a theory of probability distribution; rather we assume some basic facts and then derive some important properties that depend on the matrix theory of the previous chapters.

9.2.1 Basic Definitions and Properties

One of the most useful descriptors of a random variable is its probability density function (PDF), or probability function. Various functionals of the PDF define standard properties of the random variable, such as the mean and variance, as we discussed in Sect. 4.5.3.

If X is a random variable over $\mathbb{R}^d$ with PDF $p_X(\cdot)$ and $f(\cdot)$ is a measurable function (with respect to a dominating measure of $p_X(\cdot)$) from $\mathbb{R}^d$ to $\mathbb{R}^k$, the *expected value* of $f(X)$, which is in $\mathbb{R}^k$ and is denoted by $\mathrm{E}(g(X))$, is defined by

$$\mathrm{E}(f(X)) = \int_{\mathbb{R}^d} f(t)p_X(t)\,\mathrm{d}t.$$

The *mean* of X is the d-vector $\mathrm{E}(X)$, and the *variance* or *variance-covariance* of X, denoted by $\mathrm{V}(X)$, is the $d \times d$ matrix

$$\mathrm{V}(X) = \mathrm{E}\left((X - \mathrm{E}(X))\,(X - \mathrm{E}(X))^{\mathrm{T}}\right).$$

Given a random variable X, we are often interested in a random variable defined as a function of X, say $Y = g(X)$. To analyze properties of Y, we identify g^{-1}, which may involve another random variable. (For example, if $g(x) = x^2$ and the support of X is $\mathbb{R}$, then $g^{-1}(Y) = (-1)^{\alpha}\sqrt{Y}$, where $\alpha = 1$ with probability $\Pr(X < 0)$ and $\alpha = 0$ otherwise.) Properties of Y can be evaluated using the Jacobian of $g^{-1}(\cdot)$, as in equation (4.12).

9.2.2 The Multivariate Normal Distribution

The most important multivariate distribution is the multivariate normal, which we denote as $\mathrm{N}_d(\mu, \Sigma)$ for d dimensions; that is, for a random d-vector. The PDF for the d-variate normal distribution, as we have discussed before, is

$$p_X(x) = (2\pi)^{-d/2}|\Sigma|^{-1/2}\mathrm{e}^{-(x-\mu)^{\mathrm{T}}\Sigma^{-1}(x-\mu)/2}, \tag{9.1}$$

where the normalizing constant is Aitken's integral given in equation (4.75). The multivariate normal distribution is a good model for a wide range of random phenomena.

9.2.3 Derived Distributions and Cochran's Theorem

If X is a random variable with distribution $\mathrm{N}_d(\mu, \Sigma)$, A is a $q \times d$ matrix with rank q (which implies $q \leq d$), and $Y = AX$, then the straightforward change-of-variables technique yields the distribution of Y as $\mathrm{N}_d(A\mu,\, A\Sigma A^{\mathrm{T}})$.

Useful transformations of the random variable X with distribution $\mathrm{N}_d(\mu, \Sigma)$ are $Y_1 = \Sigma^{-1/2}X$ and $Y_2 = \Sigma_C^{-1}X$, where Σ_C is a Cholesky factor of Σ. In either case, the variance-covariance matrix of the transformed variate Y_1 or Y_2 is I_d.

Quadratic forms involving a Y that is distributed as $\mathrm{N}_d(\mu, I_d)$ have useful properties. For statistical inference it is important to know the distribution of these quadratic forms. The simplest quadratic form involves the identity matrix: $S_d = Y^{\mathrm{T}}Y$.

We can derive the PDF of S_d by beginning with $d = 1$ and using induction. If $d = 1$, for $t > 0$, we have

$$\Pr(S_1 \leq t) = \Pr(Y \leq \sqrt{t}) - \Pr(Y \leq -\sqrt{t}),$$

where $Y \sim \mathrm{N}_1(\mu, 1)$, and so the PDF of S_1 is

$$p_{S_1}(t) = \frac{1}{2\sqrt{2\pi t}}\left(\mathrm{e}^{-(\sqrt{t}-\mu)^2/2} + \mathrm{e}^{-(-\sqrt{t}-\mu)^2/2}\right)$$

$$
\begin{aligned}
&= \frac{\mathrm{e}^{-\mu^2/2}\mathrm{e}^{-t/2}}{2\sqrt{2\pi t}}\left(\mathrm{e}^{\mu\sqrt{t}} + \mathrm{e}^{-\mu\sqrt{t}}\right) \\
&= \frac{\mathrm{e}^{-\mu^2/2}\mathrm{e}^{-t/2}}{2\sqrt{2\pi t}}\left(\sum_{j=0}^{\infty}\frac{(\mu\sqrt{t})^j}{j!} + \sum_{j=0}^{\infty}\frac{(-\mu\sqrt{t})^j}{j!}\right) \\
&= \frac{\mathrm{e}^{-\mu^2/2}\mathrm{e}^{-t/2}}{\sqrt{2t}}\sum_{j=0}^{\infty}\frac{(\mu^2 t)^j}{\sqrt{\pi}(2j)!} \\
&= \frac{\mathrm{e}^{-\mu^2/2}\mathrm{e}^{-t/2}}{\sqrt{2t}}\sum_{j=0}^{\infty}\frac{(\mu^2 t)^j}{j!\Gamma(j+1/2)2^{2j}},
\end{aligned}
$$

in which we use the fact that

$$\Gamma(j+1/2) = \frac{\sqrt{\pi}(2j)!}{j!2^{2j}}$$

(see page 595). This can now be written as

$$p_{S_1}(t) = \mathrm{e}^{-\mu^2/2}\sum_{j=0}^{\infty}\frac{(\mu^2)^j}{j!2^j}\frac{1}{\Gamma(j+1/2)2^{j+1/2}}t^{j-1/2}\mathrm{e}^{-t/2}, \tag{9.2}$$

in which we recognize the PDF of the central chi-squared distribution with $2j+1$ degrees of freedom,

$$p_{\chi^2_{2j+1}}(t) = \frac{1}{\Gamma(j+1/2)2^{j+1/2}}t^{j-1/2}\mathrm{e}^{-t/2}. \tag{9.3}$$

A similar manipulation for $d = 2$ (that is, for $Y \sim \mathrm{N}_2(\mu, 1)$, and maybe $d = 3$, or as far as you need to go) leads us to a general form for the PDF of the $\chi^2_d(\delta)$ random variable S_d:

$$p_{S_d}(t) = \mathrm{e}^{-\mu^2/2}\sum_{j=0}^{\infty}\frac{(\mu^2/2)^j}{j!}p_{\chi^2_{2j+1}}(t). \tag{9.4}$$

We can show that equation (9.4) holds for any d by induction. The distribution of S_d is called the noncentral chi-squared distribution with d degrees of freedom and noncentrality parameter $\delta = \mu^{\mathrm{T}}\mu$. We denote this distribution as $\chi^2_d(\delta)$.

The induction method above involves a special case of a more general fact: if X_i for $i = 1, \ldots, k$ are independently distributed as $\chi^2_{n_i}(\delta_i)$, then $\sum_i X_i$ is distributed as $\chi^2_n(\delta)$, where $n = \sum_i n_i$ and $\delta = \sum_i \delta_i$. (Compare this with the result for Wishart distributions in Exercise 4.12b on page 225.)

In applications of linear models, a quadratic form involving Y is often partitioned into a sum of quadratic forms. Assume that Y is distributed as $\mathrm{N}_d(\mu, I_d)$, and for $i = 1, \ldots k$, let A_i be a $d \times d$ symmetric matrix with rank

r_i such that $\sum_i A_i = I_d$. This yields a partition of the total sum of squares $Y^{\mathrm{T}}Y$ into k components:

$$Y^{\mathrm{T}}Y = Y^{\mathrm{T}}A_1Y + \cdots + Y^{\mathrm{T}}A_kY. \tag{9.5}$$

One of the most important results in the analysis of linear models states that the $Y^{\mathrm{T}}A_iY$ have independent noncentral chi-squared distributions $\chi^2_{r_i}(\delta_i)$ with $\delta_i = \mu^{\mathrm{T}}A_i\mu$ if and only if $\sum_i r_i = d$.

This is called Cochran's theorem. Beginning on page 355, we discussed a form of Cochran's theorem that applies to properties of idempotent matrices. Those results immediately imply the conclusion above.

9.3 Linear Models

Some of the most important applications of statistics involve the study of the relationship of one variable, often called a "response variable", to other variables. The response variable is usually modeled as a random variable, which we indicate by using a capital letter. A general model for the relationship of a variable, Y, to other variables, x (a vector), is

$$Y \approx f(x). \tag{9.6}$$

In this asymmetric model and others like it, we call Y the *dependent variable* and the elements of x the *independent variables*.

It is often reasonable to formulate the model with a *systematic component* expressing the relationship and an *additive random component* or "additive error". We write

$$Y = f(x) + E, \tag{9.7}$$

where E is a random variable with an expected value of 0; that is,

$$\mathrm{E}(E) = 0.$$

(Although this is by far the most common type of model used by data analysts, there are other ways of building a model that incorporates systematic and random components.) The zero expectation of the random error yields the relationship

$$\mathrm{E}(Y) = f(x),$$

although this expression is not equivalent to the additive error model above because the random component could just as well be multiplicative (with an expected value of 1) and the same value of $\mathrm{E}(Y)$ would result.

Because the functional form f of the relationship between Y and x may contain a *parameter*, we may write the model as

$$Y = f(x;\theta) + E. \tag{9.8}$$

A specific form of this model is

$$Y = \beta^{\mathrm{T}} x + E, \tag{9.9}$$

which expresses the systematic component as a linear combination of the xs using the vector parameter β.

A model is more than an equation; there may be associated statements about the distribution of the random variable or about the nature of f or x. We may assume β (or θ) is a fixed but unknown constant, or we may assume it is a realization of a random variable. Whatever additional assumptions we may make, there are some standard assumptions that go with the model. We assume that Y and x are *observable* and θ and E are *unobservable*.

Models such as these that express an asymmetric relationship between some variables ("dependent variables") and other variables ("independent variables") are called regression models. A model such as equation (9.9) is called a linear regression model. There are many useful variations of the model (9.6) that express other kinds of relationships between the response variable and the other variables.

Notation

In data analysis with regression models, we have a set of observations $\{y_i, x_i\}$ where x_i is an m-vector. One of the primary tasks is to determine a reasonable value of the parameter. That is, in the linear regression model, for example, we think of β as an unknown variable (rather than as a fixed constant or a realization of a random variable), and we want to find a value of it such that the model fits the observations well,

$$y_i = \beta^{\mathrm{T}} x_i + \epsilon_i, \tag{9.10}$$

where β and x_i are m-vectors. (In the expression (9.9), "E" is an uppercase epsilon. We attempt to use notation consistently; "E" represents a random variable, and "ϵ" represents a realization, though an unobservable one, of the random variable. We will not always follow this convention, however; sometimes it is convenient to use the language more loosely and to speak of ϵ_i as a random variable.) The meaning of the phrase "the model fits the observations well" may vary depending on other aspects of the model, in particular, on any assumptions about the distribution of the random component E. If we make assumptions about the distribution, we have a basis for statistical estimation of β; otherwise, we can define some purely mathematical criterion for "fitting well" and proceed to determine a value of β that optimizes that criterion.

For any choice of β, say b, we have $y_i = b^{\mathrm{T}} x_i + r_i$. The r_is are determined by the observations. An approach that does not depend on any assumptions about the distribution but can nevertheless yield optimal estimators under many distributions is to choose the estimator so as to minimize some measure of the set of r_is.

Given the observations $\{y_i, x_i\}$, we can represent the regression model and the data as

$$y = X\beta + \epsilon, \tag{9.11}$$

where X is the $n \times m$ matrix whose rows are the x_is and ϵ is the vector of deviations ("errors") of the observations from the functional model. Throughout the rest of this section, *we will assume that the number of rows of X (that is, the number of observations n) is greater than the number of columns of X (that is, the number of variables m).*

We will occasionally refer to submatrices of the basic data matrix X using notation developed in Chap. 3. For example, $X_{(i_1,\ldots,i_k)(j_1,\ldots,j_l)}$ refers to the $k \times l$ matrix formed by retaining only the $i_1, \ldots, i_k$ rows and the $j_1, \ldots, j_l$ columns of X, and $X_{-(i_1,\ldots,i_k)(j_1,\ldots,j_l)}$ refers to the matrix formed by deleting the $i_1, \ldots, i_k$ rows and the $j_1, \ldots, j_l$ columns of X. We also use the notation x_{i*} to refer to the i^{th} row of X (the row is a vector, a column vector), and x_{*j} to refer to the j^{th} column of X. See page 599 for a summary of this notation.

9.3.1 Fitting the Model

In a model for a given dataset as in equation (9.11), although the errors are no longer random variables (they are realizations of random variables), they are not observable. To fit the model, we replace the unknowns with variables: β with b and ϵ with r. This yields

$$y = Xb + r. \tag{9.12}$$

We then proceed by applying some criterion for fitting.

The criteria generally focus on the "residuals" $r = y - Xb$. Two general approaches to fitting are:

- Define a likelihood function of r based on an assumed distribution of E, and determine a value of b that maximizes that likelihood.
- Decide on an appropriate norm on r, and determine a value of b that minimizes that norm.

There are other possible approaches, and there are variations on these two approaches. For the first approach, it must be emphasized that r is not a realization of the random variable E. Our emphasis will be on the second approach, that is, on methods that minimize a norm on r.

9.3.1.1 Statistical Estimation

The statistical problem is to *estimate* β. (Notice the distinction between the phrases "to *estimate* β" and "to determine a value of β that minimizes ...". The mechanical aspects of the two problems may be the same, of course.) The statistician uses the model and the given observations to explore relationships between the response and the regressors. Considering ϵ to be a realization of a random variable E (a vector) and assumptions about a distribution of the random variable ϵ allow us to make statistical inferences about a "true" β.

9.3.1.2 Ordinary Least Squares

The r vector contains the distances of the observations on y from the values of the variable y defined by the hyperplane $b^{\mathrm{T}}x$, measured *in the direction of the y axis*. The objective is to determine a value of b that minimizes some norm of r. The use of the L_2 norm is called "least squares". The estimate is the b that minimizes the dot product

$$(y - Xb)^{\mathrm{T}}(y - Xb) = \sum_{i=1}^{n}(y_i - x_{i*}^{\mathrm{T}}b)^2. \tag{9.13}$$

As we saw in Sect. 6.6 (where we used slightly different notation), using elementary calculus to determine the minimum of equation (9.13) yields the "normal equations"

$$X^{\mathrm{T}}X\widehat{\beta} = X^{\mathrm{T}}y. \tag{9.14}$$

9.3.1.3 Weighted Least Squares

The elements of the residual vector may be weighted differently. This is appropriate if, for instance, the variance of the residual depends on the value of x; that is, in the notation of equation (9.7), $\mathrm{V}(E) = g(x)$, where g is some function. If the function is known, we can address the problem almost identically as in the use of ordinary least squares, as we saw on page 295. Weighted least squares may also be appropriate if the observations in the sample are not independent. In this case also, if we know the variance-covariance structure, after a simple transformation, we can use ordinary least squares. If the function g or the variance-covariance structure must be estimated, the fitting problem is still straightforward, but formidable complications are introduced into other aspects of statistical inference. We discuss weighted least squares further in Sect. 9.3.6.

9.3.1.4 Variations on the Criteria for Fitting

Rather than minimizing a norm of r, there are many other approaches we could use to fit the model to the data. Of course, just the choice of the norm yields different approaches. Some of these approaches may depend on distributional assumptions, which we will not consider here. The point that we want to emphasize here, with little additional comment, is that the standard approach to regression modeling is not the only one. We mentioned some of these other approaches and the computational methods of dealing with them in Sect. 6.7. Alternative criteria for fitting regression models are sometimes considered in the many textbooks and monographs on data analysis using a linear regression model. This is because the fits may be more "robust" or more resistant to the effects of various statistical distributions.

9.3.1.5 Regularized Fits

Some variations on the basic approach of minimizing residuals involve a kind of regularization that may take the form of an additive penalty on the objective function. Regularization often results in a shrinkage of the estimator toward 0. One of the most common types of shrinkage estimator is the ridge regression estimator, which for the model $y = X\beta + \epsilon$ is the solution of the modified normal equations $(X^{\mathrm{T}}X + \lambda I)\beta = X^{\mathrm{T}}y$. We discuss this further in Sect. 9.5.4.

9.3.1.6 Orthogonal Distances

Another approach is to define an optimal value of β as one that minimizes a norm of the distances of the observed values of y from the vector $X\beta$. This is sometimes called "orthogonal distance regression". The use of the L_2 norm on this vector is sometimes called "total least squares". This is a reasonable approach when it is assumed that the observations in X are realizations of some random variable; that is, an "errors-in-variables" model is appropriate. The model in equation (9.11) is modified to consist of two error terms: one for the errors in the variables and one for the error in the equation. The methods discussed in Sect. 6.7.3 can be used to fit a model using a criterion of minimum norm of orthogonal residuals. As we mentioned there, weighting of the orthogonal residuals can be easily accomplished in the usual way of handling weights on the different observations.

The weight matrix often is formed as an inverse of a variance-covariance matrix Σ; hence, the modification is to premultiply the matrix $[X|y]$ in equation (6.56) by the Cholesky factor Σ_{C}^{-1}. In the case of errors-in-variables, however, there may be another variance-covariance structure to account for. If the variance-covariance matrix of the columns of X (that is, the independent variables) together with y is T, then we handle the weighting for variances and covariances of the columns of X in the same way, except of course we postmultiply the matrix $[X|y]$ in equation (6.56) by T_{C}^{-1}. This matrix is $(m+1) \times (m+1)$; however, it may be appropriate to assume any error in y is already accounted for, and so the last row and column of T may be 0 except for the $(m+1, m+1)$ element, which would be 1. The appropriate model depends on the nature of the data, of course.

9.3.1.7 Collinearity

A major problem in regression analysis is collinearity (or "multicollinearity"), by which we mean a "near singularity" of the X matrix. This can be made more precise in terms of a condition number, as discussed in Sect. 6.1. Ill-conditioning may not only present computational problems, but also may result in an estimate with a very large variance.

9.3.2 Linear Models and Least Squares

The most common estimator of β is one that minimizes the L_2 norm of the vertical distances in equation (9.11); that is, the one that forms a least squares fit. This criterion leads to the normal equations (9.14), whose solution is

$$\widehat{\beta} = (X^{\mathrm{T}}X)^{-}X^{\mathrm{T}}y. \tag{9.15}$$

(As we have pointed out many times, we often write formulas that are not to be used for computing a result; this is the case here.) If X is of full rank, the generalized inverse in equation (9.15) is, of course, the inverse, and $\widehat{\beta}$ is the unique least squares estimator. If X is not of full rank, we generally use the Moore-Penrose inverse, $(X^{\mathrm{T}}X)^{+}$, in equation (9.15).

As we saw in equations (6.43) and (6.44), we also have

$$\widehat{\beta} = X^{+}y. \tag{9.16}$$

On page 293, we derived this least squares solution by use of the QR decomposition of X. In Exercises 6.5a and 6.5b we mentioned two other ways to derive this important expression.

Equation (9.16) indicates the appropriate way to compute $\widehat{\beta}$. As we have seen many times before, however, we often use an expression without computing the individual terms. Instead of computing X^{+} in equation (9.16) explicitly, we use either Householder or Givens transformations to obtain the orthogonal decomposition

$$X = QR,$$

or

$$X = QRU^{\mathrm{T}}$$

if X is not of full rank. As we have seen, the QR decomposition of X can be performed row-wise using Givens transformations. This is especially useful if the data are available only one observation at a time. The equation used for computing $\widehat{\beta}$ is

$$R\widehat{\beta} = Q^{\mathrm{T}}y, \tag{9.17}$$

which can be solved by back substitution in the triangular matrix R.

Because

$$X^{\mathrm{T}}X = R^{\mathrm{T}}R,$$

the quantities in $X^{\mathrm{T}}X$ or its inverse, which are useful for making inferences using the regression model, can be obtained from the QR decomposition.

If X is not of full rank, the expression (9.16) not only is a least squares solution but the one with minimum length (minimum Euclidean norm), as we saw in equations (6.44) and (6.45).

The vector $\widehat{y} = X\widehat{\beta}$ is the projection of the n-vector y onto a space of dimension equal to the (column) rank of X, which we denote by r_X. The vector

of the model, $\mathrm{E}(Y) = X\beta$, is also in the r_X-dimensional space $\mathrm{span}(X)$. The projection matrix $I - X(X^{\mathrm{T}}X)^{+}X^{\mathrm{T}}$ projects y onto an $(n - r_X)$-dimensional residual space that is orthogonal to $\mathrm{span}(X)$. Figure 9.1 represents these subspaces and the vectors in them.

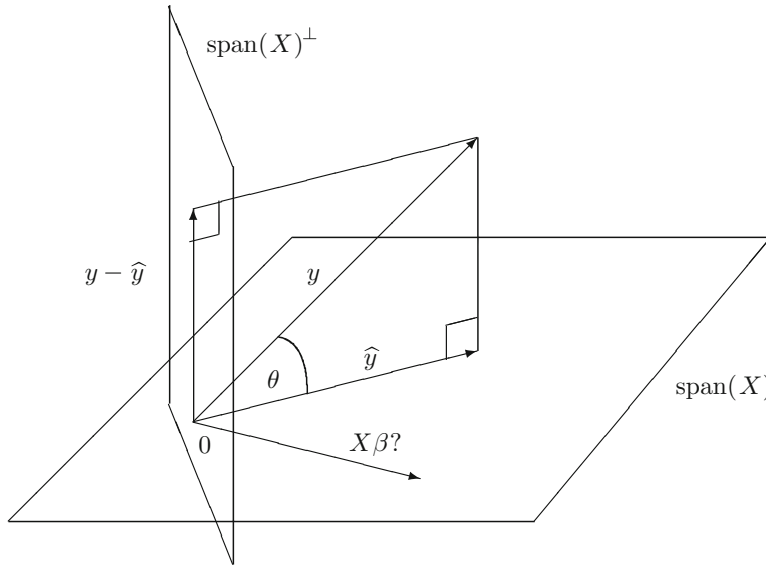


Figure 9.1. The linear least squares fit of y with X

Recall from page 291 that orthogonality of the residuals to $\mathrm{span}(X)$ is not only a property of a least squares solution, it actually characterizes a least squares solution; that is, if $\widehat{b}$ is such that $X^{\mathrm{T}}(y - X\widehat{b}) = 0$, then $\widehat{b}$ is a least squares solution.

In the $(r_X + 1)$-order vector space of the variables, the hyperplane defined by $\widehat{\beta}^{\mathrm{T}}x$ is the estimated model (assuming $\widehat{\beta} \neq 0$; otherwise, the space is of order r_X).

9.3.2.1 Degrees of Freedom

In general, the vector y can range freely over an n-dimensional space. We say the degrees of freedom of y, or the *total degrees of freedom*, is n. If we fix the mean of y, then the *adjusted total degrees of freedom* is $n - 1$.

The model $X\beta$ can range over a space with dimension equal to the (column) rank of X; that is, r_X. We say that the *model degrees of freedom* is r_X. Note that the space of $X\widehat{\beta}$ is the same as the space of $X\beta$.

Finally, the space orthogonal to $X\widehat{\beta}$ (that is, the space of the residuals $y - X\widehat{\beta}$) has dimension $n - r_X$. We say that the *residual (or error) degrees*

of freedom is $n - r_X$. (Note that the error vector ϵ can range over an n-dimensional space, but because $\widehat{\beta}$ is a least squares fit, $y - X\widehat{\beta}$ can only range over an $(n - r_X)$-dimensional space.)

9.3.2.2 The Hat Matrix and Leverage

The projection matrix $H = X(X^{\mathrm{T}}X)^{+}X^{\mathrm{T}}$ is sometimes called the "hat matrix" because

$$\begin{aligned}\widehat{y} &= X\widehat{\beta} \\ &= X(X^{\mathrm{T}}X)^{+}X^{\mathrm{T}}y \\ &= Hy,\end{aligned} \tag{9.18}$$

that is, it projects y onto $\widehat{y}$ in the span of X. Notice that the hat matrix can be computed without knowledge of the observations in y.

The elements of H are useful in assessing the effect of the particular pattern of the regressors on the predicted values of the response. The extent to which a given point in the row space of X affects the regression fit is called its "leverage". The leverage of the i^{th} observation is

$$h_{ii} = x_{i*}^{\mathrm{T}}(X^{\mathrm{T}}X)^{+}x_{i*}. \tag{9.19}$$

This is just the partial derivative of $\hat{y}_i$ with respect to y_i (Exercise 9.2). A relatively large value of h_{ii} compared with the other diagonal elements of the hat matrix means that the i^{th} observed response, y_i, has a correspondingly relatively large effect on the regression fit.

9.3.3 Statistical Inference

Fitting a model by least squares or by minimizing some other norm of the residuals in the data might be a sensible thing to do without any concern for a probability distribution. "Least squares" per se is not a statistical criterion. Certain statistical criteria, such as maximum likelihood or minimum variance estimation among a certain class of unbiased estimators, however, lead to an estimator that is the solution to a least squares problem for specific probability distributions.

For statistical inference about the parameters of the model $y = X\beta + \epsilon$ in equation (9.11), we must add something to the model. As in statistical inference generally, we must identify the random variables and make some statements (assumptions) about their distribution. The simplest assumptions are that ϵ is a random variable and $\mathrm{E}(\epsilon) = 0$. Whether or not the matrix X is random, our interest is in making inference conditional on the observed values of X.

9.3.3.1 Estimability

One of the most important questions for statistical inference involves estimating or testing some linear combination of the elements of the parameter β; for example, we may wish to estimate $\beta_1 - \beta_2$ or to test the hypothesis that $\beta_1 - \beta_2 = c_1$ for some constant c_1. In general, we will consider the linear combination $l^{\mathrm{T}}\beta$. Whether or not it makes sense to estimate such a linear combination depends on whether there is a function of the observable random variable Y such that $g(\mathrm{E}(Y)) = l^{\mathrm{T}}\beta$.

We generally restrict our attention to linear functions of $\mathrm{E}(Y)$ and formally define a linear combination $l^{\mathrm{T}}\beta$ to be *linearly estimable* if there exists a vector t such that

$$t^{\mathrm{T}}\mathrm{E}(Y) = l^{\mathrm{T}}\beta \tag{9.20}$$

for any β.

It is clear that if X is of full column rank, $l^{\mathrm{T}}\beta$ is linearly estimable for any l or, more generally, $l^{\mathrm{T}}\beta$ is linearly estimable for any $l \in \mathrm{span}(X^{\mathrm{T}})$. (The t vector is just the normalized coefficients expressing l in terms of the columns of X.)

Estimability depends only on the simplest distributional assumption about the model; that is, that $\mathrm{E}(\epsilon) = 0$. Under this assumption, we see that the estimator $\widehat{\beta}$ based on the least squares fit of β is unbiased for the linearly estimable function $l^{\mathrm{T}}\beta$. Because $l \in \mathrm{span}(X^{\mathrm{T}}) = \mathrm{span}(X^{\mathrm{T}}X)$, we can write $l = X^{\mathrm{T}}X\tilde{t}$. Now, we have

$$\begin{aligned}
\mathrm{E}(l^{\mathrm{T}}\widehat{\beta}) &= \mathrm{E}(l^{\mathrm{T}}(X^{\mathrm{T}}X)^{+}X^{\mathrm{T}}y) \\
&= \tilde{t}^{\mathrm{T}}X^{\mathrm{T}}X(X^{\mathrm{T}}X)^{+}X^{\mathrm{T}}X\beta \\
&= \tilde{t}^{\mathrm{T}}X^{\mathrm{T}}X\beta \\
&= l^{\mathrm{T}}\beta.
\end{aligned} \tag{9.21}$$

Although we have been taking $\widehat{\beta}$ to be $(X^{\mathrm{T}}X)^{+}X^{\mathrm{T}}y$, the equations above follow for other least squares fits, $b = (X^{\mathrm{T}}X)^{-}X^{\mathrm{T}}y$, for any generalized inverse. In fact, the estimator of $l^{\mathrm{T}}\beta$ is invariant to the choice of the generalized inverse. This is because if $b = (X^{\mathrm{T}}X)^{-}X^{\mathrm{T}}y$, we have $X^{\mathrm{T}}Xb = X^{\mathrm{T}}y$, and so

$$l^{\mathrm{T}}\widehat{\beta} - l^{\mathrm{T}}b = \tilde{t}^{\mathrm{T}}X^{\mathrm{T}}X(\widehat{\beta} - b) = \tilde{t}^{\mathrm{T}}(X^{\mathrm{T}}y - X^{\mathrm{T}}y) = 0. \tag{9.22}$$

In the context of the linear model, we call an estimator of β a *linear estimator* if it can be expressed as Ay for some matrix A, and we call an estimator of $l^{\mathrm{T}}\beta$ a linear estimator if it can be expressed as $a^{\mathrm{T}}y$ for some vector a. It is clear that the least squares estimators $\widehat{\beta}$ and $l^{\mathrm{T}}\widehat{\beta}$ are linear estimators.

Other properties of the estimators depend on additional assumptions about the distribution of ϵ, and we will consider some of them below.

When X is not of full rank, we often are interested in an orthogonal basis for $\mathrm{span}(X^{\mathrm{T}})$. If X includes a column of 1s, the elements of any vector in

the basis must sum to 0. Such vectors are called *contrasts*. The second and subsequent rows of the Helmert matrix (see Sect. 8.8.1 on page 381) are contrasts that are often of interest because of their regular patterns and their interpretability in applications involving the analysis of levels of factors in experiments.

9.3.3.2 Testability

We define a linear hypothesis $l^{\mathrm{T}}\beta = c_1$ as *testable* if $l^{\mathrm{T}}\beta$ is estimable. We generally restrict our attention to testable hypotheses.

It is often of interest to test multiple hypotheses concerning linear combinations of the elements of β. For the model (9.11), the *general linear hypothesis* is

$$\mathrm{H}_0 : \; L^{\mathrm{T}}\beta = c,$$

where L is $m \times q$, of rank q, and such that $\mathrm{span}(L) \subseteq \mathrm{span}(X)$.

The test for a hypothesis depends on the distributions of the random variables in the model. If we assume that the elements of ϵ are i.i.d. normal with a mean of 0, then the general linear hypothesis is tested using an F statistic whose numerator is the difference in the residual sum of squares from fitting the model with the restriction $L^{\mathrm{T}}\beta = c$ and the residual sum of squares from fitting the unrestricted model. This reduced sum of squares is

$$(L^{\mathrm{T}}\widehat{\beta} - c)^{\mathrm{T}} \, (L^{\mathrm{T}}(X^{\mathrm{T}}X)^{*}L)^{-1} \, (L^{\mathrm{T}}\widehat{\beta} - c), \tag{9.23}$$

where $(X^{\mathrm{T}}X)^{*}$ is any g_2 inverse of $X^{\mathrm{T}}X$. This test is a likelihood ratio test. (See a text on linear models, such as Searle 1971, for more discussion on this testing problem.)

To compute the quantity in expression (9.23), first observe

$$L^{\mathrm{T}}(X^{\mathrm{T}}X)^{*}L = (X(X^{\mathrm{T}}X)^{*}L)^{\mathrm{T}} \, (X(X^{\mathrm{T}}X)^{*}L). \tag{9.24}$$

Now, if $X(X^{\mathrm{T}}X)^{*}L$, which has rank q, is decomposed as

$$X(X^{\mathrm{T}}X)^{*}L = P \begin{bmatrix} T \\ 0 \end{bmatrix},$$

where P is an $m \times m$ orthogonal matrix and T is a $q \times q$ upper triangular matrix, we can write the reduced sum of squares (9.23) as

$$(L^{\mathrm{T}}\widehat{\beta} - c)^{\mathrm{T}} \, (T^{\mathrm{T}}T)^{-1} \, (L^{\mathrm{T}}\widehat{\beta} - c)$$

or

$$\left(T^{-\mathrm{T}}(L^{\mathrm{T}}\widehat{\beta} - c)\right)^{\mathrm{T}} \; \left(T^{-\mathrm{T}}(L^{\mathrm{T}}\widehat{\beta} - c)\right)$$

or

$$v^{\mathrm{T}}v. \tag{9.25}$$

To compute v, we solve

$$T^{\mathrm{T}}v = L^{\mathrm{T}}\widehat{\beta} - c \tag{9.26}$$

for v, and the reduced sum of squares is then formed as $v^{\mathrm{T}}v$.

9.3.3.3 The Gauss-Markov Theorem

The Gauss-Markov theorem provides a restricted optimality property for estimators of estimable functions of β under the condition that $\mathrm{E}(\epsilon) = 0$ and $\mathrm{V}(\epsilon) = \sigma^2 I$; that is, in addition to the assumption of zero expectation, which we have used above, we also assume that the elements of ϵ have constant variance and that their covariances are zero. (We are not assuming independence or normality, as we did in order to develop tests of hypotheses.)

Given $y = X\beta + \epsilon$ and $\mathrm{E}(\epsilon) = 0$ and $\mathrm{V}(\epsilon) = \sigma^2 I$, the Gauss-Markov theorem states that $l^{\mathrm{T}}\widehat{\beta}$ is the unique *best linear unbiased estimator* (BLUE) of the estimable function $l^{\mathrm{T}}\beta$.

"Linear" estimator in this context means a linear combination of y; that is, an estimator in the form $a^{\mathrm{T}}y$. It is clear that $l^{\mathrm{T}}\widehat{\beta}$ is linear, and we have already seen that it is unbiased for $l^{\mathrm{T}}\beta$. "Best" in this context means that its variance is no greater than any other estimator that fits the requirements. Hence, to prove the theorem, first let $a^{\mathrm{T}}y$ be any unbiased estimator of $l^{\mathrm{T}}\beta$, and write $l = X^{\mathrm{T}}X\tilde{t}$ as above. Because $a^{\mathrm{T}}y$ is unbiased for any β, as we saw above, it must be the case that $a^{\mathrm{T}}X = l^{\mathrm{T}}$. Recalling that $X^{\mathrm{T}}X\widehat{\beta} = X^{\mathrm{T}}y$, we have

$$\begin{aligned}
\mathrm{V}(a^{\mathrm{T}}y) &= \mathrm{V}(a^{\mathrm{T}}y - l^{\mathrm{T}}\widehat{\beta} + l^{\mathrm{T}}\widehat{\beta}) \\
&= \mathrm{V}(a^{\mathrm{T}}y - \tilde{t}^{\mathrm{T}}X^{\mathrm{T}}y + l^{\mathrm{T}}\widehat{\beta}) \\
&= \mathrm{V}(a^{\mathrm{T}}y - \tilde{t}^{\mathrm{T}}X^{\mathrm{T}}y) + \mathrm{V}(l^{\mathrm{T}}\widehat{\beta}) + 2\mathrm{Cov}(a^{\mathrm{T}}y - \tilde{t}^{\mathrm{T}}X^{\mathrm{T}}y,\ \tilde{t}^{\mathrm{T}}X^{\mathrm{T}}y).
\end{aligned}$$

Now, under the assumptions on the variance-covariance matrix of ϵ, which is also the (conditional, given X) variance-covariance matrix of y, we have

$$\begin{aligned}
\mathrm{Cov}(a^{\mathrm{T}}y - \tilde{t}^{\mathrm{T}}X^{\mathrm{T}}y,\ l^{\mathrm{T}}\widehat{\beta}) &= (a^{\mathrm{T}} - \tilde{t}^{\mathrm{T}}X^{\mathrm{T}})\sigma^2 I X\tilde{t} \\
&= (a^{\mathrm{T}}X - \tilde{t}^{\mathrm{T}}X^{\mathrm{T}}X)\sigma^2 I\tilde{t} \\
&= (l^{\mathrm{T}} - l^{\mathrm{T}})\sigma^2 I\tilde{t} \\
&= 0;
\end{aligned}$$

that is,

$$\mathrm{V}(a^{\mathrm{T}}y) = \mathrm{V}(a^{\mathrm{T}}y - \tilde{t}^{\mathrm{T}}X^{\mathrm{T}}y) + \mathrm{V}(l^{\mathrm{T}}\widehat{\beta}).$$

This implies that

$$\mathrm{V}(a^{\mathrm{T}}y) \geq \mathrm{V}(l^{\mathrm{T}}\widehat{\beta});$$

that is, $l^{\mathrm{T}}\widehat{\beta}$ has minimum variance among the linear unbiased estimators of $l^{\mathrm{T}}\beta$. To see that it is unique, we consider the case in which $\mathrm{V}(a^{\mathrm{T}}y) = \mathrm{V}(l^{\mathrm{T}}\widehat{\beta})$; that is, $\mathrm{V}(a^{\mathrm{T}}y - \tilde{t}^{\mathrm{T}}X^{\mathrm{T}}y) = 0$. For this variance to equal 0, it must be the case that $a^{\mathrm{T}} - \tilde{t}^{\mathrm{T}}X^{\mathrm{T}} = 0$ or $a^{\mathrm{T}}y = \tilde{t}^{\mathrm{T}}X^{\mathrm{T}}y = l^{\mathrm{T}}\widehat{\beta}$; that is, $l^{\mathrm{T}}\widehat{\beta}$ is the unique linear unbiased estimator that achieves the minimum variance.

If we assume further that $\epsilon \sim \mathrm{N}_n(0, \sigma^2 I)$, we can show that $l^{\mathrm{T}}\widehat{\beta}$ is the uniformly minimum variance unbiased estimator (UMVUE) for $l^{\mathrm{T}}\beta$. This is because $(X^{\mathrm{T}}y,\ (y - X\widehat{\beta})^{\mathrm{T}}(y - X\widehat{\beta}))$ is complete and sufficient for (β, σ^2). This line of reasoning also implies that $(y - X\widehat{\beta})^{\mathrm{T}}(y - X\widehat{\beta})/(n - r)$, where $r = \mathrm{rank}(X)$, is UMVUE for σ^2. We will not go through the details here. The interested reader is referred to a text on mathematical statistics, such as Shao (2003).

9.3.4 The Normal Equations and the Sweep Operator

The coefficient matrix in the normal equations, $X^{\mathrm{T}}X$, or the adjusted version $X_c^{\mathrm{T}}X_c$, where X_c is the centered matrix as in equation (8.64) on page 366, is often of interest for reasons other than just to compute the least squares estimators. The condition number of $X^{\mathrm{T}}X$ is the square of the condition number of X, however, and so any ill-conditioning is exacerbated by formation of the sums of squares and cross products matrix. The adjusted sums of squares and cross products matrix, $X_c^{\mathrm{T}}X_c$, tends to be better conditioned, so it is usually the one used in the normal equations, but of course the condition number of $X_c^{\mathrm{T}}X_c$ is the square of the condition number of X_c.

A useful matrix can be formed from the normal equations:

$$\begin{bmatrix} X^{\mathrm{T}}X & X^{\mathrm{T}}y \\ y^{\mathrm{T}}X & y^{\mathrm{T}}y \end{bmatrix}. \tag{9.27}$$

Applying m elementary operations on this matrix, we can get

$$\begin{bmatrix} (X^{\mathrm{T}}X)^{+} & X^{+}y \\ y^{\mathrm{T}}X^{+\mathrm{T}} & y^{\mathrm{T}}y - y^{\mathrm{T}}X(X^{\mathrm{T}}X)^{+}X^{\mathrm{T}}y \end{bmatrix}. \tag{9.28}$$

(If X is not of full rank, in order to get the Moore-Penrose inverse in this expression, the elementary operations must be applied in a fixed manner; otherwise, we get a different generalized inverse.)

The matrix in the upper left of the partition (9.28) is related to the estimated variance-covariance matrix of the particular solution of the normal equations, and it can be used to get an estimate of the variance-covariance matrix of estimates of any independent set of linearly estimable functions of

β. The vector in the upper right of the partition is the unique minimum-length solution to the normal equations, $\widehat{\beta}$. The scalar in the lower right partition, which is the Schur complement of the full inverse (see equations (3.190) and (3.214)), is the square of the residual norm. The squared residual norm provides an estimate of the variance of the errors in equation (9.11) after proper scaling.

The partitioning in expression (9.28) is the same that we encountered on page 363.

The elementary operations can be grouped into a larger operation, called the "sweep operation", which is performed for a given row. The sweep operation on row i, S_i, of the nonnegative definite matrix A to yield the matrix B, which we denote by

$$\mathrm{S}_i(A) = B,$$

is defined in Algorithm 9.1.

Algorithm 9.1 Sweep of the i^{th} row

1. If $a_{ii} = 0$, skip the following operations.
2. Set $b_{ii} = a_{ii}^{-1}$.
3. For $j \neq i$, set $b_{ij} = a_{ii}^{-1} a_{ij}$.
4. For $k \neq i$, set $b_{kj} = a_{kj} - a_{ki} a_{ii}^{-1} a_{ij}$. ∎

Skipping the operations if $a_{ii} = 0$ allows the sweep operator to handle non-full rank problems. The sweep operator is its own inverse:

$$\mathrm{S}_i(\mathrm{S}_i(A)) = A.$$

The sweep operator applied to the matrix (9.27) corresponds to adding or removing the i^{th} variable (column) of the X matrix to the regression equation.

9.3.5 Linear Least Squares Subject to Linear Equality Constraints

In the regression model (9.11), it may be known that β satisfies certain constraints, such as that all the elements be nonnegative. For constraints of the form $g(\beta) \in C$, where C is some m-dimensional space, we may estimate β by the *constrained least squares estimator*; that is, the vector $\widehat{\beta}_C$ that minimizes the dot product (9.13) among all b that satisfy $g(b) \in C$.

The nature of the constraints may or may not make drastic changes to the computational problem. (The constraints also change the statistical inference problem in various ways, but we do not address that here.) If the constraints are nonlinear, or if the constraints are inequality constraints (such as that all the elements be nonnegative), there is no general closed-form solution.

It is easy to handle linear equality constraints of the form

$$\begin{aligned} g(\beta) &= L\beta \\ &= c, \end{aligned}$$

where L is a $q \times m$ matrix of full rank. The solution is, analogous to equation (9.15),

$$\widehat{\beta}_C = (X^{\mathrm{T}}X)^+X^{\mathrm{T}}y + (X^{\mathrm{T}}X)^+L^{\mathrm{T}}(L(X^{\mathrm{T}}X)^+L^{\mathrm{T}})^+(c - L(X^{\mathrm{T}}X)^+X^{\mathrm{T}}y). \tag{9.29}$$

When X is of full rank, this result can be derived by using Lagrange multipliers and the derivative of the norm (9.13) (see Exercise 9.4 on page 452). When X is not of full rank, it is slightly more difficult to show this, but it is still true. (See a text on linear regression, such as Draper and Smith 1998).

The restricted least squares estimate, $\widehat{\beta}_C$, can be obtained (in the (1, 2) block) by performing $m + q$ sweep operations on the matrix,

$$\begin{bmatrix} X^{\mathrm{T}}X & X^{\mathrm{T}}y & L^{\mathrm{T}} \\ y^{\mathrm{T}}X & y^{\mathrm{T}}y & c^{\mathrm{T}} \\ L & c & 0 \end{bmatrix}, \tag{9.30}$$

analogous to matrix (9.27).

9.3.6 Weighted Least Squares

In fitting the regression model $y \approx X\beta$, it is often desirable to weight the observations differently, and so instead of minimizing equation (9.13), we minimize

$$\sum w_i(y_i - x_{i*}^{\mathrm{T}}b)^2,$$

where w_i represents a nonnegative weight to be applied to the i^{th} observation. One purpose of the weight may be to control the effect of a given observation on the overall fit. If a model of the form of equation (9.11),

$$y = X\beta + \epsilon,$$

is assumed, and ϵ is taken to be a random variable such that ϵ_i has variance σ_i^2, an appropriate value of w_i may be $1/\sigma_i^2$. (Statisticians almost always naturally assume that ϵ is a random variable. Although usually it is modeled this way, here we are allowing for more general interpretations and more general motives in fitting the model.)

The normal equations can be written as

$$\left(X^{\mathrm{T}}\mathrm{diag}((w_1, w_2, \ldots, w_n))X\right)\widehat{\beta} = X^{\mathrm{T}}\mathrm{diag}((w_1, w_2, \ldots, w_n))y.$$

More generally, we can consider W to be a weight matrix that is not necessarily diagonal. We have the same set of normal equations:

$$(X^{\mathrm{T}}WX)\widehat{\beta}_W = X^{\mathrm{T}}Wy. \tag{9.31}$$

When W is a diagonal matrix, the problem is called "weighted least squares". Use of a nondiagonal W is also called weighted least squares but is sometimes

called "generalized least squares". The weight matrix is symmetric and generally positive definite, or at least nonnegative definite. The weighted least squares estimator is

$$\widehat{\beta}_W = (X^{\mathrm{T}}WX)^{+}X^{\mathrm{T}}Wy.$$

As we have mentioned many times, an expression such as this is not necessarily a formula for computation. The matrix factorizations discussed above for the unweighted case can also be used for computing weighted least squares estimates.

In a model $y = X\beta + \epsilon$, where ϵ is taken to be a random variable with variance-covariance matrix Σ, the choice of W as Σ^{-1} yields estimators with certain desirable statistical properties. (Because this is a natural choice for many models, statisticians sometimes choose the weighting matrix without fully considering the reasons for the choice.) As we pointed out on page 295, weighted least squares can be handled by premultiplication of both y and X by the Cholesky factor of the weight matrix. In the case of an assumed variance-covariance matrix Σ, we transform each side by Σ_{C}^{-1}, where Σ_{C} is the Cholesky factor of Σ. The residuals whose squares are to be minimized are $\Sigma_{\mathrm{C}}^{-1}(y - Xb)$. Under the assumptions, the variance-covariance matrix of the residuals is I.

9.3.7 Updating Linear Regression Statistics

In Sect. 6.6.5 on page 295, we discussed the general problem of updating a least squares solution to an overdetermined system when either the number of equations (rows) or the number of variables (columns) is changed. In the linear regression problem these correspond to adding or deleting observations and adding or deleting terms in the linear model, respectively.

9.3.7.1 Adding More Variables

Suppose first that more variables are added, so the regression model is

$$y \approx \begin{bmatrix} X & X_+ \end{bmatrix} \theta,$$

where X_+ represents the observations on the additional variables. (We use θ to represent the parameter vector; because the model is different, it is not just β with some additional elements.)

If $X^{\mathrm{T}}X$ has been formed and the sweep operator is being used to perform the regression computations, it can be used easily to add or delete variables from the model, as we mentioned above. The Sherman-Morrison-Woodbury formulas (6.28) and (6.30) and the Hemes formula (6.31) (see page 288) can also be used to update the solution.

In regression analysis, one of the most important questions is the identification of independent variables from a set of potential explanatory variables that should be in the model. This aspect of the analysis involves adding and deleting variables. We discuss this further in Sect. 9.5.2.

9.3.7.2 Adding More Observations

If we have obtained more observations, the regression model is

$$\begin{bmatrix} y \\ y_+ \end{bmatrix} \approx \begin{bmatrix} X \\ X_+ \end{bmatrix} \beta,$$

where y_+ and X_+ represent the additional observations.

We first note some properties of the new $X^{\mathrm{T}}X$ matrix, although we will make direct use of the new X matrix, as usual. We see that

$$\begin{bmatrix} X \\ X_+ \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} X \\ X_+ \end{bmatrix} = X^{\mathrm{T}}X + X_+^{\mathrm{T}}X_+.$$

The relation of the inverse of $X^{\mathrm{T}}X + X_+^{\mathrm{T}}X_+$ to the inverse of $X^{\mathrm{T}}X$ can be seen in equation (3.177) on page 119, or in equation (3.185) for the vector corresponding to a single additional row.

If the QR decomposition of X is available, we simply augment it as in equation (6.47):

$$\begin{bmatrix} R & c_1 \\ 0 & c_2 \\ X_+ & y_+ \end{bmatrix} = \begin{bmatrix} Q^{\mathrm{T}} & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} X & y \\ X_+ & y_+ \end{bmatrix}.$$

We now apply orthogonal transformations to this to zero out the last rows and produce

$$\begin{bmatrix} R_* & c_{1*} \\ 0 & c_{2*} \end{bmatrix},$$

where R_* is an $m \times m$ upper triangular matrix and c_{1*} is an m-vector as before, but c_{2*} is an $(n - m + k)$-vector. We then have an equation of the form (9.17) and we use back substitution to solve it.

9.3.7.3 Adding More Observations Using Weights

Another way of approaching the problem of adding or deleting observations is by viewing the problem as weighted least squares. In this approach, we also have more general results for updating regression statistics. Following Escobar and Moser (1993), we can consider two weighted least squares problems: one with weight matrix W and one with weight matrix V. Suppose we have the solutions $\widehat{\beta}_W$ and $\widehat{\beta}_V$. Now let

$$\Delta = V - W,$$

and use the subscript $*$ on any matrix or vector to denote the subarray that corresponds only to the nonnull rows of Δ. The symbol Δ_*, for example, is the square subarray of Δ consisting of all of the nonzero rows and columns of Δ, and X_* is the subarray of X consisting of all the columns of X and only

the rows of X that correspond to Δ_*. From the normal equations (9.31) using W and V, and with the solutions $\widehat{\beta}_W$ and $\widehat{\beta}_V$ plugged in, we have

$$(X^{\mathrm{T}}WX)\widehat{\beta}_V + (X^{\mathrm{T}}\Delta X)\widehat{\beta}_V = X^{\mathrm{T}}Wy + X^{\mathrm{T}}\Delta y,$$

and so

$$\widehat{\beta}_V - \widehat{\beta}_W = (X^{\mathrm{T}}WX)^+ X_*^{\mathrm{T}}\Delta_*(y - X\widehat{\beta}_V)_*.$$

This gives

$$(y - X\widehat{\beta}_V)_* = (I + X(X^{\mathrm{T}}WX)^+ X_*^{\mathrm{T}}\Delta_*)^+(y - X\widehat{\beta}_W)_*,$$

and finally

$$\widehat{\beta}_V = \widehat{\beta}_W + (X^{\mathrm{T}}WX)^+ X_*^{\mathrm{T}}\Delta_* \left(I + X_*(X^{\mathrm{T}}WX)^+ X_*^{\mathrm{T}}\Delta_*\right)^+ (y - X\widehat{\beta}_W)_*.$$

If Δ_* can be written as $\pm GG^{\mathrm{T}}$, using this equation and the equations (3.176) on page 119 (which also apply to pseudoinverses), we have

$$\widehat{\beta}_V = \widehat{\beta}_W \pm (X^{\mathrm{T}}WX)^+ X_*^{\mathrm{T}}G(I \pm G^{\mathrm{T}}X_*(X^{\mathrm{T}}WX)^+ X_*^{\mathrm{T}}G)^+ G^{\mathrm{T}}(y - X\widehat{\beta}_W)_*. \tag{9.32}$$

The sign of GG^{T} is positive when observations are added and negative when they are deleted.

Equation (9.32) is particularly simple in the case where W and V are identity matrices (of different sizes, of course). Suppose that we have obtained more observations in y_+ and X_+. (In the following, the reader must be careful to distinguish "+" as a subscript to represent more data and "+" as a superscript with its usual meaning of a Moore-Penrose inverse.) Suppose we already have the least squares solution for $y \approx X\beta$, say $\widehat{\beta}_W$. Now $\widehat{\beta}_W$ is the weighted least squares solution to the model with the additional data and with weight matrix

$$W = \begin{bmatrix} I & 0 \\ 0 & 0 \end{bmatrix}.$$

We now seek the solution to the same system with weight matrix V, which is a larger identity matrix. From equation (9.32), the solution is

$$\widehat{\beta} = \widehat{\beta}_W + (X^{\mathrm{T}}X)^+ X_+^{\mathrm{T}}(I + X_+(X^{\mathrm{T}}X)^+ X_+^{\mathrm{T}})^+(y - X\widehat{\beta}_W)_*. \tag{9.33}$$

9.3.8 Linear Smoothing

The interesting reasons for doing regression analysis are to understand relationships and to predict a value of the dependent value given a value of the

independent variable. As a side benefit, a model with a smooth equation $f(x)$ "smoothes" the observed responses; that is, the elements in $\hat{y} = \widehat{f(x)}$ exhibit less variation than the elements in y. Of course, the important fact for our purposes is that $\|y - \hat{y}\|$ is smaller than $\|y\|$ or $\|y - \bar{y}\|$.

The use of the hat matrix emphasizes the smoothing perspective as a projection of the original y:

$$\hat{y} = Hy.$$

The concept of a smoothing matrix was discussed in Sect. 8.6.2. From this perspective, using H, we project y onto a vector in span(H), and that vector has a smaller variation than y; that is, H has *smoothed* y. It does not matter what the specific values in the vector y are so long as they are associated with the same values of the independent variables.

We can extend this idea to a general $n \times n$ *smoothing matrix* H_λ:

$$\tilde{y} = H_\lambda y.$$

The smoothing matrix depends only on the kind and extent of smoothing to be performed and on the observed values of the independent variables. The extent of the smoothing may be indicated by the indexing parameter λ. Once the smoothing matrix is obtained, it does not matter how the independent variables are related to the model.

In Sect. 6.7.2, we discussed regularized solutions of overdetermined systems of equations, which in the present case is equivalent to solving

$$\min_b \left((y - Xb)^{\mathrm{T}}(y - Xb) + \lambda b^{\mathrm{T}} b \right).$$

The solution of this yields the smoothing matrix

$$S_\lambda = X(X^{\mathrm{T}}X + \lambda I)^{-1}X^{\mathrm{T}},$$

as we have seen on page 364. This has the effect of shrinking the $\widehat{y}$ toward 0. (In regression analysis, this is called "ridge regression".)

We discuss ridge regression and general shrinkage estimation in Sect. 9.5.4. Loader (2012) provides additional background and discusses more general issues in smoothing.

9.3.9 Multivariate Linear Models

A simple modification of the model (9.10), $y_i = \beta^{\mathrm{T}} x_i + \epsilon_i$, on page 404, extends the scalar responses to vector responses; that is, y_i is a vector, and of course, the vector of parameters β must be replaced by a matrix. Let d be the order of y_i. Similarly, ϵ_i is a d-vector.

This is a "multivariate" linear model, meaning among other things, that the error term has a multivariate distribution (it is not a set of i.i.d. scalars).

A major difference in the multivariate linear model arises from the structure of the vector ϵ_i. It may be appropriate to assume that the ϵ_is are independent from one observation to another, but it is not likely that the individual elements within an ϵ_i vector are independent from each other or even that they have zero correlations. A reasonable assumption to complete the model is that the vectors ϵ_is are independently and identically distributed with mean 0 and variance-covariance matrix Σ. It might be reasonable also to assume that they have a normal distribution.

In statistical applications in which univariate responses are modeled, instead of the model for a single observation, we are more likely to write the model for a set of observations on y and x in the form of equation (9.11), $y = X\beta + \epsilon$, in which y and ϵ are d-vectors, X is a matrix in which the rows correspond to the individual x_i. Extending this form to the multivariate model, we write

$$Y = XB + E, \tag{9.34}$$

where now Y is an $n \times d$ matrix, X is an $n \times m$ matrix as before, B is an $m \times d$ matrix and E is an $n \times d$ matrix. Under the assumptions on the distribution of the vector ϵ_i above, and including the assumption of normality, E in (9.34) has a matrix normal distribution (see expression (4.78) on page 221):

$$E \sim \mathrm{N}_{n,d}(0, I, \Sigma), \tag{9.35}$$

or in the form of (4.79),

$$\mathrm{vec}\left(E^{\mathrm{T}}\right) \sim \mathrm{N}_{dn}\left(0, \mathrm{diag}(\Sigma, \ldots, \Sigma)\right).$$

Note that the variance-covariance matrix in this distribution has Kronecker structure, since $\mathrm{diag}(\Sigma, \ldots, \Sigma) = I \otimes \Sigma$ (see also equation (3.102)).

9.3.9.1 Fitting the Model

Fitting multivariate linear models is done in the same way as fitting univariate linear models. The most common criterion for fitting is least squares, which as we have pointed out before is the same as a maximum likelihood criterion if the errors are identically and independently normally distributed (which follows from the identity matrix I in expression (9.35)). This is the same fitting problem that we considered in Sect. 9.3.1 in this chapter or, earlier in Sect. 6.6 on page 289.

In an approach similar to the development in Sect. 6.6, for a given choice of B, say $\widetilde{B}$, we have, corresponding to equation (6.33),

$$X\widetilde{B} = Y - R, \tag{9.36}$$

where R is an $n \times d$ matrix of residuals.

A least squares solution $\widehat{B}$ is one that minimizes the sum of squares of the residuals (or, equivalently, the square root of the sum of the squares, that is, $\|R\|_{\mathrm{F}}$). Hence, we have the optimization problem

$$\min_{\widetilde{B}} \left\| Y - X\widetilde{B} \right\|_{\mathrm{F}}. \tag{9.37}$$

As in Sect. 6.6, we rewrite the square of this norm, using equation (3.291) from page 168, as

$$\mathrm{tr}\left((Y - X\widetilde{B})^{\mathrm{T}}(Y - X\widetilde{B})\right). \tag{9.38}$$

This is similar to equation (6.35), which, as before, we differentiate and set equal to zero, getting the normal equations in the vector $\widehat{B}$,

$$X^{\mathrm{T}}X\widehat{B} = X^{\mathrm{T}}Y. \tag{9.39}$$

(Exercise.)

We note that the columns of the matrices in these equations are each the same as the univariate normal equations (6.36):

$$X^{\mathrm{T}}X[\widehat{B}_{*1}, \ldots, \widehat{B}_{*d}] = X^{\mathrm{T}}[Y_{*1}, \ldots, Y_{*d}].$$

9.3.9.2 Partitioning the Sum of Squares

On page 363, we discussed the partitioning of the sum of squares of an observed vector of data, $y^{\mathrm{T}}y$. We did this in the context of the Gramian of the partitioned matrix $[X\, y]$. In the multivariate case, first of all, instead of the sum of squares $y^{\mathrm{T}}y$, we have the matrix of sums of squares and cross products, $Y^{\mathrm{T}}Y$. We now consider the Gramian matrix $[X\, Y]^{\mathrm{T}}[X\, Y]$, and partition it as in expression (9.27),

$$\begin{bmatrix} X^{\mathrm{T}}X & X^{\mathrm{T}}Y \\ Y^{\mathrm{T}}X & Y^{\mathrm{T}}Y \end{bmatrix}. \tag{9.40}$$

From this we can get

$$\begin{bmatrix} (X^{\mathrm{T}}X)^{+} & X^{+}Y \\ Y^{\mathrm{T}}X^{+\mathrm{T}} & Y^{\mathrm{T}}Y - Y^{\mathrm{T}}X(X^{\mathrm{T}}X)^{+}X^{\mathrm{T}}Y \end{bmatrix}. \tag{9.41}$$

Note that the term in the lower right side in this partitioning is the Schur complement of $X^{\mathrm{T}}X$ in $[X\, Y]^{\mathrm{T}}\, [X\, Y]$ (see equation (3.191) on page 122). This matrix of residual sums of squares and cross products provides a maximum likelihood estimator of Σ:

$$\widehat{\Sigma} = \left(Y^{\mathrm{T}}Y - Y^{\mathrm{T}}X(X^{\mathrm{T}}X)^{+}X^{\mathrm{T}}Y\right)/n. \tag{9.42}$$

If the normalizing factor is $1/(n-d)$ instead of $1/n$, the estimator is unbiased.

This partitioning breaks the matrix of total sums of squares and cross products into a sum of a matrix of sums of squares and cross products due to the fitted relationship between Y and X and a matrix of residual sums of squares and cross products. The analysis of these two matrices of sums

of squares and cross products is one of the most fundamental and important techniques in multivariate statistics.

The matrix in the upper left of the partition (9.41) can be used to get an estimate of the variance-covariance matrix of estimates of any independent set of linearly estimable functions of B. The matrix in the upper right of the partition is the solution to the normal equations, $\widehat{B}$. The matrix in the lower right partition, which is the Schur complement of the full inverse (see equations (3.190) and (3.214)), is the matrix of sums of squares and cross products of the residuals. With proper scaling, it provides an estimate of the variance-covariance Σ of each row of E in equation (9.34).

9.3.9.3 Statistical Inference

Statistical inference for multivariate linear models is similar to what is described in Sect. 9.3.3 with some obvious changes and extensions. First order properties of distributions of the analogous statistics are almost the same. Second order properties (variances), however, are rather different. The solution of the normal equations $\widehat{B}$ has a matrix normal distribution with expectation B. The scaled matrix of sums of squares and cross products of the residuals, call it $\widehat{\Sigma}$, has a Wishart distribution with parameter Σ.

The basic null hypothesis of interest that the distribution of Y is not dependent on X is essentially the same as in the univariate case. The F-test in the corresponding univariate case, which is the ratio of two independent chi-squared random variables, has an analogue in a comparison of two matrices of sums of squares and cross products. In the multivariate case, the basis for statistical inference is $\widehat{\Sigma}$, and it can be used in various ways. The relevant fact is that $\widehat{\Sigma} \sim \mathrm{W}_d(\Sigma, n-d)$, that is, it has a Wishart distribution with variance-covariance matrix Σ and parameters d and $n-d$ (see Exercise 4.12 on page 224).

In hypothesis testing, depending on the null hypothesis, there are other matrices that have Wishart distributions. There are various scalar transformations of Wishart matrices whose distributions are known (or which have been approximated). One of the most common ones, and which even has some of the flavor of an F statistic, is Wilk's Λ,

$$\Lambda = \frac{\det\left(\widehat{\Sigma}\right)}{\det\left(\widehat{\Sigma}_0\right)}, \tag{9.43}$$

where $\widehat{\Sigma}_0$ is a scaled Wishart matrix yielding a maximum of the likelihood under a null hypothesis. Other related test statistics involving $\widehat{\Sigma}$ are Pillai's trace, the Lawley-Hotelling trace (and Hotelling's T^2), and Roy's maximum root. We will not discuss these here, and the interested reader is referred to a text on multivariate analysis.

In multivariate analysis, there are other properties of the model that are subject to statistical inference. For example, we may wish to estimate or test the rank of the coefficient matrix, B. Even in the case of a single multivariate random variable, we may wish to test whether the variance-covariance matrix is of full rank. If it is not, there are fixed relationships among the elements of the random variable, and the distribution is said to be singular. We will discuss the problem of testing the rank of a matrix briefly in Sect. 9.5.5, beginning on page 433, but for more discussion on issues of statistical inference, we again refer the reader to a text on multivariate statistical inference.

9.4 Principal Components

The analysis of multivariate data involves various linear transformations that help in understanding the relationships among the features that the data represent. The second moments of the data are used to accommodate the differences in the scales of the individual variables and the covariances among pairs of variables.

If X is the matrix containing the data stored in the usual way, a useful statistic is the sums of squares and cross products matrix, $X^{\mathrm{T}}X$, or the "adjusted" squares and cross products matrix, $X_{\mathrm{c}}^{\mathrm{T}}X_{\mathrm{c}}$, where X_{c} is the centered matrix formed by subtracting from each element of X the mean of the column containing that element. The sample variance-covariance matrix, as in equation (8.67), is the Gramian matrix

$$S_X = \frac{1}{n-1}X_{\mathrm{c}}^{\mathrm{T}}X_{\mathrm{c}}, \tag{9.44}$$

where n is the number of observations (the number of rows in X).

In data analysis, the sample variance-covariance matrix S_X in equation (9.44) plays an important role. In more formal statistical inference, it is a consistent estimator of the population variance-covariance matrix (if it is positive definite), and under assumptions of independent sampling from a normal distribution, it has a known distribution. It also has important numerical properties; it is symmetric and positive definite (or, at least, nonnegative definite; see Sect. 8.6). Other estimates of the variance-covariance matrix or the correlation matrix of the underlying distribution may not be positive definite, however, and in Sect. 9.5.6 and Exercise 9.15 we describe possible ways of adjusting a matrix to be positive definite.

9.4.1 Principal Components of a Random Vector

It is often of interest to transform a given random vector into a vector whose elements are independent. We may also be interested in which of those elements of the transformed random vector have the largest variances. The transformed

vector may be more useful in making inferences about the population. In more informal data analysis, it may allow use of smaller observational vectors without much loss in information.

Stating this more formally, if Y is a random d-vector with variance-covariance matrix Σ, we seek a transformation matrix A such that $\widetilde{Y} = AY$ has a diagonal variance-covariance matrix. We are additionally interested in a transformation $a^{\mathrm{T}}Y$ that has maximal variance for a given $\|a\|$.

Because the variance of $a^{\mathrm{T}}Y$ is $\mathrm{V}(a^{\mathrm{T}}Y) = a^{\mathrm{T}}\Sigma a$, we have already obtained the solution in equation (3.265). The vector a is the eigenvector corresponding to the maximum eigenvalue of Σ, and if a is normalized, the variance of $a^{\mathrm{T}}Y$ is the maximum eigenvalue.

Because Σ is symmetric, it is orthogonally diagonalizable and the properties discussed in Sect. 3.8.10 on page 153 not only provide the transformation immediately but also indicate which elements of $\widetilde{Y}$ have the largest variances. We write the orthogonal diagonalization of Σ as (see equation (3.252))

$$\Sigma = \Gamma \Lambda \Gamma^{\mathrm{T}}, \tag{9.45}$$

where $\Gamma\Gamma^{\mathrm{T}} = \Gamma^{\mathrm{T}}\Gamma = I$, and Λ is diagonal with elements $\lambda_1 \geq \cdots \geq \lambda_m \geq 0$ (because a variance-covariance matrix is nonnegative definite). Choosing the transformation as

$$\widetilde{Y} = \Gamma^{\mathrm{T}}Y, \tag{9.46}$$

we have $\mathrm{V}(\widetilde{Y}) = \Lambda$; that is, the i^{th} element of $\widetilde{Y}$ has variance λ_i, and

$$\mathrm{Cov}(\widetilde{Y}_i, \widetilde{Y}_j) = 0 \quad \text{if } i \neq j.$$

The elements of $\widetilde{Y}$ are called the *principal components* of Y. The *first principal component*, $\widetilde{Y}_1$, which is the signed magnitude of the projection of Y in the direction of the eigenvector corresponding to the maximum eigenvalue, has the maximum variance of any of the elements of $\widetilde{Y}$, and $\mathrm{V}(\widetilde{Y}_1) = \lambda_1$. (It is, of course, possible that the maximum eigenvalue is not simple. In that case, there is no one-dimensional first principal component. If m_1 is the multiplicity of λ_1, all one-dimensional projections within the m_1-dimensional eigenspace corresponding to λ_1 have the same variance, and m_1 projections can be chosen as mutually independent.)

The second and third principal components, and so on, are likewise determined directly from the spectral decomposition.

9.4.2 Principal Components of Data

The same ideas of principal components in probability models carry over to observational data. Given an $n \times d$ data matrix X, we seek a transformation as above that will yield the linear combination of the columns that has maximum sample variance, and other linear combinations that are independent. This means that we work with the centered matrix X_{c} (equation (8.64)) and the

variance-covariance matrix S_X, as above, or the centered and scaled matrix X_{cs} (equation (8.65)) and the correlation matrix R_X (equation (8.69)). See Section 3.3 in Jolliffe (2002) for discussions of the differences in using the centered but not scaled matrix and using the centered and scaled matrix.

In the following, we will use S_X, which plays a role similar to Σ for the random variable. (This role could be stated more formally in terms of statistical estimation. Additionally, the scaling may require more careful consideration. The issue of scaling naturally arises from the arbitrariness of units of measurement in data. Random variables discussed in Sect. 9.4.1 have no units of measurement.)

In data analysis, we seek a normalized transformation vector a to apply to any centered observation x_c, so that the sample variance of $a^{\mathrm{T}}x_c$, that is,

$$a^{\mathrm{T}}S_X a, \tag{9.47}$$

is maximized.

From equation (3.265) or the spectral decomposition equation (3.256), we know that the solution to this maximization problem is the eigenvector, v_1, corresponding to the largest eigenvalue, c_1, of S_X, and the value of the expression (9.47); that is, $v_1^{\mathrm{T}}S_X v_1$ at the maximum is the largest eigenvalue. In applications, this vector is used to transform the rows of X_{c} into scalars. If we think of a generic row of X_{c} as the vector x, we call $v_1^{\mathrm{T}}x$ the *first principal component* of x. There is some ambiguity about the precise meaning of "principal component". The definition just given is a scalar; that is, a combination of values of a vector of variables. This is consistent with the definition that arises in the population model in Sect. 9.4.1. Sometimes, however, the eigenvector v_1 itself is referred to as the first principal component. More often, the vector $X_{\mathrm{c}}v_1$ of linear combinations of the columns of X_{c} is called the first principal component. We will often use the term in this latter sense.

If the largest eigenvalue, c_1, is of algebraic multiplicity $m_1 > 1$, we have seen that we can choose m_1 orthogonal eigenvectors that correspond to c_1 (because S_X, being symmetric, is simple). Any one of these vectors may be called a first principal component of X.

The second and third principal components, and so on, are likewise determined directly from the nonzero eigenvalues in the spectral decomposition of S_X. Because the eigenvectors are orthogonal (or can be chosen to be), the principal components have the property

$$z_i^{\mathrm{T}}S_X z_j = z_i^{\mathrm{T}}z_j = 0, \quad \text{for } i \neq j.$$

The full set of principal components of X_{c}, analogous to equation (9.46) except that here the random vectors correspond to the rows in X_{c}, is

$$Z = X_{\mathrm{c}}V, \tag{9.48}$$

where V has r_X columns. (As before, r_X is the rank of X.). Figure 9.2 shows two principal components, z_1 and z_2, formed from the data represented in x_1 and x_2.

9.4.2.1 Principal Components Directly from the Data Matrix

Formation of the S_X matrix emphasizes the role that the sample covariances play in principal component analysis. However, there is no reason to form

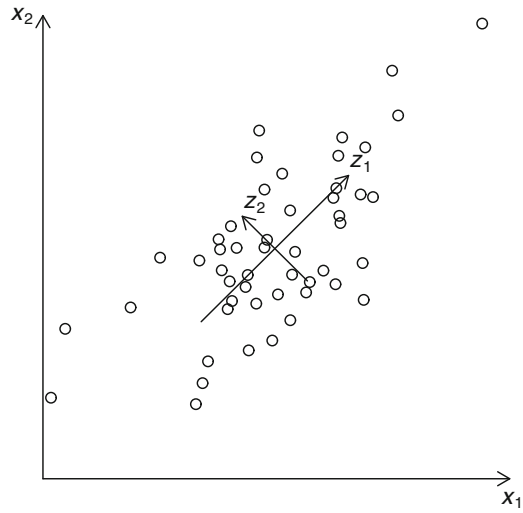


Figure 9.2. Principal components

a matrix such as $X_c^T X_c$, and indeed we may introduce significant rounding errors by doing so. (Recall our previous discussions of the condition numbers of $X^T X$ and X.)

The singular value decomposition of the $n \times m$ matrix X_c yields the square roots of the eigenvalues of $X_c^T X_c$ and the same eigenvectors. (The eigenvalues of $X_c^T X_c$ are $(n-1)$ times the eigenvalues of S_X.) We will assume that there are more observations than variables (that is, that $n > m$). In the SVD of the centered data matrix $X_c = UAV^T$, U is an $n \times r_X$ matrix with orthogonal columns, V is an $m \times r_X$ matrix whose first r_X columns are orthogonal and the rest are 0, and A is an $r_X \times r_X$ diagonal matrix whose entries are the nonnegative singular values of $X - \overline{X}$. (As before, r_X is the column rank of X.)

The spectral decomposition in terms of the singular values and outer products of the columns of the factor matrices is

$$X_c = \sum_i^{r_X} \sigma_i u_i v_i^T. \tag{9.49}$$

The vectors u_i are the same as the eigenvectors of S_X.

9.4.2.2 Dimension Reduction

If the columns of a data matrix X are viewed as variables or features that are measured for each of several observational units, which correspond to rows in the data matrix, an objective in principal components analysis may be to determine some small number of linear combinations of the columns of X that contain almost as much information as the full set of columns. (Here we are not using "information" in a precise sense; in a general sense, it means having similar statistical properties.) Instead of a space of dimension equal to the (column) rank of X (that is, r_X), we seek a subspace of span(X) with rank less than r_X that approximates the full space (in some sense). As we discussed on page 176, the best approximation in terms of the usual norm (the Frobenius norm) of X_c by a matrix of rank p is

$$\widetilde{X}_p = \sum_i^p \sigma_i u_i v_i^{\mathrm{T}} \tag{9.50}$$

for some $p < \min(n, m)$.

Principal components analysis is often used for "dimension reduction" by using the first few principal components in place of the original data. There are various ways of choosing the number of principal components (that is, p in equation (9.50)). There are also other approaches to dimension reduction. A general reference on this topic is Mizuta (2012).

9.5 Condition of Models and Data

In Sect. 6.1, we describe the concept of "condition" of a matrix for certain kinds of computations. In Sect. 6.1.3, we discuss how a large condition number may indicate the level of numerical accuracy in the solution of a system of linear equations, and on page 292 we extend this discussion to overdetermined systems such as those encountered in regression analysis. (We return to the topic of condition in Sect. 11.2 with even more emphasis on the numerical computations.) The condition of the X matrices has implications for the accuracy we can expect in the numerical computations for regression analysis.

There are other connections between the condition of the data and statistical analysis that go beyond just the purely computational issues. Analysis involves more than just computations. Ill-conditioned data also make interpretation of relationships difficult because we may be concerned with both conditional and marginal relationships. In ill-conditioned data, the relationships between any two variables may be quite different depending on whether or not the relationships are conditioned on relationships with other variables in the dataset.

9.5.1 Ill-Conditioning in Statistical Applications

We have described ill-conditioning heuristically as a situation in which small changes in the input data may result in large changes in the solution. Ill-conditioning in statistical modeling is often the result of high correlations among the independent variables. When such correlations exist, the computations may be subject to severe rounding error. This was a problem in using computer software many years ago, as Longley (1967) pointed out. When there are large correlations among the independent variables, the model itself must be examined, as Beaton, Rubin, and Barone (1976) emphasize in reviewing the analysis performed by Longley. Although the work of Beaton, Rubin, and Barone was criticized for not paying proper respect to high-accuracy computations, ultimately it is the utility of the fitted model that counts, not the accuracy of the computations.

Large correlations are reflected in the condition number of the X matrix. A large condition number may indicate the possibility of harmful numerical errors. Some of the techniques for assessing the accuracy of a computed result may be useful. In particular, the analyst may try the suggestion of Mullet and Murray (1971) to regress $y + dx_j$ on $x_1, \ldots, x_m$, and compare the results with the results obtained from just using y.

Other types of ill-conditioning may be more subtle. Large variations in the leverages may be the cause of ill-conditioning.

Often, numerical problems in regression computations indicate that the linear model may not be entirely satisfactory for the phenomenon being studied. Ill-conditioning in statistical data analysis often means that the approach or the model is not appropriate.

9.5.2 Variable Selection

Starting with a model such as equation (9.9),

$$Y = \beta^{\mathrm{T}} x + E,$$

we are ignoring the most fundamental problem in data analysis: which variables *are really related* to Y, and *how are they related*?

We often begin with the premise that a linear relationship is at least a good approximation locally; that is, with restricted ranges of the variables. This leaves us with one of the most important tasks in linear regression analysis: selection of the variables to include in the model. There are many statistical issues that must be taken into consideration. We will not discuss these issues here; rather we refer the reader to a comprehensive text on regression analysis, such as Draper and Smith (1998), or to a text specifically on this topic, such as Miller (2002). Some aspects of the statistical analysis involve tests of linear hypotheses, such as discussed in Sect. 9.3.3. There is a major difference, however; those tests were based on knowledge of the *correct* model. The basic

problem in variable selection is that we do not know the correct model. Most reasonable procedures to determine the correct model yield biased statistics. Some people attempt to circumvent this problem by recasting the problem in terms of a "full" model; that is, one that includes all independent variables that the data analyst has looked at. (Looking at a variable and then making a decision to exclude that variable from the model can bias further analyses.)

We generally approach the variable selection problem by writing the model with the data as

$$y = X_{\mathrm{i}}\beta_{\mathrm{i}} + X_{\mathrm{o}}\beta_{\mathrm{o}} + \epsilon, \tag{9.51}$$

where X_{i} and X_{o} are matrices that form some permutation of the columns of X, $X_{\mathrm{i}}|X_{\mathrm{o}} = X$, and β_{i} and β_{o} are vectors consisting of corresponding elements from β. (The i and o are "in" and "out".) We then consider the model

$$y = X_{\mathrm{i}}\beta_{\mathrm{i}} + \epsilon_{\mathrm{i}}. \tag{9.52}$$

It is interesting to note that the least squares estimate of β_{i} in the model (9.52) is the same as the least squares estimate in the model

$$\hat{y}_{\mathrm{io}} = X_{\mathrm{i}}\beta_{\mathrm{i}} + \epsilon_{\mathrm{i}},$$

where $\hat{y}_{\mathrm{io}}$ is the vector of predicted values obtained by fitting the full model (9.51). An interpretation of this fact is that fitting the model (9.52) that includes only a subset of the variables is the same as using that subset to *approximate* the predictions of the full model. The fact itself can be seen from the normal equations associated with these two models. We have

$$X_{\mathrm{i}}^{\mathrm{T}}X(X^{\mathrm{T}}X)^{-1}X^{\mathrm{T}} = X_{\mathrm{i}}^{\mathrm{T}}. \tag{9.53}$$

This follows from the fact that $X(X^{\mathrm{T}}X)^{-1}X^{\mathrm{T}}$ is a projection matrix, and X_{i} consists of a set of columns of X (see Sect. 8.5 and Exercise 9.12 on page 455).

As mentioned above, there are many difficult statistical issues in the variable selection problem. The exact methods of statistical inference generally do not apply (because they are based on a model, and we are trying to choose a model). In variable selection, as in any statistical analysis that involves the choice of a model, the effect of the given dataset may be greater than warranted, resulting in overfitting. One way of dealing with this kind of problem is to use part of the dataset for fitting and part for validation of the fit. There are many variations on exactly how to do this, but in general, "cross validation" is an important part of any analysis that involves building a model.

The computations involved in variable selection are the same as those discussed in Sects. 9.3.3 and 9.3.7.

9.5.3 Principal Components Regression

A somewhat different approach to the problem of variable selection involves selecting some linear combinations of all of the variables. The first p principal components of X cover the space of span(X) optimally (in some sense),

and so these linear combinations themselves may be considered as the "best" variables to include in a regression model. If V_p is the first p columns from V in the full set of principal components of X, equation (9.48), we use the regression model

$$y \approx Z_p\gamma, \tag{9.54}$$

where

$$Z_p = XV_p. \tag{9.55}$$

This is the idea of principal components regression.

In principal components regression, even if $p < m$ (which is the case, of course; otherwise principal components regression would make no sense), all of the original variables are included in the model. Any linear combination forming a principal component may include all of the original variables. The weighting on the original variables tends to be such that the coefficients of the original variables that have extreme values in the ordinary least squares regression are attenuated in the principal components regression using only the first p principal components.

The principal components do not involve y, so it may not be obvious that a model using only a set of principal components selected without reference to y would yield a useful regression model. Indeed, sometimes important independent variables do not get sufficient weight in principal components regression.

9.5.4 Shrinkage Estimation

As mentioned in the previous section, instead of selecting specific independent variables to include in the regression model, we may take the approach of shrinking the coefficient estimates toward zero. This of course has the effect of introducing a bias into the estimates (in the case of a true model being used), but in the process of reducing the inherent instability due to collinearity in the independent variables, it may also reduce the mean squared error of linear combinations of the coefficient estimates. This is one approach to the problem of overfitting.

The shrinkage can also be accomplished by a regularization of the fitting criterion. If the fitting criterion is minimization of a norm of the residuals, we add a norm of the coefficient estimates to minimize

$$\|r(b)\|_f + \lambda\|b\|_b, \tag{9.56}$$

where λ is a tuning parameter that allows control over the relative weight given to the two components of the objective function. This regularization is also related to the variable selection problem by the association of superfluous variables with the individual elements of the optimal b that are close to zero.

9.5.4.1 Ridge Regression

If the fitting criterion is least squares, we may also choose an L_2 norm on b, and we have the fitting problem

$$\min_b \left((y - Xb)^T(y - Xb) + \lambda b^T b\right). \tag{9.57}$$

This is called Tikhonov regularization (from A. N. Tikhonov), and it is by far the most commonly used regularization. This minimization problem yields the modified normal equations

$$(X^T X + \lambda I)b = X^T y, \tag{9.58}$$

obtained by adding λI to the sums of squares and cross products matrix. This is the ridge regression we discussed on page 364, and as we saw in Sect. 6.1, the addition of this positive definite matrix has the effect of reducing numerical ill-conditioning.

Interestingly, these normal equations correspond to a least squares approximation for

$$\begin{pmatrix} y \\ 0 \end{pmatrix} \approx \begin{bmatrix} X \\ \sqrt{\lambda} I \end{bmatrix} \beta. \tag{9.59}$$

(See Exercise 9.11.) The shrinkage toward 0 is evident in this formulation. Because of this, we say the "effective" degrees of freedom of a ridge regression model decreases with increasing λ. In equation (8.61), we formally defined the *effective model degrees of freedom* of any linear fit

$$\widehat{y} = S_\lambda y$$

as

$$\mathrm{tr}(S_\lambda),$$

and we saw in equation (8.62) that indeed it does decrease with increasing λ.

Even if all variables are left in the model, the ridge regression approach may alleviate some of the deleterious effects of collinearity in the independent variables.

9.5.4.2 Lasso Regression

The norm for the regularization in expression (9.56) does not have to be the same as the norm applied to the model residuals. An alternative fitting criterion, for example, is to use an L_1 norm,

$$\min_b (y - Xb)^T(y - Xb) + \lambda \|b\|_1.$$

Rather than strictly minimizing this expression, we can formulate a constrained optimization problem

$$\min_{\|b\|_1<t} (y - Xb)^{\mathrm{T}}(y - Xb), \tag{9.60}$$

for some tuning constant t. The solution of this quadratic programming problem yields a b with some elements identically 0, depending on t. As t decreases, more elements of the optimal b are identically 0, and thus this is an effective method for variable selection. The use of expression (9.60) is called lasso regression. ("Lasso" stands for "least absolute shrinkage and selection operator".)

Lasso regression is computationally expensive if several values of t are explored. Efron et al. (2004) propose "least angle regression" (LAR), the steps of which effectively yield the entire lasso regularization path.

9.5.5 Statistical Inference about the Rank of a Matrix

An interesting problem in numerical linear algebra is to approximate the rank of a given matrix. A related problem in statistical inference is to estimate or to test an hypothesis concerning the rank of an unknown matrix. For example, in the multivariate regression model discussed in Sect. 9.3.9 (beginning on page 420), we may wish to test whether the coefficient matrix B is of full rank.

In statistical inference, we use observed data to make inferences about a model, but we do not "estimate" or "test an hypothesis" concerning the rank of a given matrix of data.

9.5.5.1 Numerical Approximation and Statistical Inference

The rank of a matrix is not a continuous function of the elements of the matrix. It is often difficult to compute the rank of a given matrix; hence, we often seek to approximate the rank. We alluded to the problem of approximating the rank of a matrix on page 252, and indicated that a QR factorization of the given matrix might be an appropriate approach to the problem. (In Sect. 11.4, we discuss the rank-revealing QR (or LU) method for approximating the rank of a matrix.)

The SVD can also be used to approximate the rank of a given $n \times m$ matrix. The approximation would be based on a decision that either the rank is $\min(n, m)$, or that the rank is r because $d_i = 0$ for $i > r$ in the decomposition UDV^{T} given in equation (3.276) on page 161.

Although we sometimes refer to the problem as one of "estimating the rank of a matrix", "estimation" in the numerical-analytical sense refers to "approximation", rather than to statistical estimation. This is an important distinction that is often lost. Estimation and testing in a statistical sense do not apply to a given entity; these methods of inference apply to properties

of a random variable. We use observed realizations of the random variable to make inferences about unobserved properties or parameters that describe the distribution of the random variable.

A statistical test is a decision rule for rejection of an hypothesis about which empirical evidence is available. The empirical evidence consists of observations on some random variable, and the hypothesis is a statement about the distribution of the random variable. In simple cases of hypothesis testing, the distribution is assumed to be characterized by a parameter, and the hypothesis merely specifies the value of that parameter. The statistical test is based on the distribution of the underlying random variable if the hypothesis is true.

9.5.5.2 Statistical Tests of the Rank of a Class of Matrices

Most common statistical tests involve hypotheses concerning a scalar parameter. We have encountered two examples that involve tests of hypotheses concerning matrix parameters. One involved tests of the variance-covariance matrix Σ in a multivariate distribution (Exercise 4.12 on page 224), and the other was for tests of the coefficient matrix B in multivariate linear regression (see page 423). The tests of the variance-covariance matrix are based on a Wishart matrix W, but for a specific hypothesis, the test statistic is a chi-squared statistic. In the multivariate linear regression testing problem, the least-squares estimator of the coefficient matrix, which has a matrix normal distribution, is used to form two matrices that have independent Wishart distributions. The hypotheses of interest are that certain elements of the coefficient matrix are zero, and the test statistics involve functions of the Wishart matrices, such as Wilk's Λ, which is the ratio of the determinants.

In multivariate linear regression, given n observations on the vectors y and x, we use the model for the data given in equation (9.34), on page 421,

$$Y = XB + E,$$

where Y is an $n \times d$ matrix and X is an $n \times m$ matrix of observations, B is an $m \times d$ unknown matrix, E is an $n \times d$ matrix of n unobserved realizations of a d-variate random variable. The canonical problem in statistical applications is to test whether $B = 0$, that is, whether there is any linear relationship between y and x. A related but less encompassing question is whether B is of full rank. (If $B = 0$, its rank is zero.) Testing whether B is of full rank is similar to the familiar univariate statistical problem of testing if some elements of β in the model $y = x^{\mathrm{T}}\beta + \epsilon$ are zero. In the multivariate case, this is sometimes referred to as the "reduced rank regression" problem. The null hypothesis of interest is

$$H_0 : \ \mathrm{rank}(B) \leq \min(m, d) - 1.$$

One approach is to test sequentially the null hypotheses $H_{0_i} : \ \mathrm{rank}(B) = i$ for $i = 1, \ldots \min(m, d) - 1$.

The other problem referred to above is, given n d-vectors $y_1, \ldots y_n$ assumed to be independent realizations of random vectors distributed as $\mathrm{N}_d(\mu, \Sigma)$, to test whether Σ is of full rank (that is, whether the multivariate normal distribution is singular; see page 219).

In other applications, in vector stochastic processes, the matrix of interest is one that specifies the relationship of one time series to another. In such applications the issue of stationarity is important, and may be one of the reasons for performing the rank test.

The appropriate statistical models in these settings are different, and the forms of the models in the different applications affect the distributions of any test statistics. The nature of the subject of the hypothesis, that is, the rank of a matrix, poses some difficulty. Much of the statistical theory on hypothesis testing involves an open parameter space over a dense set of reals, but of course the rank is an integer. Because of this, even if for no other reason, we would not expect to be able to work out an exact distribution of any estimator or test statistic for the rank. At best we would seek an estimator or test statistic for which we could derive, or at least approximate, an asymptotic distribution.

Problems of testing the rank of a matrix have been addressed in the statistical literature for some time; see, for example, Anderson (1951), Gill and Lewbel (1992), and Cragg and Donald (1996). They have also been discussed frequently in econometric applications; see, for example, Robin and Smith (2000) and Kleibergen and Paap (2006). In some of the literature, it is not clear whether or not the authors are describing a test for the rank of a given matrix, which, as pointed out above, is not a statistical procedure, even if a "test statistic" and a "null probability distribution" are involved.

My purpose in this section is not to review the various approaches to statistical inference about the rank of matrices or to discuss the "best" tests under various scenarios, but rather to describe one test in order to give the flavor of the approaches.

9.5.5.3 Statistical Tests of the Rank Based on an LDU Factorization

Gill and Lewbel (1992) and Cragg and Donald (1996) describe tests of the rank of a matrix that use factors from an LDU factorization. For an $m \times d$ matrix Θ, the tests are of the null hypothesis $H_0 : \mathrm{rank}(\Theta) = r$, where $r < \min(m, d)$. (There are various ways an alternative hypothesis could be phrased, but we will not specify one here.)

We first decompose the unknown matrix Θ as in equation (5.32), using permutation matrices so that the diagonal elements of D are nonincreasing in absolute value: $E_{(\pi_1)}\Theta E_{(\pi_2)} = LDU$.

The $m \times d$ matrix Θ (with $m \geq d$ without loss of generality) can be decomposed as

$$E_{(\pi_1)}\Theta E_{(\pi_2)} = LDU$$
$$= \begin{bmatrix} L_{11} & 0 & 0 \\ L_{21} & L_{22} & 0 \\ L_{31} & L_{32} & I_{m-d} \end{bmatrix} \begin{bmatrix} D_1 & 0 & 0 \\ 0 & D_2 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \\ 0 & U_{22} \\ 0 & 0 \end{bmatrix}, \quad (9.61)$$

where the unknown matrices L_{11}, U_{11}, and D_1 are $r \times r$, and the elements of the diagonal submatrices D_1 and D_2 are arranged in nonincreasing order. If the rank of Θ is r, $D_2 = 0$, but no diagonal element of D_1 is 0.

For a statistical test of the rank of Θ, we need to identify an observable random variable (random vector) whose distribution depends on Θ. To proceed, we take a sample of realizations of this random variable. We let $\widehat{\Theta}$ be an estimate of Θ based on n such realizations, and assume the central limit property,

$$\sqrt{k}\,\mathrm{vec}(\widehat{\Theta} - \Theta) \to_{\mathrm{d}} \mathrm{N}(0, V), \quad (9.62)$$

where V is $nm \times nm$ and positive definite. (For example, if B is the coefficient matrix in the multivariate linear regression model (9.34) and $\widehat{B}$ is the least-squares estimator from expression (9.37), then $\widehat{B}$ and B have this property. If Σ is the variance-covariance matrix in the multivariate normal distribution, expression (4.74), then we use the sample variance-covariance matrix, equation (8.67), page 367; but in this case, the analogous asymptotic distribution relating $\widehat{\Sigma}$ and Σ is a Wishart distribution.)

Now if $D_2 = 0$ (that is, if Θ has rank r) and $\widehat{\Theta}$ is decomposed in the same way as Θ in equation (9.61), then

$$\sqrt{k}\,\mathrm{diag}(\widehat{D}_2) \to_{\mathrm{d}} \mathrm{N}(0, W)$$

for some positive definite matrix W, and the quantity

$$n\widehat{d}_2^{\mathrm{T}} W^{-1} \widehat{d}_2, \quad (9.63)$$

where

$$\widehat{d}_2 = \mathrm{diag}(\widehat{D}_2),$$

has an asymptotic chi-squared distribution with $(m - r)$ degrees of freedom. If a consistent and independent estimator of W, say $\widehat{W}$, is used in place of W in the expression (9.63), this would be a test statistic for the hypothesis that the rank of Θ is r. (Note that W is $m - r \times m - r$.)

Gill and Lewbel (1992) derive a consistent estimator to use in expression (9.63) as a test statistic. Following their derivation, first let $\widehat{V}$ be a consistent estimator of V. (It would typically be a sample variance-covariance matrix.) Then

$$\left(\widehat{Q}^{\mathrm{T}} \otimes \widehat{P}\right) \widehat{V} \left(\widehat{Q} \otimes \widehat{P}^{\mathrm{T}}\right)$$

is a consistent estimator of the variance-covariance of $\mathrm{vec}(\widehat{P}(\widehat{\Theta} - \Theta)\widehat{Q})$. Next, define the matrices

$$\widehat{H} = \left[-\widehat{L}_{22}^{-1}\widehat{L}_{21}\widehat{L}_{11}^{-1} \;\middle|\; \widehat{L}_{22}^{-1} \;\middle|\; 0 \right],$$

$$\widehat{K} = \begin{bmatrix} -\widehat{U}_{11}^{-1}\widehat{U}_{12}\widehat{U}_{22}^{-1} \\ \widehat{U}_{22}^{-1} \end{bmatrix},$$

and T such that

$$\text{vec}(\widehat{D}_2) = T\widehat{d}_2.$$

The matrix T is $(m-r)^2 \times (m-r)$, consisting of a stack of square matrices with 0s in all positions except for a 1 in one diagonal element. The matrix is orthogonal; that is,

$$T^{\mathrm{T}}T = I_{m-r}.$$

The matrix

$$(\widehat{K} \otimes \widehat{H}^{\mathrm{T}})T$$

transforms $\text{vec}(\widehat{P}(\widehat{\Theta} - \Theta)\widehat{Q})$ into $\widehat{d}_2$; hence the variance-covariance estimator, $(\widehat{Q}^{\mathrm{T}} \otimes \widehat{P})\widehat{V}(\widehat{Q} \otimes \widehat{P}^{\mathrm{T}})$, is adjusted by this matrix. The estimator $\widehat{W}$ therefore is given by

$$\widehat{W} = T^{\mathrm{T}}(\widehat{K}^{\mathrm{T}} \otimes \widehat{H})(\widehat{Q}^{\mathrm{T}} \otimes \widehat{P})\widehat{V}(\widehat{Q} \otimes \widehat{P}^{\mathrm{T}})(\widehat{K} \otimes \widehat{H}^{\mathrm{T}})T.$$

The test statistic is

$$n\widehat{d}_2^{\mathrm{T}}\widehat{W}^{-1}\widehat{d}_2, \tag{9.64}$$

with an approximate chi-squared distribution with $(m-r)$ degrees of freedom.

Cragg and Donald (1996), however, have pointed out that the indeterminacy of the LDU decomposition casts doubts on the central limiting distribution in (9.62). Kleibergen and Paap (2006) proposed a related test for a certain class of matrices based on the SVD. Because the SVD is unique (within the limitations mentioned on page 163), it does not suffer from the indeterminacy.

9.5.6 Incomplete Data

Missing values in a dataset can not only result in ill-conditioned problems but can cause some matrix statistics to lack their standard properties, such as covariance or correlation matrices formed from the available data not being positive definite.

In the standard flat data file represented in Fig. 8.1, where a row holds data from a given observation and a column represents a specific variable or feature, it is often the case that some values are missing for some observation/variable combination. This can occur for various reasons, such as a failure of a measuring device, refusal to answer a question in a survey, or an indeterminate or infinite value for a derived variable (for example, a coefficient of

variation when the mean is 0). This causes problems for our standard storage of data in a matrix. The values for some cells are not available.

The need to make provisions for missing data is one of the important differences between statistical numerical processing and ordinary numerical analysis. First of all, we need a method for representing a "not available" (NA) value, and then we need a mechanism for avoiding computations with this NA value. There are various ways of doing this, including the use of special computer numbers (see pages 464 and 475).

The layout of the data may be of the form

$$X = \begin{bmatrix} \mathtt{X} & \mathtt{X} & \mathtt{NA} \\ \mathtt{X} & \mathtt{NA} & \mathtt{NA} \\ \mathtt{X} & \mathtt{NA} & \mathtt{X} \\ \mathtt{X} & \mathtt{X} & \mathtt{X} \end{bmatrix}. \tag{9.65}$$

In the data matrix of equation (9.65), all rows could be used for summary statistics relating to the first variable, but only two rows could be used for summary statistics relating to the second and third variables. For summary statistics such as the mean or variance for any one variable, it would seem to make sense to use all of the available data.

The picture is not so clear, however, for statistics on two variables, such as the covariance. If all observations that contain data on both variables are used for computing the covariance, then the covariance matrix may not be positive definite. If the correlation matrix is computed using covariances computed in this way but variances computed on all of the data, some off-diagonal elements may be larger than 1. If the correlation matrix is computed using covariances from all available pairs and variances computed only from the data in complete pairs (that is, the variances used in computing correlations involving a given variable are different for different variables), then no off-diagonal element can be larger than 1, but the correlation matrix may not be nonnegative definite.

An alternative, of course, is to use only data in records that are complete. This is called "casewise deletion", whereas use of all available data for bivariate statistics is called "pairwise deletion". One must be very careful in computing bivariate statistics from data with missing values; see Exercise 9.14 (and a solution on page 612).

Estimated or approximate variance-covariance or correlation matrices that are not positive definite can arise in other ways in applications. For example, the data analyst may have an estimate of the correlation matrix that was not based on a single sample.

Various approaches to handling an approximate correlation matrix that is not positive definite have been considered. Devlin et al. (1975) describe a method of shrinking the given R toward a chosen positive definite matrix, R_1, which may be an estimator of a correlation matrix computed in other ways (perhaps a robust estimator) or may just be chosen arbitrarily; for example, R_1 may just be the identity matrix. The method is to choose the largest value α in $[0, 1]$ such that the matrix

$$\widetilde{R} = \alpha R + (1 - \alpha) R_1 \tag{9.66}$$

is positive definite. This optimization problem can be solved iteratively starting with $\alpha = 1$ and decreasing α in small steps while checking whether $\widetilde{R}$ is positive definite. (The checks may require several computations.) A related method is to use a modified Cholesky decomposition. If the symmetric matrix S is not positive definite, a diagonal matrix D can be determined so that $S + D$ is positive definite. Eskow and Schnabel (1991), for example, describe one way to determine D with values near zero and to compute a Cholesky decomposition of $S + D$.

Devlin, Gnanadesikan, and Kettenring (1975) also describe nonlinear shrinking methods in which all of the off-diagonal elements r_{ij} are replaced iteratively, beginning with $r_{ij}^{(0)} = r_{ij}$ and proceeding with

$$r_{ij}^{(k)} = \begin{cases} f^{-1}\left(f\left(r_{ij}^{(k-1)}\right) + \delta\right) & \text{if } r_{ij}^{(k-1)} < -f^{-1}(\delta) \\ 0 & \text{if } \left|r_{ij}^{(k-1)}\right| \le f^{-1}(\delta) \\ f^{-1}\left(f\left(r_{ij}^{(k-1)}\right) - \delta\right) & \text{if } r_{ij}^{(k-1)} > f^{-1}(\delta) \end{cases} \tag{9.67}$$

for some invertible positive-valued function f and some small positive constant δ (for example, 0.05). The function f may be chosen in various ways; one suggested function is the hyperbolic tangent, which makes f^{-1} Fisher's variance-stabilizing function for a correlation coefficient; see Exercise 9.19b.

Rousseeuw and Molenberghs (1993) suggest a method in which some approximate correlation matrices can be adjusted to a nearby correlation matrix, where closeness is determined by the Frobenius norm. Their method applies to pseudo-correlation matrices. Recall that any symmetric nonnegative definite matrix with ones on the diagonal is a correlation matrix. A *pseudo-correlation matrix* is a symmetric matrix R with positive diagonal elements (but not necessarily 1s) and such that $r_{ij}^2 \le r_{ii} r_{jj}$. (This is inequality (8.12), which is a necessary but not sufficient condition for the matrix to be nonnegative definite.)

The method of Rousseeuw and Molenberghs adjusts an $m \times m$ pseudo-correlation matrix R to the closest correlation matrix $\widetilde{R}$, where closeness is determined by the Frobenius norm; that is, we seek $\widetilde{R}$ such that

$$\|R - \widetilde{R}\|_{\mathrm{F}} \tag{9.68}$$

is minimum over all choices of $\widetilde{R}$ that are correlation matrices (that is, matrices with 1s on the diagonal that are positive definite). The solution to this optimization problem is not as easy as the solution to the problem we consider on page 176 of finding the best approximate matrix of a given rank. Rousseeuw and Molenberghs describe a computational method for finding $\widetilde{R}$ to minimize

expression (9.68). A correlation matrix $\widetilde{R}$ can be formed as a Gramian matrix formed from a matrix U whose columns, $u_1, \ldots, u_m$, are normalized vectors, where

$$\tilde{r}_{ij} = u_i^{\mathrm{T}} u_j.$$

If we choose the vector u_i so that only the first i elements are nonzero, then they form the Cholesky factor elements of $\widetilde{R}$ with nonnegative diagonal elements,

$$\widetilde{R} = U^{\mathrm{T}} U,$$

and each u_i can be completely represented in $\mathbb{R}^i$. We can associate the $m(m-1)/2$ unknown elements of U with the angles in their spherical coordinates. In u_i, the j^{th} element is 0 if $j > i$ and otherwise is

$$\sin(\theta_{i1}) \cdots \sin(\theta_{i,i-j}) \cos(\theta_{i,i-j+1}),$$

where $\theta_{i1}, \ldots, \theta_{i,i-j}, \theta_{i,i-j+1}$ are the unknown angles that are the variables in the optimization problem for the Frobenius norm (9.68). The problem now is to solve

$$\min \sum_{i=1}^{m} \sum_{j=1}^{i} (r_{ij} - \sin(\theta_{i1}) \cdots \sin(\theta_{i,i-j}) \cos(\theta_{i,i-j+1}))^2. \tag{9.69}$$

This optimization problem is well-behaved and can be solved by steepest descent (see page 201). Rousseeuw and Molenberghs (1993) also mention that a weighted least squares problem in place of equation (9.69) may be more appropriate if the elements of the pseudo-correlation matrix R result from different numbers of observations.

In Exercise 9.15, we describe another way of converting an approximate correlation matrix that is not positive definite into a correlation matrix by iteratively replacing negative eigenvalues with positive ones.

9.6 Optimal Design

When an experiment is designed to explore the effects of some variables (usually called "factors") on another variable, the settings of the factors (independent variables) should be determined so as to yield a maximum amount of information from a given number of observations. The basic problem is to determine from a set of candidates the best rows for the data matrix X. For example, if there are six factors and each can be set at three different levels, there is a total of $3^6 = 729$ combinations of settings. In many cases, because of the expense in conducting the experiment, only a relatively small number of runs can be made. If, in the case of the 729 possible combinations, only 30 or so runs can be made, the scientist must choose the subset of combinations that will be most informative. A row in X may contain more elements than

just the number of factors (because of interactions), but the factor settings completely determine the row.

We may quantify the information in terms of variances of the estimators. If we assume a linear relationship expressed by

$$y = \beta_0 1 + X\beta + \epsilon$$

and make certain assumptions about the probability distribution of the residuals, the variance-covariance matrix of estimable linear functions of the least squares solution (9.15) is formed from

$$(X^{\mathrm{T}}X)^{-}\sigma^2.$$

(The assumptions are that the residuals are independently distributed with a constant variance, σ^2. We will not dwell on the statistical properties here, however.) If the emphasis is on estimation of β, then X should be of full rank. In the following, we assume X is of full rank; that is, that $(X^{\mathrm{T}}X)^{-1}$ exists.

An objective is to minimize the variances of estimators of linear combinations of the elements of β. We may identify three types of relevant measures of the variance of the estimator $\widehat{\beta}$: the average variance of the elements of $\widehat{\beta}$, the maximum variance of any elements, and the "generalized variance" of the vector $\widehat{\beta}$. The property of the design resulting from maximizing the information by reducing these measures of variance is called, respectively, A-optimality, E-optimality, and D-optimality. They are achieved when X is chosen as follows:

- A-optimality: minimize $\mathrm{tr}((X^{\mathrm{T}}X)^{-1})$.
- E-optimality: minimize $\rho((X^{\mathrm{T}}X)^{-1})$.
- D-optimality: minimize $\det((X^{\mathrm{T}}X)^{-1})$.

Using the properties of eigenvalues and determinants that we discussed in Chap. 3, we see that E-optimality is achieved by maximizing $\rho(X^{\mathrm{T}}X)$ and D-optimality is achieved by maximizing $\det(X^{\mathrm{T}}X)$.

9.6.1 D-Optimal Designs

The D-optimal criterion is probably used most often. If the residuals have a normal distribution (and the other distributional assumptions are satisfied), the D-optimal design results in the smallest volume of confidence ellipsoids for β. (See Titterington 1975; Nguyen and Miller 1992; and Atkinson and Donev 1992. Identification of the D-optimal design is related to determination of a minimum-volume ellipsoid for multivariate data.) The computations required for the D-optimal criterion are the simplest, and this may be another reason it is used often.

To construct an optimal X with a given number of rows, n, from a set of N potential rows, one usually begins with an initial choice of rows, perhaps random, and then determines the effect on the determinant by exchanging a

selected row with a different row from the set of potential rows. If the matrix X has n rows and the row vector x^{T} is appended, the determinant of interest is

$$\det(X^{\mathrm{T}}X + xx^{\mathrm{T}})$$

or its inverse. Using the relationship $\det(AB) = \det(A)\det(B)$, it is easy to see that

$$\det(X^{\mathrm{T}}X + xx^{\mathrm{T}}) = \det(X^{\mathrm{T}}X)(1 + x^{\mathrm{T}}(X^{\mathrm{T}}X)^{-1}x). \tag{9.70}$$

Now, if a row x_+^{T} is exchanged for the row x_-^{T}, the effect on the determinant is given by

$$\begin{aligned}\det(X^{\mathrm{T}}X + x_+x_+^{\mathrm{T}} - x_-x_-^{\mathrm{T}}) = \det(X^{\mathrm{T}}X) \times \\ \Big(1 + x_+^{\mathrm{T}}(X^{\mathrm{T}}X)^{-1}x_+ - \\ x_-^{\mathrm{T}}(X^{\mathrm{T}}X)^{-1}x_-(1 + x_+^{\mathrm{T}}(X^{\mathrm{T}}X)^{-1}x_+) + \\ (x_+^{\mathrm{T}}(X^{\mathrm{T}}X)^{-1}x_-)^2\Big)\end{aligned} \tag{9.71}$$

(see Exercise 9.8).

Following Miller and Nguyen (1994), writing $X^{\mathrm{T}}X$ as $R^{\mathrm{T}}R$ from the QR decomposition of X, and introducing z_+ and z_- as

$$Rz_+ = x_+$$

and

$$Rz_- = x_-,$$

we have the right-hand side of equation (9.71):

$$z_+^{\mathrm{T}}z_+ - z_-^{\mathrm{T}}z_-(1 + z_+^{\mathrm{T}}z_+) + (z_-^{\mathrm{T}}z_+)^2. \tag{9.72}$$

Even though there are $n(N-n)$ possible pairs (x_+, x_-) to consider for exchanging, various quantities in (9.72) need be computed only once. The corresponding (z_+, z_-) are obtained by back substitution using the triangular matrix R. Miller and Nguyen use the Cauchy-Schwarz inequality (2.26) (page 24) to show that the quantity (9.72) can be no larger than

$$z_+^{\mathrm{T}}z_+ - z_-^{\mathrm{T}}z_-; \tag{9.73}$$

hence, when considering a pair (x_+, x_-) for exchanging, if the quantity (9.73) is smaller than the largest value of (9.72) found so far, then the full computation of (9.72) can be skipped. Miller and Nguyen also suggest not allowing the last point added to the design to be considered for removal in the next iteration and not allowing the last point removed to be added in the next iteration.

The procedure begins with an initial selection of design points, yielding the $n \times m$ matrix $X^{(0)}$ that is of full rank. At the k^{th} step, each row of $X^{(k)}$ is considered for exchange with a candidate point, subject to the restrictions mentioned above. Equations (9.72) and (9.73) are used to determine the best exchange. If no point is found to improve the determinant, the process terminates. Otherwise, when the optimal exchange is determined, $R^{(k+1)}$ is formed using the updating methods discussed in the previous sections. (The programs of Gentleman 1974, referred to in Sect. 6.6.5 can be used.)

9.7 Multivariate Random Number Generation

The need to simulate realizations of random variables arises often in statistical applications, both in the development of statistical theory and in applied data analysis. In this section, we will illustrate only a couple of problems in multivariate random number generation. These make use of some of the properties we have discussed previously.

Most methods for random number generation assume an underlying source of realizations of a uniform $(0, 1)$ random variable. If U is a uniform $(0, 1)$ random variable, and F is the cumulative distribution function of a continuous random variable, then the random variable

$$X = F^{-1}(U)$$

has the cumulative distribution function F. (If the support of X is finite, $F^{-1}(0)$ and $F^{-1}(1)$ are interpreted as the limits of the support.) This same idea, the basis of the so-called inverse CDF method, can also be applied to discrete random variables.

9.7.1 The Multivariate Normal Distribution

If Z has a multivariate normal distribution with the identity as variance-covariance matrix, then for a given positive definite matrix Σ, both

$$Y_1 = \Sigma^{1/2} Z \tag{9.74}$$

and

$$Y_2 = \Sigma_C Z, \tag{9.75}$$

where Σ_C is a Cholesky factor of Σ, have a multivariate normal distribution with variance-covariance matrix Σ (see page 401). The mean of Y_1 is $\Sigma^{1/2}\mu$, where μ is the mean of Z, and the mean of Y_1 is $\Sigma_C \mu$. If Z has 0 mean, then the distributions are identical, that is, $Y_1 \overset{\text{d}}{=} Y_2$.

This leads to a very simple method for generating a multivariate normal random d-vector: generate into a d-vector z d independent $\mathrm{N}_1(0, 1)$. Then form a vector from the desired distribution by the transformation in equation (9.74) or (9.75) together with the addition of a mean vector if necessary.

9.7.2 Random Correlation Matrices

Occasionally we wish to generate random numbers but do not wish to specify the distribution fully. We may want a "random" matrix, but we do not know an exact distribution that we wish to simulate. (There are only a few "standard" distributions of matrices. The Wishart distribution and the Haar distribution (page 222) are the only two common ones. We can also, of course, specify the distributions of the individual elements.)

We may want to simulate random correlation matrices. Although we do not have a specific distribution, we may want to specify some characteristics, such as the eigenvalues. (All of the eigenvalues of a correlation matrix, not just the largest and smallest, determine the condition of data matrices that are realizations of random variables with the given correlation matrix.)

Any nonnegative definite (symmetric) matrix with 1s on the diagonal is a correlation matrix. A correlation matrix is diagonalizable, so if the eigenvalues are $c_1, \ldots, c_d$, we can represent the matrix as

$$V \mathrm{diag}((c_1, \ldots, c_d)) V^{\mathrm{T}}$$

for an orthogonal matrix V. (For a $d \times d$ correlation matrix, we have $\sum c_i = d$; see page 368.) Generating a random correlation matrix with given eigenvalues becomes a problem of generating the random orthogonal eigenvectors and then forming the matrix V from them. (Recall from page 153 that the eigenvectors of a symmetric matrix can be chosen to be orthogonal.) In the following, we let $C = \mathrm{diag}((c_1, \ldots, c_d))$ and begin with $E = I$ (the $d \times d$ identity) and $k = 1$. The method makes use of deflation in step 6 (see page 310). The underlying randomness is that of a normal distribution.

Algorithm 9.2 Random correlation matrices with given eigenvalues

1. Generate a d-vector w of i.i.d. standard normal deviates, form $x = Ew$, and compute $a = x^{\mathrm{T}}(I - C)x$.
2. Generate a d-vector z of i.i.d. standard normal deviates, form $y = Ez$, and compute $b = x^{\mathrm{T}}(I - C)y$, $c = y^{\mathrm{T}}(I - C)y$, and $e^2 = b^2 - ac$.
3. If $e^2 < 0$, then go to step 2.
4. Choose a random sign, $s = -1$ or $s = 1$. Set $r = \dfrac{b + se}{a} x - y$.
5. Choose another random sign, $s = -1$ or $s = 1$, and set $v_k = \dfrac{sr}{(r^{\mathrm{T}} r)^{\frac{1}{2}}}$.
6. Set $E = E - v_k v_k^{\mathrm{T}}$, and set $k = k + 1$.
7. If $k < d$, then go to step 1.
8. Generate a d-vector w of i.i.d. standard normal deviates, form $x = Ew$, and set $v_d = \dfrac{x}{(x^{\mathrm{T}} x)^{\frac{1}{2}}}$.
9. Construct the matrix V using the vectors v_k as its rows. Deliver VCV^{T} as the random correlation matrix. ∎

9.8 Stochastic Processes

Many stochastic processes are modeled by a "state vector" and rules for updating the state vector through a sequence of discrete steps. At time t, the elements of the state vector x_t are values of various characteristics of the system. A model for the stochastic process is a probabilistic prescription for x_{t_a} in terms of x_{t_b}, where $t_a > t_b$; that is, given observations on the state vector prior to some point in time, the model gives probabilities for, or predicts values of, the state vector at later times.

A stochastic process is distinguished in terms of the countability of the space of states, $\mathcal{X}$, and the index of the state (that is, the parameter space, $\mathcal{T}$); either may or may not be countable. If the parameter space is continuous, the process is called a *diffusion process*. If the parameter space is countable, we usually consider it to consist of the nonnegative integers.

If the properties of a stochastic process do not depend on the index, the process is said to be *stationary*. If the properties also do not depend on any initial state, the process is said to be *time homogeneous* or *homogeneous with respect to the parameter space*. (We usually refer to such processes simply as "homogeneous".)

9.8.1 Markov Chains

The *Markov* (or Markovian) *property* in a stochastic process is the condition in which the current state does not depend on any states prior to the immediately previous state; that is, the process is *memoryless*. If the transitions occur at discrete intervals, the Markov property is the condition where the probability distribution of the state at time $t + 1$ depends only on the state at time t.

In what follows, we will briefly consider some Markov processes in which both the set of states is countable and the transitions occur at discrete intervals (discrete times). Such a process is called a *Markov chain*. (Some authors' use of the term "Markov chain" allows the state space to be continuous, and others' allows time to be continuous; here we are not defining the term. We will be concerned with only a subclass of Markov chains, whichever way they are defined. The models for this subclass are easily formulated in terms of vectors and matrices.)

If the state space is countable, it is equivalent to $\mathcal{X} = \{1, 2, \ldots\}$. If X is a random variable from some sample space to $\mathcal{X}$, and

$$\pi_i = \Pr(X = i),$$

then the vector π defines a distribution of X on $\mathcal{X}$. (A vector of nonnegative numbers that sum to 1 is a *distribution*.)

Formally, we define a Markov chain (of random variables) $X_0, X_1, \ldots$ in terms of an initial distribution π and a conditional distribution for X_{t+1} given X_t. Let X_0 have distribution π, and given $X_t = i$, let X_{t+1} have distribution

$(p_{ij}; j \in \mathcal{X})$; that is, p_{ij} is the probability of a transition from state i at time t to state j at time $t+1$. Let

$$P = (p_{ij}).$$

This square matrix is called the *transition matrix* of the chain. It is clear that P is a stochastic matrix (it is nonnegative and the elements in any row sum to 1), and hence $\rho(P) = \|P\|_\infty = 1$, and $(1, 1)$ is an eigenpair of P (see page 379).

If P does not depend on the time (and our notation indicates that we are assuming this), the Markov chain is *stationary*.

The initial distribution π and the transition matrix P characterize the chain, which we sometimes denote as $Markov(\pi, P)$.

If the set of states is countably infinite, the vectors and matrices have infinite order; that is, they have "infinite dimension". (Note that this use of "dimension" is different from our standard definition that is based on linear independence.)

We denote the distribution at time t by $\pi^{(t)}$ and hence often write the initial distribution as $\pi^{(0)}$. A distribution at time t can be expressed in terms of π and P if we extend the definition of (Cayley) matrix multiplication in equation (3.43) in the obvious way to handle any countable number of elements so that PP or P^2 is the matrix defined by

$$(P^2)_{ij} = \sum_{k \in \mathcal{X}} p_{ik} p_{kj}.$$

We see immediately that

$$\pi^{(t)} = (P^t)^{\mathrm{T}} \pi^{(0)}. \tag{9.76}$$

Because of equation (9.76), P^t is often called the *t-step transition matrix*. (The somewhat awkward notation with the transpose results from the historical convention in Markov chain theory of expressing distributions as "row vectors".)

9.8.1.1 Properties of Markov Chains

The transition matrix determines various relationships among the states of a Markov chain. State j is said to be *accessible* from state i if it can be reached from state i in a finite number of steps. This is equivalent to $(P^t)_{ij} > 0$ for some t. If state j is accessible from state i and state i is accessible from state j, states j and i are said to *communicate*. Communication is clearly an equivalence relation. (A binary relation $\sim$ is an *equivalence relation* over some set S if for $x, y, z \in S$, (1) $x \sim x$, (2) $x \sim y \Rightarrow y \sim x$, and (3) $x \sim y \wedge y \sim z \Rightarrow x \sim z$; that is, it is reflexive, symmetric, and transitive.) The set of all states that communicate with each other is an *equivalence class*. States belonging to different equivalence classes do not communicate, although a state in one class may be accessible from a state in a different class.

Identification and analysis of states that communicate can be done by the reduction of the transition matrix in the manner discussed on page 375 and illustrated in equation (8.76), in which by permutations of the rows and columns, square 0 submatrices are formed. If the transition matrix is irreducible, that is, if no such 0 submatrices can be formed, then all states in a Markov chain are in a single equivalence class. In that case the chain is said to be *irreducible*. Irreducible matrices are discussed in Sect. 8.7.3, beginning on page 375, and the implication (8.77) in that section provides a simple characterization of irreducibility. Reducibility of Markov chains is also clearly related to the reducibility in graphs that we discussed in Sect. 8.1.2. (In graphs, the connectivity matrix is similar to the transition matrix in Markov chains.)

If the transition matrix is primitive (that is, it is irreducible and its eigenvalue with maximum modulus has algebraic multiplicity of 1, see page 377), then the Markov chain is said to be *primitive*.

Primitivity and irreducibility are important concepts in analysis of Markov chains because they imply interesting limiting behavior of the chains.

9.8.1.2 Limiting Behavior of Markov Chains

The limiting behavior of the Markov chain is of interest. This of course can be analyzed in terms of $\lim_{t\to\infty} P^t$. Whether or not this limit exists depends on the properties of P. If P is primitive and irreducible, we can make use of the results in Sect. 8.7.3. In particular, because 1 is an eigenvalue and the vector 1 is the eigenvector associated with 1, from equation (8.79), we have

$$\lim_{t\to\infty} P^t = 1\pi_s^{\mathrm{T}}, \tag{9.77}$$

where π_s is the Perron vector of P^{T}.

This also gives us the *limiting distribution* for an irreducible, primitive Markov chain,

$$\lim_{t\to\infty} \pi^{(t)} = \pi_s.$$

The Perron vector has the property $\pi_s = P^{\mathrm{T}}\pi_s$ of course, so this distribution is the *invariant distribution* of the chain. This invariance is a necessary condition for most uses of Markov chains in Monte Carlo methods for generating posterior distributions in Bayesian statistical analysis. These methods are called Markov chain Monte Carlo (MCMC) methods, and are widely used in Bayesian analyses.

There are many other interesting properties of Markov chains that follow from various properties of nonnegative matrices that we discuss in Sect. 8.7, but rather than continuing the discussion here, we refer the interested reader to a text on Markov chains, such as Meyn and Tweedie (2009).

9.8.2 Markovian Population Models

A simple but useful model for population growth measured at discrete points in time, $t, t+1, \ldots$, is constructed as follows. We identify k age groupings for the members of the population; we determine the number of members in each age group at time t, calling this $p^{(t)}$,

$$p^{(t)} = \left(p_1^{(t)}, \ldots, p_k^{(t)}\right);$$

determine the reproductive rate in each age group, calling this α,

$$\alpha = (\alpha_1, \ldots, \alpha_k);$$

and determine the survival rate in each of the first $k-1$ age groups, calling this σ,

$$\sigma = (\sigma_1, \ldots, \sigma_{k-1}).$$

It is assumed that the reproductive rate and the survival rate are constant in time. (There are interesting statistical estimation problems here that are described in standard texts in demography or in animal population models.) The survival rate σ_i is the proportion of members in age group i at time t who survive to age group $i+1$. (It is assumed that the members in the last age group do not survive from time t to time $t+1$.) The total size of the population at time t is $N^{(t)} = 1^{\mathrm{T}} p^{(t)}$. (The use of the capital letter N for a scalar variable is consistent with the notation used in the study of finite populations.)

If the population in each age group is relatively large, then given the sizes of the population age groups at time t, the approximate sizes at time $t+1$ are given by

$$p^{(t+1)} = Ap^{(t)}, \tag{9.78}$$

where A is a Leslie matrix as in equation (8.85),

$$A = \begin{bmatrix} \alpha_1 & \alpha_2 & \cdots & \alpha_{m-1} & \alpha_m \\ \sigma_1 & 0 & \cdots & 0 & 0 \\ 0 & \sigma_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_{m-1} & 0 \end{bmatrix}, \tag{9.79}$$

where $0 \leq \alpha_i$ and $0 \leq \sigma_i \leq 1$.

The Leslie population model can be useful in studying various species of plants or animals. The parameters in the model determine the vitality of the species. For biological realism, at least one α_i and all σ_i must be positive. This model provides a simple approach to the study and simulation of population dynamics. The model depends critically on the eigenvalues of A.

As we have seen (Exercise 8.10), the Leslie matrix has a single unique positive eigenvalue. If that positive eigenvalue is strictly greater in modulus

than any other eigenvalue, then given some initial population size, $p^{(0)}$, the model yields a few damping oscillations and then an exponential growth,

$$p^{(t_0+t)} = p^{(t_0)}e^{rt}, \tag{9.80}$$

where r is the *rate constant*. The vector $p^{(t_0)}$ (or any scalar multiple) is called the *stable age distribution*. (You are asked to show this in Exercise 9.22a.) If 1 is an eigenvalue and all other eigenvalues are strictly less than 1 in modulus, then the population eventually becomes constant; that is, there is a *stable population*. (You are asked to show this in Exercise 9.22b.)

The survival rates and reproductive rates constitute an age-dependent *life table*, which is widely used in studying population growth. The age groups in life tables for higher-order animals are often defined in years, and the parameters often are defined only for females. The first age group is generally age 0, and so $\alpha_1 = 0$. The *net reproductive rate*, r_0, is the average number of (female) offspring born to a given (female) member of the population over the lifetime of that member; that is,

$$r_0 = \sum_{i=2}^{m} \alpha_i \sigma_{i-1}. \tag{9.81}$$

The *average generation time*, T, is given by

$$T = \sum_{i=2}^{m} i\alpha_i \sigma_{i-1}/r_0. \tag{9.82}$$

The net reproductive rate, average generation time, and exponential growth rate constant are related by

$$r = \log(r_0)/T. \tag{9.83}$$

(You are asked to show this in Exercise 9.22c.)

Because the process being modeled is continuous in time and this model is discrete, there are certain averaging approximations that must be made. There are various refinements of this basic model to account for continuous time. There are also refinements to allow for time-varying parameters and for the intervention of exogenous events. Of course, from a statistical perspective, the most interesting questions involve the estimation of the parameters. See Cullen (1985), for example, for further discussions of this modeling problem.

Various starting age distributions can be used in this model to study the population dynamics.

9.8.3 Autoregressive Processes

An interesting type of stochastic process is the p^{th}-order autoregressive time series, defined by the stochastic difference equation

$$x_t = \phi_0 + \phi_1 x_{t-1} + \cdots + \phi_p x_{t-p} + e_t, \tag{9.84}$$

where $\phi_p \neq 0$, the e_t have constant mean of 0 and constant variance of $\sigma^2 > 0$, and any two different e_s and e_t have zero correlation. This model is called an autoregressive model of order p or $AR(p)$.

Comments on the model and notation:

I have implied that e_t is considered to be a random variable, even though it is not written in upper case, and of course, if e_t is a random variable then x_t is also, even though it is not written in upper case either; likewise, I will sometimes consider x_{t-1} and the other x_{t-j} to be random variables.

Stationarity (that is, constancy over time) is an issue. In the simple model above all of the simple parameters are constant; however, unless certain conditions are met, the moments of x_t can grow without bounds. (This is related to the "unit roots" mentioned below. Some authors require that some other conditions be satisfied in an $AR(p)$ model so that moments of the process do not grow without bounds.

Also, many authors omit the ϕ_0 term in the $AR(p)$ model.

The most important properties of this process arise from the autocorrelations, $\mathrm{Cor}(x_s, x_t)$. If these autocorrelations depend on s and t only through $|s-t|$ and if for given $h = |s-t|$ the autocorrelation,

$$\rho_h = \mathrm{Cor}(x_{t+h}, x_t),$$

is constant, the autoregressive process has some simple, but useful properties.

The model (9.84) is a little more complicated than it appears. This is because the specification of x_t is conditional on $x_{t-1}, \ldots, x_{t-p}$. Presumably, also, x_{t-1} is dependent on $x_{t-2}, \ldots, x_{t-p-1}$, and so on. There are no marginal (unconditional) properties of the xs that are specified in the model; that is, we have not specified a starting point.

The stationarity of the e_t (constant mean and constant variance) does not imply that the x_t are stationary. We can make the model more specific by adding a condition of stationarity on the x_t. Let us assume that the x_t have constant and finite means and variances; that is, the $\{x_t\}$ process is (weakly) stationary.

To continue the analysis, consider the $AR(1)$ model. If the x_t have constant means and variances, then

$$\mathrm{E}(x_t) = \frac{\phi_0}{1-\phi_1} \tag{9.85}$$

and

$$\mathrm{V}(x_t) = \frac{\sigma^2}{1-\phi_1^2}. \tag{9.86}$$

The first equation indicates that we cannot have $\phi_1 = 1$ and the second equation makes sense only if $|\phi_1| < 1$. For $AR(p)$ models in general, similar

situations can occur. The denominators in the expressions for the mean and variance involve p-degree polynomials , similar to the first degree polynomials in the denominators of equations (9.85) and (9.86).

We call $f(z) = 1 - \phi_1 z^1 - \cdots - \phi_p z^p$ the *associated polynomial*. If $f(z) = 0$, we have situations similar to a 0 in the denominator of equation (9.86). If a root of $f(z) = 0$ is 1, the expression for a variance is infinite (which we see immediately from equation (9.86) for the $AR(1)$ model). This situation is called a "unit root". If some root is greater than 1, we have an expression for a variance that is negative. Hence, in order for the model to make sense in all respects, all roots of of the associated polynomial must be less than 1 in modulus. (Note some roots can contain imaginary components.)

Although many of the mechanical manipulations in the analysis of the model may be unaffected by unit roots, they have serious implications for the interpretation of the model.

9.8.3.1 Relation of the Autocorrelations to the Autoregressive Coefficients

From the model (9.84) we can see that $\rho_h = 0$ for $h > p$, and $\rho_1, \ldots \rho_p$ are determined by $\phi_1, \ldots \phi_p$ by the relationship

$$R\phi = \rho, \tag{9.87}$$

where ϕ and ρ are the p-vectors of the ϕ_is ($i \neq 0$) and the ρ_is, and R is the Toeplitz matrix (see Sect. 8.8.4)

$$R = \begin{bmatrix} 1 & \rho_1 & \rho_2 & \cdots & \rho_{p-1} \\ \rho_1 & 1 & \rho_1 & \cdots & \rho_{p-2} \\ \rho_2 & \rho_1 & 1 & \cdots & \rho_{p-3} \\ \vdots & & & \ddots & \vdots \\ \rho_{p-1} & \rho_{p-2} & \rho_{p-3} & \cdots & 1 \end{bmatrix}.$$

This system of equations is called the Yule-Walker equations. Notice the relationship of the Yule-Walker equations to the unit root problem mentioned above. For example, for $p = 1$, we have $\phi_1 = \rho_1$. In order for to be a correlation, it must be the case that $|\phi_1| \leq 1$.

For a given set of ρs, possibly estimated from some observations on the time series, Algorithm 9.3 can be used to solve the system (9.87).

Algorithm 9.3 Solution of the Yule-Walker system (9.87)

1. Set $k = 0$; $\phi_1^{(k)} = -\rho_1$; $b^{(k)} = 1$; and $a^{(k)} = -\rho_1$.
2. Set $k = k + 1$.
3. Set $b^{(k)} = \left(1 - \left(a^{(k-1)}\right)^2\right) b^{(k-1)}$.

4. Set $a^{(k)} = -\left(\rho_{k+1} + \sum_{i=1}^{k} \rho_{k+1-i}\phi_1^{(k-1)}\right)/b^{(k)}$.
5. For $i = 1, 2, \ldots, k$
   set $y_i = \phi_i^{(k-1)} + a^{(k)}\phi_{k+1-i}^{(k-1)}$.
6. For $i = 1, 2, \ldots, k$
   set $\phi_i^{(k)} = y_i$.
7. Set $\phi_{k+1}^{(k)} = a^{(k)}$.
8. If $k < p - 1$, go to step 1; otherwise terminate. ∎

This algorithm is $O(p)$ (see Golub and Van Loan 1996).

The Yule-Walker equations arise in many places in the analysis of stochastic processes. Multivariate versions of the equations are used for a vector time series (see Fuller 1995; for example).

Exercises

9.1. Let X be an $n \times m$ matrix with $n > m$ and with entries sampled independently from a continuous distribution (of a real-valued random variable). What is the probability that $X^{\mathrm{T}}X$ is positive definite?

9.2. From equation (9.18), we have $\hat{y}_i = y^{\mathrm{T}}X(X^{\mathrm{T}}X)^{+}x_{i*}$. Show that h_{ii} in equation (9.19) is $\partial\hat{y}_i/\partial y_i$.

9.3. Formally prove from the definition that the sweep operator is its own inverse.

9.4. Consider the regression model

$$y = X\beta + \epsilon \tag{9.88}$$

subject to the linear equality constraints

$$L\beta = c, \tag{9.89}$$

and assume that X is of full column rank.

a) Let λ be the vector of Lagrange multipliers. Form

$$(b^{\mathrm{T}}L^{\mathrm{T}} - c^{\mathrm{T}})\lambda$$

and

$$(y - Xb)^{\mathrm{T}}(y - Xb) + (b^{\mathrm{T}}L^{\mathrm{T}} - c^{\mathrm{T}})\lambda.$$

Now differentiate these two expressions with respect to λ and b, respectively, set the derivatives equal to zero, and solve to obtain

$$\begin{aligned}\widehat{\beta}_C &= (X^{\mathrm{T}}X)^{-1}X^{\mathrm{T}}y - \frac{1}{2}(X^{\mathrm{T}}X)^{-1}L^{\mathrm{T}}\widehat{\lambda}_C \\ &= \widehat{\beta} - \frac{1}{2}(X^{\mathrm{T}}X)^{-1}L^{\mathrm{T}}\widehat{\lambda}_C\end{aligned}$$

and

$$\widehat{\lambda}_C = -2(L(X^{\mathrm{T}}X)^{-1}L^{\mathrm{T}})^{-1}(c - L\widehat{\beta}).$$

Now combine and simplify these expressions to obtain expression (9.29) (on page 416).

b) Prove that the stationary point obtained in Exercise 9.4a actually minimizes the residual sum of squares subject to the equality constraints.
Hint: First express the residual sum of squares as

$$(y - X\widehat{\beta})^{\mathrm{T}}(y - X\widehat{\beta}) + (\widehat{\beta} - b)^{\mathrm{T}}X^{\mathrm{T}}X(\widehat{\beta} - b),$$

and show that is equal to

$$(y-X\widehat{\beta})^{\mathrm{T}}(y-X\widehat{\beta})+(\widehat{\beta}-\widehat{\beta}_C)^{\mathrm{T}}X^{\mathrm{T}}X(\widehat{\beta}-\widehat{\beta}_C)+(\widehat{\beta}_C-b)^{\mathrm{T}}X^{\mathrm{T}}X(\widehat{\beta}_C-b),$$

which is minimized when $b = \widehat{\beta}_C$.

c) Show that sweep operations applied to the matrix (9.30) on page 416 yield the restricted least squares estimate in the (1,2) block.

d) For the weighting matrix W, derive the expression, analogous to equation (9.29), for the generalized or weighted least squares estimator for β in equation (9.88) subject to the equality constraints (9.89).

9.5. Derive a formula similar to equation (9.33) to update $\widehat{\beta}$ due to the deletion of the i^{th} observation.

9.6. When data are used to fit a model such as $y = X\beta + \epsilon$, a large leverage of an observation is generally undesirable. If an observation with large leverage just happens not to fit the "true" model well, it will cause $\widehat{\beta}$ to be farther from β than a similar observation with smaller leverage.

a) Use artificial data to study influence. There are two main aspects to consider in choosing the data: the pattern of X and the values of the residuals in ϵ. The true values of β are not too important, so β can be chosen as 1. Use 20 observations. First, use just one independent variable ($y_i = \beta_0 + \beta_1 x_i + \epsilon_i$). Generate 20 x_is more or less equally spaced between 0 and 10, generate 20 ϵ_is, and form the corresponding y_is. Fit the model, and plot the data and the model. Now, set $x_{20} = 20$, set ϵ_{20} to various values, form the y_i's and fit the model for each value. Notice the influence of x_{20}.
Now, do similar studies with three independent variables. (Do not plot the data, but perform the computations and observe the effect.) Carefully write up a clear description of your study with tables and plots.

b) Heuristically, the leverage of a point arises from the distance from the point to a fulcrum. In the case of a linear regression model, the measure of the distance of observation i is

$$\Delta(x_i, X1/n) = \|x_i, X1/n\|.$$

(This is not the same quantity from the hat matrix that is defined as the leverage on page 410, but it should be clear that the influence of a point for which $\Delta(x_i, X1/n)$ is large is greater than that of a point for which the quantity is small.) It may be possible to overcome some of the undesirable effects of differential leverage by using weighted least squares to fit the model. The weight w_i would be a decreasing function of $\Delta(x_i, X1/n)$.
Now, using datasets similar to those used in the previous part of this exercise, study the use of various weighting schemes to control the influence. Weight functions that may be interesting to try include

$$w_i = e^{-\Delta(x_i, X1/n)}$$

and

$$w_i = \max(w_{\max}, \|\Delta(x_i, X1/n)\|^{-p})$$

for some $w_{\max}$ and some $p > 0$. (Use your imagination!)
Carefully write up a clear description of your study with tables and plots.

c) Now repeat Exercise 9.6b except use a decreasing function of the leverage, h_{ii} from the hat matrix in equation (9.18) instead of the function $\Delta(x_i, X1/n)$.
Carefully write up a clear description of this study, and compare it with the results from Exercise 9.6b.

9.7. By differentiating expression (9.38), derive the normal equations (9.39) for the multivariate linear model.

9.8. Formally prove the relationship expressed in equation (9.71) on page 442.
Hint: Use equation (9.70) twice.

9.9. On page 211, we used Lagrange multipliers to determine the normalized vector x that maximized $x^{\mathrm{T}}Ax$. If A is S_X, this is the first principal component. We also know the principal components from the spectral decomposition. We could also find them by sequential solutions of Lagrangians. After finding the first principal component, we would seek the linear combination z such that $X_{\mathrm{c}}z$ has maximum variance among all normalized z that are orthogonal to the space spanned by the first principal component; that is, that are $X_{\mathrm{c}}^{\mathrm{T}}X_{\mathrm{c}}$-*conjugate* to the first principal component (see equation (3.93) on page 94). If V_1 is the matrix whose columns are the eigenvectors associated with the largest eigenvector, this is equivalent to finding z so as to maximize $z^{\mathrm{T}}Sz$ subject to $V_1^{\mathrm{T}}z = 0$. Using the method of Lagrange multipliers as in equation (4.54), we form the Lagrangian corresponding to equation (4.56) as

$$z^{\mathrm{T}}Sz - \lambda(z^{\mathrm{T}}z - 1) - \phi V_1^{\mathrm{T}}z,$$

where λ is the Lagrange multiplier associated with the normalization requirement $z^{\mathrm{T}}z = 1$, and ϕ is the Lagrange multiplier associated with the orthogonality requirement. Solve this for the second principal component, and show that it is the same as the eigenvector corresponding to the second-largest eigenvalue.

9.10. Obtain the "Longley data". (It is a dataset in R, and it is also available from `statlib`.) Each observation is for a year from 1947 to 1962 and consists of the number of people employed, five other economic variables, and the year itself. Longley (1967) fitted the number of people employed to a linear combination of the other variables, including the year.

a) Use a regression program to obtain the fit.

b) Now consider the year variable. The other variables are measured (estimated) at various times of the year, so replace the year variable with a "midyear" variable (i.e., add $\frac{1}{2}$ to each year). Redo the regression. How do your estimates compare?

c) Compute the L_2 condition number of the matrix of independent variables. Now add a ridge regression diagonal matrix, as in the matrix (9.90), and compute the condition number of the resulting matrix. How do the two condition numbers compare?

9.11. Consider the least squares regression estimator (9.15) for full rank $n \times m$ matrix X $(n > m)$:

$$\widehat{\beta} = (X^{\mathrm{T}}X)^{-1}X^{\mathrm{T}}y.$$

a) Compare this with the ridge estimator

$$\widehat{\beta}_{R(d)} = (X^{\mathrm{T}}X + dI_m)^{-1}X^{\mathrm{T}}y$$

for $d \geq 0$. Show that

$$\|\widehat{\beta}_{R(d)}\| \leq \|\widehat{\beta}\|.$$

b) Show that $\widehat{\beta}_{R(d)}$ is the least squares solution to the regression model similar to $y = X\beta + \epsilon$ except with some additional artificial data; that is, y is replaced with

$$\begin{pmatrix} y \\ 0 \end{pmatrix},$$

where 0 is an m-vector of 0s, and X is replaced with

$$\begin{bmatrix} X \\ dI_m \end{bmatrix}. \tag{9.90}$$

Now explain why $\widehat{\beta}_{R(d)}$ is shorter than $\widehat{\beta}$.

9.12. Use the Schur decomposition (equation (3.190), page 122) of the inverse of $(X^{\mathrm{T}}X)$ to prove equation (9.53).

9.13. Given the matrix

$$A = \begin{bmatrix} 2\ 1\ 3 \\ 1\ 2\ 3 \\ 1\ 1\ 1 \\ 1\ 0\ 1 \end{bmatrix},$$

assume the random 3×2 matrix X is such that

$$\text{vec}(X - A)$$

has a $\text{N}(0, V)$ distribution, where V is block diagonal with the matrix

$$\begin{bmatrix} 2\ 1\ 1\ 1 \\ 1\ 2\ 1\ 1 \\ 1\ 1\ 2\ 1 \\ 1\ 1\ 1\ 2 \end{bmatrix}$$

along the diagonal. Generate ten realizations of X matrices, and use them to test that the rank of A is 2. Use the test statistic (9.64) on page 437.

9.14. Construct a 9×2 matrix X with some missing values, such that S_X computed using all available data for the covariance or correlation matrix is not nonnegative definite.

9.15. Consider an $m \times m$, symmetric nonsingular matrix, R, with 1s on the diagonal and with all off-diagonal elements less than 1 in absolute value. If this matrix is positive definite, it is a correlation matrix. Suppose, however, that some of the eigenvalues are negative. Iman and Davenport (1982) describe a method of adjusting the matrix to a "near-by" matrix that is positive definite. (See Ronald L. Iman and James M. Davenport, 1982, *An Iterative Algorithm to Produce a Positive Definite Correlation Matrix from an "Approximate Correlation Matrix"*, Sandia Report SAND81-1376, Sandia National Laboratories, Albuquerque, New Mexico.) For their method, they assumed the eigenvalues are unique, but this is not necessary in the algorithm.
Before beginning the algorithm, choose a small positive quantity, ϵ, to use in the adjustments, set $k = 0$, and set $R^{(k)} = R$.

1. Compute the eigenvalues of $R^{(k)}$,

$$c_1 \geq c_2 \geq \ldots \geq c_m,$$

and let p be the number of eigenvalues that are negative. If $p = 0$, stop. Otherwise, set

$$c_i^* = \begin{cases} \epsilon & \text{if } c_i < \epsilon \\ c_i & \text{otherwise} \end{cases} \quad \text{for } i = p_1, \ldots, m - p, \tag{9.91}$$

where $p_1 = \max(1, m - 2p)$.

2. Let
$$\sum_i c_i v_i v_i^{\mathrm{T}}$$
be the spectral decomposition of R (equation (3.256), page 154), and form the matrix R^*:
$$R^* = \sum_{i=1}^{p_1} c_i v_i v_i^{\mathrm{T}} + \sum_{i=p_1+1}^{m-p} c_i^* v_i v_i^{\mathrm{T}} + \sum_{i=m-p+1}^{m} \epsilon v_i v_i^{\mathrm{T}}.$$
3. Form $R^{(k)}$ from R^* by setting all diagonal elements to 1.
4. Set $k = k + 1$, and go to step 1. (The algorithm iterates on k until $p = 0$.)

Write a program to implement this adjustment algorithm. Write your program to accept any size matrix and a user-chosen value for ϵ. Test your program on the correlation matrix from Exercise 9.14.

9.16. Consider some variations of the method in Exercise 9.15. For example, do not make the adjustments as in equation (9.91), or make different ones. Consider different adjustments of R^*; for example, adjust any off-diagonal elements that are greater than 1 in absolute value.
Compare the performance of the variations.

9.17. Investigate the convergence of the method in Exercise 9.15. Note that there are several ways the method could converge.

9.18. Suppose the method in Exercise 9.15 converges to a positive definite matrix $R^{(n)}$. Prove that all off-diagonal elements of $R^{(n)}$ are less than 1 in absolute value. (This is true for any positive definite matrix with 1s on the diagonal.)

9.19. Shrinkage adjustments of approximate correlation matrices.
a) Write a program to implement the linear shrinkage adjustment of equation (9.66). Test your program on the correlation matrix from Exercise 9.14.
b) Write a program to implement the nonlinear shrinkage adjustment of equation (9.67). Let $\delta = 0.05$ and
$$f(x) = \tanh(x).$$
Test your program on the correlation matrix from Exercise 9.14.
c) Write a program to implement the scaling adjustment of equation (9.68). Recall that this method applies to an approximate correlation matrix that is a pseudo-correlation matrix. Test your program on the correlation matrix from Exercise 9.14.

9.20. Show that the matrices generated in Algorithm 9.2 are correlation matrices. (They are clearly nonnegative definite, but how do we know that they have 1s on the diagonal?)

9.21. Consider a two-state Markov chain with transition matrix

$$P = \begin{bmatrix} 1-\alpha & \alpha \\ \beta & 1-\beta \end{bmatrix}$$

for $0 < \alpha < 1$ and $0 < \beta < 1$. Does an invariant distribution exist, and if so what is it?

9.22. Recall from Exercise 8.10 that a Leslie matrix has a single unique positive eigenvalue.

a) What are the conditions on a Leslie matrix A that allow a stable age distribution? Prove your assertion.
Hint: Review the development of the power method in equations (7.9) and (7.10).

b) What are the conditions on a Leslie matrix A that allow a stable population, that is, for some x_t, $x_{t+1} = x_t$?

c) Derive equation (9.83). (Recall that there are approximations that result from the use of a discrete model of a continuous process.)

9.23. Derive equations (9.85) and (9.86) under the stationarity assumptions for the model (9.84).

9.24. Derive the Yule-Walker equations (9.87) for the model (9.84).

Part III

Numerical Methods and Software

10

Numerical Methods

The computer is a tool for storage, manipulation, and presentation of data. The data may be numbers, text, or images, but no matter what the data are, they must be coded into a sequence of 0s and 1s because that is what the computer stores. For each type of data, there are several ways of coding. For any unique coding scheme, the primary considerations are efficiency in storage, retrieval, and computations. Each of these considerations may depend on the computing system to be used. Another important consideration is coding that can be shared or transported to other systems.

How much a computer user needs to know about the way the computer works depends on the complexity of the use and the extent to which the necessary operations of the computer have been encapsulated in software that is oriented toward the specific application. This chapter covers many of the basics of how digital computers represent data and perform operations on the data. Although some of the specific details we discuss will not be important for a computational scientist or for someone doing statistical computing, the consequences of those details are important, and the serious computer user must be at least vaguely aware of the consequences.

There are some interesting facts that should cause anyone who programs a computer to take care:

- adding a positive number to a given positive number may not change that positive number;
- performing two multiplications in one order may yield a different result from doing the multiplications in the other order;
- for any given number, there is a "next" number;
- the next number after 1 is farther from 1 than the next number before 1;
- the next number after 1 is closer to 1 than the next number after 2 is to 2;

. . . and so on. . .

J.E. Gentle, *Matrix Algebra*, Springer Texts in Statistics,
DOI 10.1007/978-3-319-64867-5_10

Some of the material in this chapter and the next chapter is covered more extensively in Gentle (2009), Chapters 2 through 6.

Standards

Standards are agreed-upon descriptions of processes, languages, or hardware. They facilitate development of computers and software. Most engineers are very familiar with standards for various types of hardware, such as electronic components, in which case the standard may specify a range for input voltage and current and a range for generated heat. In computing, standards may allow a software developer to know exactly what a piece of Fortran code will do, for example.

Development of a standard often begins with one or more *working groups* of experts, then evolves through a process of *public comment*, and finally is *adopted* by a *standards agency*. Working groups are often formed by professional societies, for example, the International Federation for Information Processing has a number of working groups, including the IFIP Working Group 2.5 on Numerical Software. Standards agencies may be formed by professional societies or by governments. How useful a standard is depends on how widely the standard is followed; hence, larger, stronger, and fewer standards agencies are desirable.

In this and the next two chapters we will mention a number of standards, the most important of which are the IEEE standards for computer representation of numeric values and computations with them, and the ISO (formerly ANSI) standards for computer languages.

Coding Systems

Data of whatever form are represented by groups of 0s and 1s, called *bits* from the words "binary" and "digits". (The word was coined by John Tukey.) For representing simple text (that is, strings of characters each of which has no separate visual distinctions), the bits are usually taken in groups of eight, called *bytes*, and associated with a specific character according to a fixed coding rule. Because of the common association of a byte with a character, those two words are often used synonymously.

The most widely used code for representing characters in bytes is "ASCII" (pronounced "askey", from American Standard Code for Information Interchange). Because the code is so widely used, the phrase "ASCII data" is sometimes used as a synonym for "text data" or "character data". The ASCII code for the character "A", for example, is 01000001; for "a" it is 01100001; and for "5" it is 00110101. Humans can more easily read shorter strings with several different characters than they can longer strings, even if those longer strings consist of only two characters. Bits, therefore, are often grouped into strings of fours; a four-bit string is equivalent to a hexadecimal digit, 1, 2, . . . , 9, A,

B, ... , or F. Thus, the ASCII codes above could be written in hexadecimal notation as 41 ("A"), 61 ("a"), and 35 ("5").

Because the common character sets differ from one language to another (both natural languages and computer languages), there are several modifications of the basic ASCII code set. Also, when there is a need for more different characters than can be represented in a byte (2^8), codes to associate characters with larger groups of bits are necessary. For compatibility with the commonly used ASCII codes using groups of 8 bits, these codes usually are for groups of 16 bits. These codes for "16-bit characters" are useful for representing characters in some Oriental languages, for example. The Unicode Consortium (1990, 1992) has developed a 16-bit standard, called Unicode, that is widely used for representing characters from a variety of languages. For any ASCII character, the Unicode representation uses eight leading 0s and then the same eight bits as the ASCII representation.

A standard scheme for representing data is very important when data are moved from one computer system to another or when researchers at different sites want to share data. Except for some bits that indicate how other bits are to be formed into groups (such as an indicator of the end of a file, or the delimiters of a record within a file), a set of data in ASCII representation is the same on different computer systems. Software systems that process documents either are specific to a given computer system or must have some standard coding to allow portability. The Java system, for example, uses Unicode to represent characters so as to ensure that documents can be shared among widely disparate platforms.

In addition to standard schemes for representing the individual data elements, there are some standard formats for organizing and storing sets of data. These standards specify properties of "data structure".

Types of Data

Bytes that correspond to characters are often concatenated to form *character string data* (or just "strings"). Strings represent text without regard to the appearance of the text if it were to be printed. Thus, a string representing "ABC" does not distinguish between "ABC", "*ABC*", and "**ABC**". The appearance of the printed character must be indicated some other way, perhaps by additional bit strings designating a font.

The appearance of characters or other visual entities such as graphs or pictures is often represented more directly as a "bitmap". Images on a display medium such as paper or a computer screen consist of an arrangement of small dots, possibly of various colors. The dots must be coded into a sequence of bits, and there are various coding schemes in use, such as JPEG (for Joint Photographic Experts Group). Image representations of "ABC", "*ABC*", and "**ABC**" would all be different. The computer's internal representation may correspond directly to the dots that are displayed or it may be a formula to generate the dots, but in each case, the data are represented as a set of dots

located with respect to some coordinate system. More dots would be turned on to represent "**ABC**" than to represent "ABC". The location of the dots and the distance between the dots depend on the coordinate system; thus the image can be repositioned or rescaled.

Computers initially were used primarily to process numeric data, and numbers are still the most important type of data in statistical computing. There are important differences between the numerical quantities with which the computer works and the numerical quantities of everyday experience. The fact that numbers in the computer must have a finite representation has very important consequences.

Missing Data

In statistical computing and in any applications of the data sciences, we must deal with the reality of *missing data*. For any of a number of reasons, certain pieces of a dataset to be analyzed may be missing. Exactly how to proceed with an analysis when some of the data elements are missing depends on the nature of the data and the analysis to be performed and which data elements are missing. For example, there are three different ways to compute a sample variance-covariance matrix from a dataset with missing elements, as discussed in Sect. 9.5.6, beginning on page 437.

In this chapter, we are concerned with computer representation of data, so the first issue is what to store in a computer memory location when the value that should go there is missing. We need a method for representing a "not available" (NA) value, and then we need a mechanism for avoiding operations with this NA value. There are various ways of doing this, and different software systems provide different methods. The standard computer number system itself provides for special computer numbers, as we will discuss on page 475.

Data Structures

The user's interactions with the computing system may include data input/output and other transmissions and various manipulations of the data. The organization of data is called the *data structure*, and there are various standard types of data structures that are useful for different types of computations. In many numerical computations, such as those involving vectors and matrices, the data structures are usually very simple and correspond closely to the ordinary mathematical representation of the entities. In some cases, however, the data may be organized so as to optimize for sparsity or in order to take advantage of some aspect of the computer's architecture, such as interleaved memory banks. In large-scale computational problems, the data may be stored in separate files that reside on separate computing systems.

For more general databases, commercial software vendors have defined data structures and developed data management systems to support accessing and updating the datasets. Some of the details of those systems may be proprietary.

There are also some open systems that are widely used in science applications. One is the Common Data Format (CDF), developed by the NASA's National Space Science Data Center, and date from the 1980s. It is essentially a programming interface for storage and manipulation of multidimensional data. A related system, NetCDF, is built on CDF. There are R packages (`ncdf.tools` and `ncdf4`) to read and write NetCDF files. NetCDF is the standard data structure for some software systems for spatial statistics.

Another standard structure is the Hierarchical Data Format (HDF), developed by the National Center for Supercomputing Applications. The current version of the Hierarchical Data Format, HDF5, which is built on NetCDF, is supported in many software systems including R and Matlab, as well as Fortran, C, Java, and Python.

Both CDF and HDF standards allow a variety of types and structures of data; the standardization is in the descriptions that accompany the datasets.

Computer Architectures and File Systems

The user interacts with the computing system at one level through an *operating system*, such as Linux, Microsoft Windows, or Apple iOS, and at different levels through applications software or other types of software systems. The general organizational structure or *architecture* of the computing system ultimately determines how the user interacts with the computing system. We call the way that the user interacts with the computing system the *programming model*, and it generally consists of multiple simpler programming models. The simplest and probably most common programming model applies to a single user on a single computing system running a single process. In this model, the computing system appears to the user as a single entity, although the processing unit itself make consist of multiple processors (or "cores"). Multiple processors allow for various types of *parallel processing*. Even if the computing system actually consists of several separate "computers", the same model can apply. If the computing system allows for multiple computations to occur simultaneously or when more than one computer comprises the system, a programming model that allows control of the separate processors may be more appropriate. This is especially true if there are different computers that are linked to make up the computer system. This kind of setup is called *distributed computing*.

How data input and output are performed depend on a *file system*. A single computer may have a simple file system that the user is often not even aware of.

More interesting computing applications may involve a distributed environment across many computers or clusters of computers and many program-

ming models. Instead of an ad hoc top-level programming model, it is desirable to have a model that scales from a single system to many machines, each with its own local computing and storage resources. The challenge is to access each resource seamlessly. MapReduce is an efficient method for doing this that was developed by Google, Inc. (now Alphabet, Inc.). Hadoop is an open-source system based on MapReduce methods that provides a scalable programming model. The basic feature of Hadoop is a file system, called the Hadoop Distributed File System or HDFS. (We will discuss MapReduce on page 515 and again briefly on page 533.)

10.1 Digital Representation of Numeric Data

For representing a number in a finite number of digits or bits, the two most relevant things are the magnitude of the number and the precision with which the number is to be represented. Whenever a set of numbers are to be used in the same context, we must find a method of representing the numbers that will accommodate their full range and will carry enough precision for all of the numbers in the set.

Another important aspect in the choice of a method to represent data is the way data are communicated within a computer and between the computer and peripheral components such as data storage units. Data are usually treated as a fixed-length sequence of bits. The basic grouping of bits in a computer is sometimes called a "word" or a "storage unit". The length of words or storage units commonly used in computers is 64 bits, although some systems use 32 bits or other lengths. (In the following examples, I will use 32 bits for simplicity.)

It is desirable to have different kinds of representations for different sets of numbers, even on the same computer. Like the ASCII standard for characters, however, there are some standards for representation of, and operations on, numeric data. The Institute of Electrical and Electronics Engineers (IEEE) and, subsequently, the International Electrotechnical Commission (IEC) have been active in promulgating these standards, and the standards themselves are designated by an IEEE number and/or an IEC number.

The two mathematical models that are often used for numeric data are the ring of integers, $\mathbb{Z}$, and the field of reals, $\mathbb{R}$. We use two computer models, $\mathbb{I}$ and $\mathbb{F}$, to simulate these mathematical entities. (Neither $\mathbb{I}$ nor $\mathbb{F}$ is a simple mathematical construct such as a ring or field.)

10.1.1 The Fixed-Point Number System

Because an important set of numbers is a finite set of reasonably-sized integers, efficient schemes for representing these special numbers are available in most computing systems. The scheme is usually some form of a base 2 representation and may use one storage unit (this is most common), two storage

units, or one half of a storage unit. For example, if a storage unit consists of 32 bits and one storage unit is used to represent an integer, the integer 5 may be represented in binary notation using the low-order bits, as shown in Fig. 10.1.

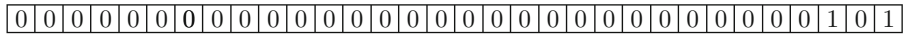


Figure 10.1. The value 5 in a binary representation

The sequence of bits in Fig. 10.1 represents the value 5, using one storage unit. The character "5" is represented in the ASCII code shown previously, 00110101.

If the set of integers includes the negative numbers also, some way of indicating the sign must be available. The first bit in the bit sequence (usually one storage unit) representing an integer is usually used to indicate the sign; if it is 0, a positive number is represented; if it is 1, a negative number is represented. In a common method for representing negative integers, called "twos-complement representation", the sign bit is set to 1 and the remaining bits are set to their opposite values (0 for 1; 1 for 0), and then 1 is added to the result. If the bits for 5 are ...00101, the bits for -5 would be ...11010 + 1, or ...11011. If there are k bits in a storage unit (and one storage unit is used to represent a single integer), the integers from 0 through $2^{k-1} - 1$ would be represented in ordinary binary notation using $k-1$ bits. An integer i in the interval $[-2^{k-1}, -1]$ would be represented by the same bit pattern by which the nonnegative integer $2^{k-1} + i$ is represented, except the sign bit would be 1.

The sequence of bits in Fig. 10.2 represents the value -5 using twos-complement notation in 32 bits, with the leftmost bit being the sign bit and the rightmost bit being the least significant bit; that is, the 1 position. The ASCII code for "-5" consists of the codes for "$-$" and "5"; that is, 00101101 00110101.

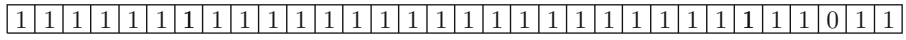


Figure 10.2. The value -5 in a twos-complement representation

The special representations for numeric data are usually chosen so as to facilitate manipulation of data. The twos-complement representation makes arithmetic operations particularly simple. It is easy to see that the largest integer that can be represented in the twos-complement form is $2^{k-1} - 1$ and that the smallest integer is -2^{k-1}.

A representation scheme such as that described above is called *fixed-point* representation or *integer* representation, and the set of such numbers is denoted by $\mathbb{I}$. The notation $\mathbb{I}$ is also used to denote the system built on this set. This system is similar in some ways to a ring, which is what the integers $\mathbb{Z}$ are.

10.1.1.1 Software Representation and Big Integers

The description above relates directly to how the bits in computer memory are stored in a fixed-point representation. The number of bits used and the method of representing negative numbers are two aspects that may vary from one computer to another. Various software systems provide other options, such as "long", "short", "signed", and "unsigned", so that even within a single computer system, the number of bits used in fixed-point representation may vary. The common number of bits is typically one or two storage units or half of a storage unit.

A different approach implemented in the software is to use as much memory as necessary to represent any integer value, if the memory is available. Such a scheme is called *big integer* representation. The representation is also known as multiple precision, arbitrary precision, or infinite precision integer. GMP, for "GNU Multiple Precision", is a C and C++ library that provides mathematical operations involving big integers, as well as multiple precision floating-point numbers. The Python interpreter seamlessly moves between standard fixed-width integers and big integers depending upon the result of a calculation. The `gmp` package in R supports big integers in the `bigz` class.

We discuss the operations with numbers in the fixed-point system in Sect. 10.2.1.

10.1.2 The Floating-Point Model for Real Numbers

In a fixed-point representation, all bits represent values greater than or equal to 1; the *base point* or *radix point* is at the far right, before the first bit. In a fixed-point representation scheme using k bits, the range of representable numbers is of the order of 2^k, usually from approximately -2^{k-1} to 2^{k-1}. Numbers outside of this range cannot be represented directly in the fixed-point scheme. Likewise, nonintegral numbers cannot be represented. Large numbers and fractional numbers are generally represented in a scheme similar to what is sometimes called "scientific notation" or in a type of logarithmic notation. Because within a fixed number of digits the radix point is not fixed, this scheme is called *floating-point* representation, and the set of such numbers is denoted by $\mathbb{F}$. The notation $\mathbb{F}$ is also used to denote the system built on this set.

In a misplaced analogy to the real numbers, a floating-point number is also called "real". Both computer "integers", $\mathbb{I}$, and "reals", $\mathbb{F}$, represent useful subsets of the corresponding mathematical entities, $\mathbb{Z}$ and $\mathbb{R}$, but while the computer numbers called "integers" do constitute a fairly simple subset of the integers, the computer numbers called "real" do not correspond to the real numbers in a natural way. In particular, the floating-point numbers do not occur uniformly over the real number line.

Within the allowable range, a mathematical integer is exactly represented by a computer fixed-point number, but a given real number, even a rational, of any size may or may not have an exact representation by a floating-point number. This is the familiar situation where fractions such as $\frac{1}{3}$ have no finite representation in base 10. The simple rule, of course, is that the number must be a rational number whose denominator in reduced form factors into only primes that appear in the factorization of the base. In base 10, for example, only rational numbers whose factored denominators contain only 2s and 5s have an exact, finite representation; and in base 2, only rational numbers whose factored denominators contain only 2s have an exact, finite representation.

For a given real number x, we will occasionally use the notation

$$[x]_c$$

to indicate the floating-point number used to approximate x, and we will refer to the exact value of a floating-point number as a *computer number*. We will also use the phrase "computer number" to refer to the value of a computer fixed-point number. It is important to understand that the sets of computer numbers $\mathbb{I}$ and $\mathbb{F}$ are finite. The set of fixed-point numbers $\mathbb{I}$ is a proper subset of $\mathbb{Z}$. The set of floating-point numbers is almost a proper subset of $\mathbb{R}$, but it is not a subset because it contains some numbers not in $\mathbb{R}$; see the special floating-point numbers discussed on page 475.

Our main task in using computers, of course, is not to evaluate functions of the set of computer floating-point numbers or the set of computer integers; the main immediate task usually is to perform operations in the field of real (or complex) numbers or occasionally in the ring of integers. Doing computations on the computer, then, involves using the sets of computer numbers to simulate the sets of reals or integers.

10.1.2.1 The Parameters of the Floating-Point Representation

The parameters necessary to define a floating-point representation are the *base* or *radix*, the range of the *mantissa* or *significand*, and the range of the *exponent*. Because the number is to be represented in a fixed number of bits, such as one storage unit or word, the ranges of the significand and exponent must be chosen judiciously so as to fit within the number of bits available. If the radix is b and the integer digits d_i are such that $0 \leq d_i < b$, and there are enough bits in the significand to represent p digits, then a real number is approximated by

$$\pm\, 0.d_1 d_2 \cdots d_p \times b^e, \tag{10.1}$$

where e is an integer. This is the standard model for the floating-point representation. (The d_i are called "digits" from the common use of base 10.)

The number of bits allocated to the exponent e must be sufficient to represent numbers within a reasonable range of magnitudes; that is, so that the

smallest number in magnitude that may be of interest is approximately $b^{e_{\min}}$ and the largest number of interest is approximately $b^{e_{\max}}$, where $e_{\min}$ and $e_{\max}$ are, respectively, the smallest and the largest allowable values of the exponent. Because $e_{\min}$ is likely negative and $e_{\max}$ is positive, the exponent requires a sign. In practice, most computer systems handle the sign of the exponent by defining a *bias* and then subtracting the bias from the value of the exponent evaluated without regard to a sign.

The parameters b, p, and $e_{\min}$ and $e_{\max}$ are so fundamental to the operations of the computer that on most computers they are fixed, except for a choice of two or three values for p and maybe two choices for the range of e.

In order to ensure a unique representation for all numbers (except 0), most floating-point systems require that the leading digit in the significand be nonzero unless the magnitude is less than $b^{e_{\min}}$. A number with a nonzero leading digit in the significand is said to be *normalized*.

The most common value of the base b is 2, although 16 and even 10 are sometimes used. If the base is 2, in a normalized representation, the first digit in the significand is always 1; therefore, it is not necessary to fill that bit position, and so we effectively have an extra bit in the significand. The leading bit, which is not represented, is called a "hidden bit". This requires a special representation for the number 0, however.

In a typical computer using a base of 2 and 64 bits to represent one floating-point number, 1 bit may be designated as the sign bit, 52 bits may be allocated to the significand, and 11 bits allocated to the exponent. The arrangement of these bits is somewhat arbitrary, and of course the physical arrangement on some kind of storage medium would be different from the "logical" arrangement. A common logical arrangement assigns the first bit as the sign bit, the next 11 bits as the exponent, and the last 52 bits as the significand. (Computer engineers sometimes label these bits as $0, 1, \ldots$, and then get confused as to which is the i^{th} bit. When we say "first", we mean "first", whether an engineer calls it the "0^{th}" or the "1^{st}".) The range of exponents for the base of 2 in this typical computer would be 2,048. If this range is split evenly between positive and negative values, the range of orders of magnitude of representable numbers would be from -308 to 308. The bits allocated to the significand would provide roughly 16 decimal places of precision.

Figure 10.3 shows the bit pattern to represent the number 5, using $b = 2$, $p = 24$, $e_{\min} = -126$, and a bias of 127, in a word of 32 bits. The first bit on the left is the sign bit, the next 8 bits represent the exponent, 129, in ordinary base 2 with a bias, and the remaining 23 bits represent the significand beyond the leading bit, known to be 1. (The binary point is to the right of the leading bit that is not represented.) The value is therefore $+1.01 \times 2^2$ in binary notation.

While in fixed-point twos-complement representations there are considerable differences between the representation of a given integer and the negative of that integer (see Figs. 10.1 and 10.2), the only difference between the floating-point representation of a number and its additive inverse is usually

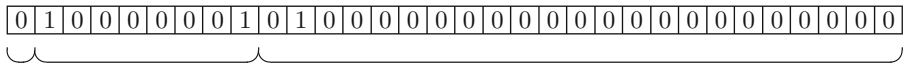


Figure 10.3. The value 5 in a floating-point representation

just in one bit. In the example of Fig. 10.3, only the first bit would be changed to represent the number -5.

As mentioned above, the set of floating-point numbers is not uniformly distributed over the ordered set of the reals. There are the same number of floating-point numbers in the interval $[b^i, b^{i+1}]$ as in the interval $[b^{i+1}, b^{i+2}]$, even though the second interval is b times as long as the first. Figures 10.4, 10.5, 10.6 illustrate this.

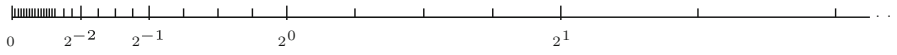


Figure 10.4. The floating-point number line, nonnegative half

Figure 10.5. The floating-point number line, nonpositive half

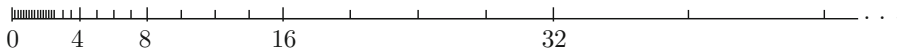


Figure 10.6. The floating-point number line, nonnegative half; another view

The fixed-point numbers, on the other hand, are uniformly distributed over their range, as illustrated in Fig. 10.7.

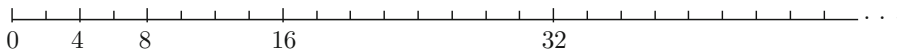


Figure 10.7. The fixed-point number line, nonnegative half

The density of the floating-point numbers is generally greater closer to zero. Notice that if floating-point numbers are all normalized, the spacing between 0 and $b^{e_{\min}}$ is $b^{e_{\min}}$ (that is, there is no floating-point number in that open interval), whereas the spacing between $b^{e_{\min}}$ and $b^{e_{\min}+1}$ is $b^{e_{\min}-p+1}$.

Most systems do not require floating-point numbers less than $b^{e_{\min}}$ in magnitude to be normalized. This means that the spacing between 0 and $b^{e_{\min}}$ can be $b^{e_{\min}-p}$, which is more consistent with the spacing just above $b^{e_{\min}}$. When these nonnormalized numbers are the result of arithmetic operations, the result is called "graceful" or "gradual" underflow.

The spacing between floating-point numbers has some interesting (and, for the novice computer user, surprising!) consequences. For example, if 1 is repeatedly added to x, by the recursion

$$x^{(k+1)} = x^{(k)} + 1,$$

the resulting quantity does not continue to get ever larger. Obviously, it could not increase without bound because of the finite representation. It does not even approach the largest number representable, however! (This is assuming that the parameters of the floating-point representation are reasonable ones.) In fact, if x is initially smaller in absolute value than $b^{e_{\max}-p}$ (approximately), the recursion

$$x^{(k+1)} = x^{(k)} + c$$

will converge to a stationary point for any value of c smaller in absolute value than $b^{e_{\max}-p}$.

The way the arithmetic is performed would determine these values precisely; as we shall see below, arithmetic operations may utilize more bits than are used in the representation of the individual operands.

The spacings of numbers just smaller than 1 and just larger than 1 are particularly interesting. This is because we can determine the *relative spacing* at any point by knowing the spacing around 1. These spacings at 1 are sometimes called the "machine epsilons", denoted $\epsilon_{\min}$ and $\epsilon_{\max}$ (not to be confused with $e_{\min}$ and $e_{\max}$ defined earlier). It is easy to see from the model for floating-point numbers on page 469 that

$$\epsilon_{\min} = b^{-p}$$

and

$$\epsilon_{\max} = b^{1-p};$$

see Fig. 10.8. The more conservative value, $\epsilon_{\max}$, sometimes called "the machine epsilon", ϵ or ϵ_{mach}, provides an upper bound on the rounding that occurs when a floating-point number is chosen to represent a real number. A floating-point number near 1 can be chosen within $\epsilon_{\max}/2$ of a real number that is near 1. This bound, $\frac{1}{2}b^{1-p}$, is called the *unit roundoff*.

These machine epsilons are also called the "smallest relative spacing" and the "largest relative spacing" because they can be used to determine the relative spacing at the point x (Fig. 10.8).

If x is not zero, the relative spacing at x is approximately

$$\frac{x - (1 - \epsilon_{\min})x}{x}$$

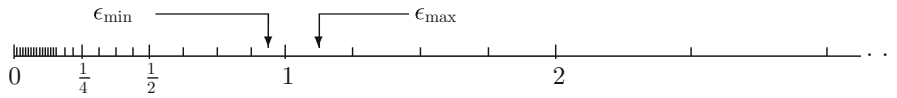


Figure 10.8. Relative spacings at 1: "Machine Epsilons"

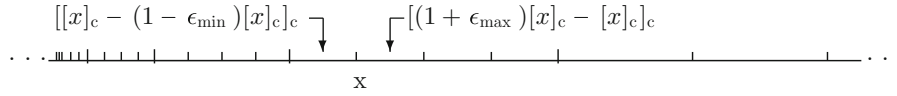


Figure 10.9. Relative spacings

or

$$\frac{(1+\epsilon_{\max})x - x}{x}.$$

Notice that we say "approximately". First of all, we do not even know that x is representable (Fig. 10.9). Although $(1-\epsilon_{\min})$ and $(1+\epsilon_{\max})$ are members of the set of floating-point numbers by definition, that does not guarantee that the product of either of these numbers and $[x]_c$ is also a member of the set of floating-point numbers. However, the quantities $[(1-\epsilon_{\min})[x]_c]_c$ and $[(1+\epsilon_{\max})[x]_c]_c$ are representable (by the definition of $[\cdot]_c$ as a floating point number approximating the quantity within the brackets); and, in fact, they are respectively the next smallest number than $[x]_c$ (if $[x]_c$ is positive, or the next largest number otherwise) and the next largest number than $[x]_c$ (if $[x]_c$ is positive). The spacings at $[x]_c$ therefore are

$$[x]_c - [(1-\epsilon_{\min})[x]_c]_c$$

and

$$[(1+\epsilon_{\max})[x]_c - [x]_c]_c.$$

As an aside, note that this implies it is probable that

$$[(1-\epsilon_{\min})[x]_c]_c = [(1+\epsilon_{\min})[x]_c]_c.$$

In practice, to compare two numbers x and y, we must compare $[x]_c$ and $[y]_c$. We consider x and y different if

$$[|y|]_c < [|x|]_c - [\epsilon_{\min}[|x|]_c]_c$$

or if

$$[|y|]_c > [|x|]_c + [\epsilon_{\max}[|x|]_c]_c.$$

The relative spacing at any point obviously depends on the value represented by the least significant digit in the significand. This digit (or bit) is called the "unit in the last place", or "ulp". The magnitude of an ulp depends

of course on the magnitude of the number being represented. Any real number within the range allowed by the exponent can be approximated within $\frac{1}{2}$ ulp by a floating-point number.

The subsets of numbers that we need in the computer depend on the kinds of numbers that are of interest for the problem at hand. Often, however, the kinds of numbers of interest change dramatically within a given problem. For example, we may begin with integer data in the range from 1 to 50. Most simple operations, such as addition, squaring, and so on, with these data would allow a single paradigm for their representation. The fixed-point representation should work very nicely for such manipulations.

Something as simple as a factorial, however, immediately changes the paradigm. It is unlikely that the fixed-point representation would be able to handle the resulting large numbers. When we significantly change the range of numbers that must be accommodated, another change that occurs is the ability to represent the numbers exactly. If the beginning data are integers between 1 and 50, and no divisions or operations leading to irrational numbers are performed, one storage unit would almost surely be sufficient to represent all values exactly. If factorials are evaluated, however, the results cannot be represented exactly in one storage unit and so must be approximated (even though the results are integers). When data are not integers, it is usually obvious that we must use approximations, but it may also be true for integer data.

10.1.2.2 Standardization of Floating-Point Representation

Over the past several years, standards for floating-point representation have evolved. In the mid 1980s, IEEE promulgated a standard, called the IEEE Standard 754, that specifies the exact layout of the bits in floating-point representations. This standard, which also became the IEC 60559 Standard, was modified slightly in 2008 (IEEE 2008), and is now sometimes referred to as IEEE Standard 754-2008.

The IEEE Standard 754 specifies characteristics for two different precisions, "single" and "double". In both cases, the standard requires that the radix be 2; hence, it is sometimes called the "binary standard". For single precision, p must be 24, $e_{\max}$ must be 127, and $e_{\min}$ must be -126. For double precision, p must be 53, $e_{\max}$ must be 1023, and $e_{\min}$ must be -1022. The standard also defines two additional precisions, "single extended" and "double extended". For each of the extended precisions, the standard sets bounds on the precision and exponent ranges rather than specifying them exactly. The extended precisions have larger exponent ranges and greater precision than the corresponding precision that is not "extended".

The standard also specifies how a number is to be represented when it cannot be represented exactly, that is, how rounding should occur. In practice, this is most relevant in defining the result of a numerical operation. The standard allows for round-to-nearest (with ties going to the smallest in absolute value), round-up, and round-down, but it requires that the default rounding be round-to-nearest.

Most of the computers developed in the past few years comply with the standards, but it is up to the computer manufacturers to conform voluntarily to these standards. We would hope that the marketplace would penalize the manufacturers who do not conform.

Additional information about the IEEE standards for floating-point numbers can be found in Overton (2001).

10.1.2.3 Special Floating-Point Numbers

It is necessary to be able to represent certain special entities, such as infinity, indeterminate (0/0), or simply missing, which do not have ordinary representations in any base-digit system. Although 8 bits are available for the exponent in the single-precision IEEE binary standard, $e_{\max} = 127$ and $e_{\min} = -126$. This means there are two unused possible values for the exponent; likewise, for the double-precision standard, there are two unused possible values for the exponent. These extra possible values for the exponent allow us to represent certain special floating-point numbers. An exponent of $e_{\min} - 1$ allows us to handle 0 and the numbers between 0 and $b^{e_{\min}}$ unambiguously even though there is a hidden bit (see the discussion above about normalization and gradual underflow). The special number 0 is represented with an exponent of $e_{\min} - 1$ and a significand of $00\ldots0$.

An exponent of $e_{\max} + 1$ allows us to represent $\pm\infty$ and indeterminate or missing values. A floating-point number with this exponent and a significand of 0 represents $\pm\infty$ (the sign bit determines the sign, as usual). A floating-point number with this exponent and a nonzero significand represents an indeterminate value such as $\frac{0}{0}$. This value is called "not-a-number", or NaN. There are various ways of representing NaNs. In statistical data processing, a NaN is sometimes used to represent a missing value, but a particular software system may use a specific NaN representation to distinguish different kinds of indeterminate values. (R does this, for example, using a special coding to represent "not available", NA. The concept of missing values can also apply to other things that are not numbers, such as character data.)

Because a NaN is an indeterminate value, if a variable x has a value of NaN, it is neither true that $x = x$ nor that $x \neq x$. Also, because a NaN can be represented in different ways, however, a programmer must be careful in testing for NaNs. Some software systems provide explicit functions for testing for a NaN. The IEEE binary standard recommended that a function `isnan` be provided to test for a NaN. Cody and Coonen (1993) provide C programs for `isnan` and other functions useful in working with floating-point numbers. (R provides `is.nan` for this purpose, but also provides `is.na` for values that have been designated as missing; that is, as not available or NA. Both `is.na(0/0)` and `is.nan(0/0)` are true, but if `x = NA`, then `is.na(x)` is true, but `is.nan(x)` is false.)

We discuss computations with floating-point numbers in Sect. 10.2.2

10.1.3 Language Constructs for Representing Numeric Data

Most general-purpose computer programming languages, such as Fortran, C, and Python, provide constructs for the user to specify the type of representation for numeric quantities. Within the two basic types supported by the computer hardware and described in the preceding sections, the computer language may support additional types of numeric quantities, such as complex numbers.

These specifications of the type are made in declaration statements that are made at the beginning of some section of the program for which they apply, or they are made by "casting" functions or operations.

The difference between fixed-point and floating-point representations has a conceptual basis that may correspond to the problem being addressed. The differences between other kinds of representations often are not because of conceptual differences; rather, they are the results of increasingly irrelevant limitations of the computer. The reasons there are "short" and "long", or "signed" and "unsigned", representations do not arise from the problem the user wishes to solve; the representations are to allow more efficient use of computer resources. The software designer nowadays generally eschews the space-saving constructs that apply to only a relatively small proportion of the data. In some applications, however, the short representations of numeric data still have a place.

10.1.3.1 C

In C, the types of all variables must be specified with a basic declarator, which may be qualified further. For variables containing numeric data, the possible types are shown in Table 10.1.

Table 10.1. Numeric data types in C

Basic type	Basic declarator	Fully qualified declarator
Fixed-point	`int`	`signed short int`
		`unsigned short int`
		`signed long int`
		`unsigned long int`
		`signed long long int`
		`unsigned long long int`
Floating-point	`float`	
	`double`	`double`
		`long double`

Exactly what these types mean is not specified by the language but depends on the specific implementation, which associates each type with some

natural type supported by the specific computer. Common storage for a fixed-point variable of type `short int` uses 16 bits and for type `long int` uses 32 bits. An `unsigned` quantity of either type specifies that no bit is to be used as a sign bit, which effectively doubles the largest representable number. Of course, this is essentially irrelevant for scientific computations, so `unsigned` integers are generally just a nuisance. If neither `short` nor `long` is specified, there is a default interpretation that is implementation-dependent. The default always favors `signed` over `unsigned`. There is a movement toward standardization of the meanings of these types. The American National Standards Institute (ANSI) and its international counterpart, the International Organization for Standardization (ISO), have specified standard definitions of several programming languages. ANSI (1989) is a specification of the C language, which we will refer to as "ANSI C". The ISO has promulgated revisions of the C standard, called "C99" and "C11", corresponding to the years the standards were adopted. These standards are essentially backward-compatible with ANSI C, which remains the most widely used version. C11 has added an additional numeric type, called `long long int`.

Standards for computer languages usually provide minimum specifications, rather than exact meanings of the various data types. ANSI C as well as C99 and C11 require that `short int` use at least 16 bits, that `long int` use at least 32 bits, and that `long int` be at least as long as `int`, which in turn must be least as long as `short int`. The `long double` type may or may not have more precision and a larger range than the `double` type. C11 has added an additional numeric type, called `long long int`, which must use at least 64 bits.

The object-oriented hybrid language built on C, C++ (ANSI 1998), provides the user with the ability also to define operator functions, so that the four simple arithmetic operations can be implemented by the operators "+", "−", "∗", and "/". There is no good way of defining an exponentiation operator, however, because the user-defined operators are limited to extended versions of the operators already defined in the language. (The ISO has also promulgated revisions of the C++ standard, called "C++03", "C++11", "C++14", and "C++17", corresponding to the years the standards were adopted. The revisions add types among other things, but ANSI C++ remains the most widely-used version.)

10.1.3.2 Fortran

Fortran provides three *intrinsic* numeric data types (and also two additional intrinsic types for logical and character quantities), and within each type provides *kinds*. One intrinsic type, `INTEGER`, corresponds to a standard fixed-point entity, and another intrinsic type, `REAL`, corresponds to a standard floating-point entity. The exact form of the storage, including the number of bits, can vary from one platform to another. A third intrinsic type, `COMPLEX`, corresponds to a doubleton of two units of type `REAL`. Arithmetic over numbers

of type COMPLEX, including also numbers of type REAL, conform to the usual rules of arithmetic in the complex field, within the usual limitations of arithmetic in IF. Within each of the five intrinsic types (including LOGICAL and CHARACTER), Fortran provides for various kinds; for example within the real type, there must be at least two kinds available, a default kind and a kind with greater precision (usually "double precision"). The declarator for double precision is of the form

```
REAL(KIND=LONG)
```

There are many other kinds available for the various data types.

In Fortran, variables have a default numeric type that depends on the first letter in the name of the variable. The type can be explicitly declared (and, in fact, should be in careful programming). The signed and unsigned qualifiers of C, which have very little use in scientific computing, are missing in Fortran. Basic types for variables containing numeric data are shown in Table 10.2.

Table 10.2. Numeric data types in Fortran

Basic type	Basic declarator	Default variable name
Fixed-point	INTEGER	Begin with i–n or I–N
Floating-point	REAL	Begin with a–h or o–z or with A–H or O–Z
	REAL(KIND=LONG)	No default, although d or D is sometimes used
Complex	COMPLEX	No default, although c or C is sometimes used
	COMPLEX(KIND=LONG)	

Although the standards organizations have defined these constructs for the Fortran language, just as is the case with C, exactly what these types mean is not specified by the language but depends on the specific implementation, although in most implementations, the number of bytes to be used for storage can be specified through the KIND keyword.

The kind is a qualifier for the basic type; thus a fixed-point number may be an INTEGER of kind 1 or kind 2, for example. The actual value of the qualifier kind may differ from one compiler to another, so the user defines a program parameter to be the kind that is appropriate to the range and precision required for a given variable. Fortran provides the functions SELECTED_INT_KIND and SELECTED_REAL_KIND to do this. Thus, to declare some fixed-point variables that have at least three decimal digits and some more fixed-point variables that have at least eight decimal digits, the user may write the following statements:

```
INTEGER, PARAMETER  :: little = SELECTED_INT_KIND(3)
INTEGER, PARAMETER  :: big    = SELECTED_INT_KIND(8)
INTEGER (little)    :: ismall, jsmall
INTEGER (big)       :: itotal_accounts, igain
```

The variables `little` and `big` would have integer values, chosen by the compiler designer, that could be used in the program to qualify integer types to ensure that range of numbers could be handled. Thus, `ismall` and `jsmall` would be fixed-point numbers that could represent integers between −999 and 999, and `itotal_accounts` and `igain` would be fixed-point numbers that could represent integers between −99,999,999 and 99,999,999. Depending on the basic hardware, the compiler may assign two bytes as kind = `little`, meaning that integers between −32,768 and 32,767 could probably be accommodated by any variable, such as `ismall`, that is declared as `integer (little)`. Likewise, it is probable that the range of variables declared as `integer (big)` could handle numbers in the range −2,147,483,648 and 2,147,483,647. For declaring floating-point numbers, the user can specify a minimum range and precision with the function `SELECTED_REAL_KIND`, which takes two arguments, the number of decimal digits of precision and the exponent of 10 for the range. Thus, the statements

```
INTEGER, PARAMETER  :: real4 = SELECTED_REAL_KIND(6,37)
INTEGER, PARAMETER  :: real8 = SELECTED_REAL_KIND(15,307)
```

would yield designators of floating-point types that would have either six decimals of precision and a range up to 10^{37} or fifteen decimals of precision and a range up to 10^{307}. The statements

```
REAL (real4)    :: x, y
REAL (real8)    :: dx, dy
```

declare `x` and `y` as variables corresponding roughly to `REAL` on most systems and `dx` and `dy` as variables corresponding roughly to `REAL(KIND=LONG` (or `DOUBLE PRECISION`).

If the system cannot provide types matching the requirements specified in `SELECTED_INT_KIND` or `SELECTED_REAL_KIND`, these functions return −1. Because it is not possible to handle such an error situation in the declaration statements, the user should know in advance the available ranges. Fortran provides a number of intrinsic functions, such as `EPSILON`, `RRSPACING`, and `HUGE`, to use in obtaining information about the fixed- and floating-point numbers provided by the system (see Table 10.3).

As with other language standards, the Fortran standard provides minimum specifications, rather than exact meanings of the various data types. Metcalf et al. (2011) summarize the Fortran standard specifications for the various data types.

Fortran also provides a number of intrinsic functions for dealing with bits. These functions are essentially those specified in the MIL-STD-1753 standard of the U.S. Department of Defense. These bit functions, which have been

a part of many Fortran implementations for years, provide for shifting bits within a string, extracting bits, exclusive or inclusive oring of bits, and so on. (See ANSI 1992, or Lemmon and Schafer 2005, for more extensive discussions of the intrinsic functions provided in Fortran.)

10.1.3.3 Python

The standards organizations have not defined a standard for any version of Python. Furthermore, the language has changed in incompatible ways; in particular, from the previous version into the current Python Version 3. Some of the changes affected the constructs for representing numeric data.

Python has supports three different numerical types, an integer type, a floating-point type, and a complex type. The integer type comes in two versions, which are not noticeable to the user. For integral values that can be accommodated in the usual integer representation of the computer, the Python integer type, `int` is essentially a signed long int, and arithmetic operations are performed directly in the arithmetic unit of the computer. (The meaning of the `int` type changed from Python 2 to Python 3.) For integral values outside of the range that can be accommodated in the usual integer representation, Python sets up exact computations on separate parts of the integer values. The limits on the sizes of the integers depend only on the hardware resources available. The floating-point type is the same as the native double precision type of the computer (the same as `double` in C), and arithmetic operations are performed directly in the arithmetic unit of the computer. The complex type is composed of a doubleton of double precision floating-point numbers (the same as `COMPLEX(KIND=LONG)` in Fortran), and Python sets up computations that correspond to the rules of arithmetic on the complex numbers.

10.1.3.4 Determining the Numerical Characteristics of a Particular Computer

The environmental inquiry program MACHAR by Cody (1988) can be used to determine the characteristics of a specific computer's floating-point representation and its arithmetic. Although it may be of interest to examine the source code to see how methods to determine these characteristics work (the program is available in CALGO from `netlib`, see page 619 in the Bibliography), modern systems for numerical computations provide functions to determine the characteristics directly. In R, the results on a given system are stored in the variable `.Machine`. Other R objects that provide information on a computer's characteristics are the variable `.Platform` and the function `capabilities`. Modern Fortran provides a number of functions for inquiring about many of the characteristics of the computer. The standard Fortran intrinsic functions for numeric inquiry are shown in Table 10.3.

Table 10.3. Fortran numeric inquiry intrinsic functions

Generic intrinsic name	Description
`DIGITS(x)`	Number of significant digits
`EPSILON(x)`	"Machine epsilon", $\epsilon_{\max}$
`HUGE(x)`	Largest number
`MAXEXPONENT(x)`	Maximum exponent
`MINEXPONENT(x)`	Minimum exponent
`PRECISION(x)`	Decimal precision
`RADIX(x)`	Base of the model
`RANGE(x)`	Decimal exponent range
`TINY(x)`	Smallest positive number
`RRSPACING(x)`	Reciprocal of the relative spacing of numbers near `x`
`SPACING(x)`	Absolute spacing of numbers near `x`

Value returned is for numbers of the same `KIND` as `x`

Many higher-level languages and application software packages do not give the user a choice of how to represent numeric data. The software system may consistently use a type thought to be appropriate for the kinds of applications addressed. For example, many statistical analysis application packages choose to use a floating-point representation with about 64 bits for all numeric data. Making a choice such as this yields more comparable results across a range of computer platforms on which the software system may be implemented.

Whenever the user chooses the type and precision of variables, it is a good idea to use some convention to name the variable in such a way as to indicate the type and precision. Books or courses on elementary programming suggest using mnemonic names, such as "`time`", for a variable that holds the measure of time. If the variable takes fixed-point values, a better name might be "`itime`". It still has the mnemonic value of "time", but it also helps us to remember that, in the computer, `itime/length` may not be the same thing as `time/xlength`. Although the variables are declared in the program to be of a specific type, the programmer can benefit from a reminder of the type. Even as we "humanize" computing, we must remember that there are details about the computer that matter. (The operator "`/`" is said to be "overloaded": in a general way, it means "divide", but it means different things depending on the contexts of the two expressions above.) Whether a quantity is a member of $\mathbb{I}$ or $\mathbb{F}$ may have major consequences for the computations, and a careful choice of notation can help to remind us of that, even if the notation may look old-fashioned.

Numerical analysts sometimes use the phrase "full precision" to refer to a precision of about sixteen decimal digits and the phrase "half precision" to refer to a precision of about seven decimal digits. These terms are not defined precisely, but they do allow us to speak of the precision in roughly equivalent ways for different computer systems without specifying the precision exactly. Full precision is roughly equivalent to Fortran `REAL(KIND=LONG)` (or `DOUBLE`

PRECISION) on 32-bit computers, and to Fortran REAL on 64-bit machines. Half precision corresponds roughly to Fortran REAL on 32-bit machines. Full and half precision can be handled in a portable way in Fortran. The following statements declare a variable x to be one with full precision:

```
INTEGER, PARAMETER  :: full = SELECTED\_REAL\_KIND(15,307)
REAL(full)          :: x
```

In a construct of this kind, the user can define "full" or "half" as appropriate.

10.1.4 Other Variations in the Representation of Data; Portability of Data

As we have indicated already, computer designers have a great deal of latitude in how they choose to represent data. The ASCII standards of ANSI and ISO have provided a common representation for individual characters. The IEEE Standard 754-2008 referred to previously (IEEE, 2008) brought some standardization to the representation of floating-point data, but do not specify how the available bits are to be allocated among the sign, exponent, and significand.

Because the number of bits used as the basic storage unit has generally increased over time, some computer designers have arranged small groups of bits, such as bytes, together in strange ways to form words. There are two common schemes of organizing bits into bytes and bytes into words. In one scheme, called "big end" or "big endian", the bits are indexed from the "left", or most significant, end of the byte, and bytes are indexed within words and words are indexed within groups of words in the same direction.

In another scheme, called "little end" or "little endian", the bytes are indexed within the word in the opposite direction. Here's a program in Fig. 10.10 that produces difference output on different systems. Figures 10.11 and 10.12 illustrate some of the differences, using the program shown in Fig. 10.10. The R function .Platform provides information on the type of endian of the given machine on which the program is running.

These differences are important only when accessing the individual bits and bytes, when making data type transformations directly, or when moving data from one machine to another without interpreting the data in the process ("binary transfer"). One lesson to be learned from observing such subtle differences in the way the same quantities are treated in different computer systems is that programs should rarely rely on the inner workings of the computer. A program that does will not be *portable*; that is, it will not give the same results on different computer systems. Programs that are not portable may work well on one system, and the developers of the programs may never intend for them to be used anywhere else. As time passes, however, systems change or users change systems. When that happens, the programs

```
CHARACTER a
CHARACTER*4 b
INTEGER i, j
EQUIVALENCE (b,i), (a,j)
PRINT '(10x, a7 , a8)', ' Bits  ', '   Value'
a = 'a'
PRINT '(1X, A10, Z2, 7X, A1)',  'a:        ', a, a
PRINT '(1X, A10, Z8, 1X, I12)', 'j (=a):   ', j, j
b = 'abcd'
PRINT '(1X, A10, Z8, 1X, A4)',  'b:        ', b, b
PRINT '(1X, A10, Z8, 1X, I12)', 'i (=b):   ', i, i
END
```

Figure 10.10. A Fortran program illustrating bit and byte organization

```
            Bits     Value
a:          61       a
j (=a):     00000061           97
b:          61626364 abcd
i (=b):     64636261   1684234849
```

Figure 10.11. Output from a Little Endian System

```
            Bits     Value
a:          61       a
j (=a):     61000000   1627389952
b:          61626364 abcd
i (=b):     61626364   1633837924
```

Figure 10.12. Output from a Big Endian System

that were not portable may cost more than they ever saved by making use of computer-specific features.

The *external data representation*, or XDR, standard format, developed by Sun Microsystems for use in remote procedure calls, is a widely used machine-independent standard for binary data structures.

10.2 Computer Operations on Numeric Data

As we have emphasized above, the numerical quantities represented in the computer are used to simulate or approximate more interesting quantities, namely the real numbers or perhaps the integers. Obviously, because the sets (computer numbers and real numbers) are not the same, we could not define operations on the computer numbers that would yield the same field as the familiar field of the reals. In fact, because of the nonuniform spacing of floating-point numbers, we would suspect that some of the fundamental properties of a field may not hold. Depending on the magnitudes of the quantities

involved, it is possible, for example, that if we compute ab and ac and then $ab + ac$, we may not get the same thing as if we compute $(b + c)$ and then $a(b + c)$. Just as we use the computer quantities to simulate real quantities, we define operations on the computer quantities to simulate the familiar operations on real quantities. Designers of computers attempt to define computer operations so as to correspond closely to operations on real numbers, but we must not lose sight of the fact that the computer uses a different arithmetic system.

The basic operational objective in numerical computing, of course, is that a computer operation, when applied to computer numbers, yield computer numbers that approximate the number that would be yielded by a certain mathematical operation applied to the numbers approximated by the original computer numbers. Just as we introduced the notation

$$[x]_c$$

on page 469 to denote the computer floating-point number approximation to the real number x, we occasionally use the notation

$$[\circ]_c$$

to refer to a computer operation that simulates the mathematical operation $\circ$. Thus,

$$[+]_c$$

represents an operation similar to addition but that yields a result in a set of computer numbers. (We use this notation only where necessary for emphasis, however, because it is somewhat awkward to use it consistently.) The failure of the familiar laws of the field of the reals, such as the distributive law cited above, can be anticipated by noting that

$$[a]_c\ [+]_c\ [b]_c \neq [a + b]_c,$$

or by considering the simple example in which all numbers are rounded to one decimal place and so $\frac{1}{3} + \frac{1}{3} \neq \frac{2}{3}$ (that is, $.3 + .3 \neq .7$).

The three familiar laws of the field of the reals (commutativity of addition and multiplication, associativity of addition and multiplication, and distribution of multiplication over addition) result in the independence of the order in which operations are performed; the failure of these laws implies that the order of the operations may make a difference. When computer operations are performed sequentially, we can usually define and control the sequence fairly easily. If the computer performs operations in parallel, the resulting differences in the orders in which some operations may be performed can occasionally yield unexpected results.

Because the operations are not closed, special notice may need to be taken when the operation would yield a number not in the set. Adding two numbers, for example, may yield a number too large to be represented well by

a computer number, either fixed-point or floating-point. When an operation yields such an anomalous result, an *exception* is said to exist.

The computer operations for the two different types of computer numbers are different, and we discuss them separately.

10.2.1 Fixed-Point Operations

The operations of addition, subtraction, and multiplication for fixed-point numbers are performed in an obvious way that corresponds to the similar operations on the ring of integers. Subtraction is addition of the additive inverse. (In the usual twos-complement representation we described earlier, all fixed-point numbers have additive inverses except -2^{k-1}.) Because there is no multiplicative inverse, however, division is not multiplication by the inverse. The result of division with fixed-point numbers is the result of division with the corresponding real numbers rounded toward zero. This is not considered an exception.

As we indicated above, the set of fixed-point numbers together with addition and multiplication is not the same as the ring of integers, if for no other reason than that the set is finite. Under the ordinary definitions of addition and multiplication, the set is not closed under either operation. The computer operations of addition and multiplication, however, are defined so that the set is closed. These operations occur as if there were additional higher-order bits and the sign bit were interpreted as a regular numeric bit. The result is then whatever would be in the standard number of lower-order bits. If the lost higher-order bits are necessary, the operation is said to *overflow*. If fixed-point overflow occurs, the result is not correct under the usual interpretation of the operation, so an error situation, or an exception, has occurred. Most computer systems allow this error condition to be detected, but most software systems do not take note of the exception. The result, of course, depends on the specific computer architecture. On many systems, aside from the interpretation of the sign bit, the result is essentially the same as would result from a modular reduction. There are some special-purpose algorithms that actually use this modified modular reduction, although such algorithms would not be portable across different computer systems.

10.2.2 Floating-Point Operations

As we have seen, real numbers within the allowable range may or may not have an exact floating-point operation, and the computer operations on the computer numbers may or may not yield numbers that represent exactly the real number that would result from mathematical operations on the numbers. If the true result is r, the best we could hope for would be $[r]_c$. As we have mentioned, however, the computer operation may not be exactly the same as the mathematical operation being simulated, and furthermore, there may be several operations involved in arriving at the result. Hence, we expect some error in the result.

10.2.2.1 Errors

If the computed value is $\tilde{r}$ (for the true value r), we speak of the *absolute error*,

$$|\tilde{r} - r|,$$

and the *relative error*,

$$\frac{|\tilde{r} - r|}{|r|}$$

(so long as $r \neq 0$). An important objective in numerical computation obviously is to ensure that the error in the result is small.

We will discuss error in floating-point computations further in Sect. 10.3.2.

10.2.2.2 Guard Digits and Chained Operations

Ideally, the result of an operation on two floating-point numbers would be the same as if the operation were performed exactly on the two operands (considering them to be exact also) and the result was then rounded. Attempting to do this would be very expensive in both computational time and complexity of the software. If care is not taken, however, the relative error can be very large. Consider, for example, a floating-point number system with $b = 2$ and $p = 4$. Suppose we want to add 8 and -7.5. In the floating-point system, we would be faced with the problem

$$\begin{aligned} 8 &: 1.000 \times 2^3 \\ 7.5 &: 1.111 \times 2^2. \end{aligned}$$

To make the exponents the same, we have

$$\begin{aligned} 8 &: 1.000 \times 2^3 \\ 7.5 &: 0.111 \times 2^3 \end{aligned} \quad \text{or} \quad \begin{aligned} 8 &: 1.000 \times 2^3 \\ 7.5 &: 1.000 \times 2^3. \end{aligned}$$

The subtraction will yield either 0.000_2 or $1.000_2 \times 2^0$, whereas the correct value is $1.000_2 \times 2^{-1}$. Either way, the absolute error is 0.5_{10}, and the relative error is 1. Every bit in the significand is wrong. The magnitude of the error is the same as the magnitude of the result. This is not acceptable. (More generally, we could show that the relative error in a similar computation could be as large as $b - 1$ for any base b.) The solution to this problem is to use one or more *guard digits*. A guard digit is an extra digit in the significand that participates in the arithmetic operation. If one guard digit is used (and this is the most common situation), the operands each have $p + 1$ digits in the significand. In the example above, we would have

$$\begin{aligned} 8 &: 1.0000 \times 2^3 \\ 7.5 &: 0.1111 \times 2^3, \end{aligned}$$

and the result is exact. In general, one guard digit can ensure that the relative error is less than $2\epsilon_{\max}$. The use of guard digits requires that the operands be stored in special storage units. Whenever multiple operations are to be performed together, the operands and intermediate results can all be kept in the special registers to take advantage of the guard digits or even longer storage units. This is called chaining of operations.

10.2.2.3 Addition of Several Numbers

When several numbers x_i are to be summed, it is likely that as the operations proceed serially, the magnitudes of the partial sum and the next summand will be quite different. In such a case, the full precision of the next summand is lost. This is especially true if the numbers are of the same sign. As we mentioned earlier, a computer program to implement serially the algorithm implied by $\sum_{i=1}^{\infty} i$ will converge to some number much smaller than the largest floating-point number.

If the numbers to be summed are not all the same constant (and if they are constant, just use multiplication!), the accuracy of the summation can be increased by first sorting the numbers and summing them in order of increasing magnitude. If the numbers are all of the same sign and have roughly the same magnitude, a pairwise "fan-in" method may yield good accuracy. In the fan-in method, the n numbers to be summed are added two at a time to yield $\lceil n/2 \rceil$ partial sums. The partial sums are then added two at a time, and so on, until all sums are completed. The name "fan-in" comes from the tree diagram of the separate steps of the computations:

$$
\begin{array}{ccccccc}
s_1^{(1)} = x_1 + x_2 & & s_2^{(1)} = x_3 + x_4 & \vdots & s_{2m-1}^{(1)} = x_{4m-3} + x_{4m-2} & & s_{2m}^{(1)} = \\
\searrow & & \swarrow & \vdots & \searrow & & \swarrow \\
s_1^{(2)} = s_1^{(1)} + s_2^{(1)} & & & \vdots & s_m^{(2)} = s_{2m-1}^{(1)} + s_{2m}^{(1)} & & \\
\searrow & & & \vdots & \downarrow & & \\
s_1^{(3)} = s_1^{(2)} + s_2^{(2)} & & & \vdots & \cdots & & \\
\end{array}
$$

It is likely that the numbers to be added will be of roughly the same magnitude at each stage. Remember we are assuming they have the same sign initially; this would be the case, for example, if the summands are squares.

10.2.2.4 Compensated Summation

Another way of summing many numbers of varying magnitudes is due to W. Kahan. It is called *compensated summation*, and for $x_1, \ldots, x_n$, it follows these steps:

$$
\begin{array}{l}
s = x_1 \\
a = 0 \\
\text{for } i = 2, \ldots, n \\
\{ \\
\quad y = x_i - a \\
\quad t = s + y \\
\quad a = (t - s) - y \\
\quad s = t \\
\}.
\end{array}
\tag{10.2}
$$

Much of the work on improving the accuracy of summation was motivated by the problem of computation of L_2 norms, or more generally computation of dot products. The exact dot product (EDP) procedure, described on page 495 addresses this problem. Both compensated summation and EDP implemented in the hardware have been used in some versions of the BLAS. (See Sect. 12.2.1 on page 555 for descriptions of the BLAS.)

In Sect. 10.2.5 we discuss exact computation, and obviously any of the methods of exact computation can also be used to increase the accuracy of summations, and, indeed, in most cases, to make the summations exact.

10.2.2.5 Catastrophic Cancellation

Another kind of error that can result because of the finite precision used for floating-point numbers is *catastrophic cancellation*. This can occur when two rounded values of approximately equal magnitude and opposite signs are added. (If the values are exact, cancellation can also occur, but it is *benign*.) After catastrophic cancellation, the digits left are just the digits that represented the rounding. Suppose $x \approx y$ and that $[x]_c = [y]_c$. The computed result will be zero, whereas the correct (rounded) result is $[x-y]_c$. The relative error is 100%. This error is caused by rounding, but it is different from the "rounding error" discussed above. Although the loss of information arising from the rounding error is the culprit, the rounding would be of little consequence were it not for the cancellation.

To avoid catastrophic cancellation, watch for possible additions of quantities of approximately equal magnitude and opposite signs, and rearrange the computations if possible. Consider the problem of computing the roots of a quadratic polynomial, $ax^2 + bx + c$. In the quadratic formula

$$
x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a},
\tag{10.3}
$$

the square root of the discriminant, $(b^2 - 4ac)$, may be approximately equal to b in magnitude, meaning that one of the roots is close to zero and, in fact, may be computed as zero. The solution is to compute only one of the roots, x_1, by the formula (the "$-$" root if b is positive and the "$+$" root if b is negative) and then compute the other root, x_2 by the relationship $x_1 x_2 = c/a$.

In discussing Householder reflections, we have seen another example where catastrophic cancellation could occur, and how we can avoid it; in particular, in equation (5.8) on page 236.

10.2.2.6 Standards for Floating-Point Operations

The IEEE Binary Standard 754 (IEEE,2008) applies not only to the representation of floating-point numbers but also to certain operations on those numbers. The standard requires correct rounded results for addition, subtraction, multiplication, division, remaindering, and extraction of the square root. It also requires that conversion between fixed-point numbers and floating-point numbers yield correct rounded results.

An *inexact operation* is one for which the result must be rounded. For example, in a multiplication operation, if all p bits of the significand are required to represent both the multiplier and multiplicand, approximately $2p$ bits would be required to represent the product. Because only p are available, however, the result generally must be rounded.

If a floating-point operation does not yield the exact result of the corresponding mathematical operation, then an exception is said to occur. In this sense, inexact operations are exceptions; but that kind of exception is generally ignored.

The IEEE Binary Standard defines how exceptions should be handled. The exceptions are divided into four types: division of a nonzero number by zero, overflow, underflow, and invalid operation.

Division by zero results in a special number if the dividend is nonzero. The result is either ∞ or $-\infty$, and these have special representations, as we have seen.

If an operation on floating-point numbers would result in a number beyond the range of representable floating-point numbers, the exception, called *overflow*, is generally very serious. (Overflow is serious in fixed-point operations also if it is unplanned. Because we have the alternative of using floating-point numbers if the magnitude of the numbers is likely to exceed what is representable in fixed-point numbers, the user is expected to use this alternative. If the magnitude exceeds what is representable in floating-point numbers, however, the user must resort to some indirect means, such as scaling, to solve the problem.)

Underflow occurs whenever the result is too small to be represented as a normalized floating-point number. As we have seen, a nonnormalized representation can be used to allow a gradual underflow.

An *invalid operation* is one for which the result is not defined because of the values of the operands. The invalid operations are addition of ∞ to $-\infty$, multiplication of $\pm\infty$ and 0, 0 divided by 0 or by $\pm\infty$, $\pm\infty$ divided by 0 or by $\pm\infty$, extraction of the square root of a negative number (some systems, such as Fortran, R, and Matlab, have a special type for complex numbers and deal correctly with them), and remaindering any quantity with 0 or remaindering

$\pm\infty$ with any quantity. An invalid operation results in a NaN. The IEEE Binary Standard 754 provides for two classes of NaNs, a "quiet NaN" and a "signaling NaN", for which the processing unit will raise an exception; that is, set a condition that can be queried by a program. (As it turns out, this feature is rarely useful except in debugging programs.)

Conformance to the IEEE Binary Standard 754 does not ensure that the results of multiple floating-point computations will be the same on all computers. The standard does not specify the order of the computations, and differences in the order can change the results. The slight differences are usually unimportant, but Blackford et al. (1997a) describe some examples of problems that occurred when computations were performed in parallel using a heterogeneous network of computers all of which conformed to the IEEE standard. See also Gropp (2005) for further discussion of some of these issues.

10.2.2.7 Operations Involving Special Floating-Point Numbers

Operations involving any of the special floating-point numbers result in values consistent with the meaning of the special value. For example, $\pm\infty + 1 = \pm\infty$, $\pm\infty * 1 = \pm\infty$, and $\pm\infty * (-1) = \mp\infty$.

Any operation involving a single NaN results in a NaN of that same type. In systems that distinguish various types of NaNs, an operation involving different types results in a value consistent with a hierarchy of NaNs. (In such cases, the terminology may differ from that of the IEEE Binary Standard. For example, in R, any NaN is an NA, but an NA is not a NaN. Adding an NaN to an NA results in an NA, consistent with the relationship of not-available to not-a-number.)

10.2.2.8 Comparison of Real Numbers and Floating-Point Numbers

For most applications, the system of floating-point numbers simulates the field of the reals very well. It is important, however, to be aware of some of the differences in the two systems. There is a very obvious useful measure for the reals, namely the Lebesgue measure, based on lengths of open intervals. An approximation of this measure is appropriate for floating-point numbers, even though the set of floating-point numbers within any interval is finite. (Because the set is finite, we might consider basing a measure on a counting measure, but a counting measure does not work well at all. For one thing, the finiteness of the set of floating-point numbers means that there may be a difference in the cardinality of an open interval and a closed interval with the same endpoints. Also, the uneven distribution of floating-point values relative to the reals (Figs. 10.4 and 10.5) means that the cardinalities of two interval-bounded sets with the same interval length may be different.)

Some general differences in the real numbers and the floating-point system are exhibited in Table 10.4. The last four properties in Table 10.4 are properties of a field. The important facts are that $\mathbb{R}$ is an uncountable field and that $\mathbb{F}$ is a more complicated finite mathematical structure.

10.2.3 Language Constructs for Operations on Numeric Data

Most general-purpose computer programming languages provide constructs for operations that correspond to the common operations on scalar numeric data, such as "`+`", "`-`", "`*`" (multiplication), and "`/`". These operators *simulate* the corresponding mathematical operations. As we mentioned on page 484, we will occasionally use notation such as

$$[+]_c$$

to indicate the computer operator. The operators have slightly different meanings depending on the operand objects; that is, the operations are "*overloaded*". Most of these operators are *binary infix* operators, meaning that the operator is written between the two operands.

Most languages provide operations beyond the four basic scalar arithmetic operations, especially for exponentiation, usually specified as "`**`" or "`^`". (In C, exponentiation is handled by a function provided in a standard supplemental library `<math.h>`.) Exactly what the exponentiation operator means may be slightly different. Some versions of Fortran, for example, interpret the operator always to mean

1. take log
2. multiply by power
3. exponentiate

if the base and the power are both floating-point types. This, of course, will not work if the base is negative, even if the power is an integer. Most versions of Fortran will determine at run time if the power is an integer and use repeated multiplication if it is.

C provides some specialized operations, such as the unary prefix and postfix increment "`++`" and decrement "`--`" operators, for adding or subtracting 1. Although these operations can easily be performed directly by standard operators, the operators look cool, and some other languages have adopted them. Related augmented assignment operators, "`+=`", "`-=`", and so on, were also defined in C, and adopted by some other languages.

C also overloads the basic multiplication operator so that it can indicate a change of meaning of a variable in addition to indicating the multiplication of two scalar numbers. A standard library in C (`<signal.h>`) allows easy handling of arithmetic exceptions. With this facility, for example, the user can distinguish a quiet NaN from a signaling NaN.

Table 10.4. Differences in real numbers and floating-point numbers

	$\mathbb{R}$	$\mathbb{F}$
Cardinality:	Uncountable	Finite
Measure:	$\mu((x,y)) = \lvert x-y\rvert$ $\mu((x,y)) = \mu([x,y])$	$\nu((\mathtt{x},\mathtt{y})) = \nu([\mathtt{x},\mathtt{y}]) = \lvert\mathtt{x}-\mathtt{y}\rvert$ $\exists\, \mathtt{x},\mathtt{y},\mathtt{z},\mathtt{w} \ni \lvert\mathtt{x}-\mathtt{y}\rvert = \lvert\mathtt{z}-\mathtt{w}\rvert$, but $\#(\mathtt{x},\mathtt{y}) \neq \#(\mathtt{z},\mathtt{w})$
Continuity:	if $x < y$, $\exists z \ni x < z < y$ and $\mu([x,y]) = \mu((x,y))$	$\mathtt{x} < \mathtt{y}$, but no $\mathtt{z} \ni \mathtt{x} < \mathtt{z} < \mathtt{y}$ and $\#[\mathtt{x},\mathtt{y}] > \#(\mathtt{x},\mathtt{y})$
Convergence	$\sum_{x=1}^{\infty} x$ diverges	$\sum_{\mathtt{x}=1}^{\infty} \mathtt{x}$ converges, if interpreted as $(\cdots((1+2)+3)\cdots)$
Closure:	$x,y \in \mathbb{R} \Rightarrow x+y \in \mathbb{R}$ $x,y \in \mathbb{R} \Rightarrow xy \in \mathbb{R}$	Not closed wrt addition Not closed wrt multiplication (exclusive of infinities)
Operations with an identity, a or a:	$a = 0$, unique $a + x = x$, for any x $x - x = a$, for any x	$\mathtt{a}+\mathtt{x} = \mathtt{b}+\mathtt{x}$, but $\mathtt{b} \neq \mathtt{a}$ $\mathtt{a}+\mathtt{x} = \mathtt{x}$, but $\mathtt{a}+\mathtt{y} \neq \mathtt{y}$ $\mathtt{a}+\mathtt{x} = \mathtt{x}$, but $\mathtt{x}-\mathtt{x} \neq \mathtt{a}$
Associativity:	$x,y,z \in \mathbb{R} \Rightarrow$ $(x+y)+z = x+(y+z)$ $(xy)z = x(yz)$	 Not associative Not associative
Distributivity:	$x,y,z \in \mathbb{R} \Rightarrow$ $x(y+z) = xy+xz$	 Not distributive

Fortran and Python provide the usual five operators (the basic four plus exponentiation) for complex data . As mentioned before, the actual arithmetic

is set up to use the ordinary arithmetic operations in such a way as to conform to the rules of complex arithmetic.

Modern Fortran also provides numeric operators for vectors and matrices. The usual vector/matrix operators are implemented as intrinsic functions or as prefix operators in Fortran.

In addition to the basic arithmetic operators, the common programming languages provide several other types of operators, including relational operators and operators for manipulating structures of data.

10.2.4 Software Methods for Extending the Precision

Software packages have been built to extend the numerical accuracy of computations, possibly to the extent that the computations are exact. Four ways in which this is done are by use of *multiple precision*, *rational fractions*, *interval arithmetic*, and *residue arithmetic*. We discuss the first three of these in this section, and then in the next section briefly discuss residue arithmetic, which may also be implemented at the microcode or hardware level.

10.2.4.1 Multiple Precision

Multiple-precision operations are performed in the software by combining more than one computer-storage unit to represent a single number. For example, to operate on x and y, we may represent x as $a \cdot 10^p + b$ and y as $c \cdot 10^p + d$. The product xy then is formed as $ac \cdot 10^{2p} + (ad + bc) \cdot 10^p + bd$. The representation is chosen so that any of the coefficients of the scaling factors (in this case powers of 10) can be represented to within the desired accuracy.

Multiple precision is different from "extended precision", discussed earlier; extended precision is implemented at the hardware level or at the microcode level. A multiple-precision package may allow the user to specify the number of digits to use in representing data and performing computations.

Bailey (1993, 1995)gives software for instrumenting Fortran code to use multiple-precision operations. The C and C++ library GMP, mentioned earlier, provides support for multiple precision floating-point numbers in its `mpf` package. The software packages for symbolic computations, such as Maple or Mathematica, generally provide multiple precision capabilities also.

10.2.4.2 Rational Fractions

Rational fractions are ratios of two integers. If the input data can be represented exactly as rational fractions, it may be possible to preserve exact values of the results of computations. Using rational fractions allows avoidance of reciprocation, which is the operation that most commonly yields a nonrepresentable value from one that is representable. Of course, any addition or multiplication that increases the magnitude of an integer in a rational

fraction beyond a value that can be represented exactly (that is, beyond approximately 2^{23}, 2^{31}, or 2^{53}, depending on the computing system) may break the error-free chain of operations.

Computations with rational fractions are often performed using a fixed-point representation. As mentioned earlier, the GMP library supports mathematical operations involving big integers. This facility is what is needed to use rational fractions to achieve high accuracy or even exactness in computations. The `mpq` package in GMP facilitates use of big integers in operations involving rational fractions.

10.2.4.3 Interval Arithmetic

Interval arithmetic maintains intervals in which the data and results of computations are known to lie. Instead of working with single-point approximations, for which we used notation such as

$$[x]_c$$

on page 469 for the value of the floating-point approximation to the real number x and

$$[\circ]_c$$

on page 484 for the simulated operation $\circ$, we can approach the problem by identifying a closed interval in which x lies and a closed interval in which the result of the operation $\circ$ lies. We denote the interval operation as

$$[\circ]_I.$$

For the real number x, we identify two floating-point numbers, x_l and x_u, such that $x_l \leq x \leq x_u$. (This relationship also implies $x_l \leq [x]_c \leq x_u$.) The real number x is then considered to be the interval $[x_l, x_u]$. For this approach to be useful, of course, we seek tight bounds. If $x = [x]_c$, the best interval is degenerate. In some other cases, either x_l or x_u is $[x]_c$, and the length of the interval is the floating-point spacing from $[x]_c$ in the appropriate direction.

Addition and multiplication in interval arithmetic yield the intervals

$$x\ [+]_I\ y = [x_l + y_l,\ x_u + y_u]$$

and

$$x\ [*]_I\ y = [\min(x_l y_l,\ x_l y_u,\ x_u y_l,\ x_u y_u),\ \max(x_l y_l,\ x_l y_u,\ x_u y_l,\ x_u y_u)].$$

A change of sign results in $[-x_u,\ -x_l]$ and if $0 \notin [x_l,\ x_u]$, reciprocation results in $[1/x_u,\ 1/x_l]$. See Moore (1979) or Alefeld and Herzberger (1983) for discussions of these kinds of operations and an extensive treatment of interval arithmetic. The journal *Reliable Computing* is devoted to interval computations. The book edited by Kearfott and Kreinovich (1996) addresses

various aspects of interval arithmetic. One chapter in that book, by Walster (1996), discusses how both hardware and system software could be designed to implement interval arithmetic.

Most software support for interval arithmetic is provided through subroutine libraries. The `ACRITH` package of IBM (see Jansen and Weidner 1986) is a library of Fortran subroutines that perform computations in interval arithmetic and also in multiple precision. Kearfott et al. (1994) have produced a portable Fortran library of basic arithmetic operations and elementary functions in interval arithmetic, and Kearfott (1996) gives a Fortran 90 module defining an interval data type. Jaulin et al. (2001) describe additional sources of software. Sun Microsystems Inc. provided full intrinsic support for interval data types in their Fortran compiler Sun™ ONE Studio Fortran 95 (now distributed and maintained by Oracle, Inc.); see Walster (2005) for a description of the compiler extensions. The IEEE Standard P1788 specifies interval arithmetic operations based on intervals whose endpoints are IEEE binary64 floating-point numbers. The standard ensures exact propagation of properties of the computed results.

10.2.5 Exact Computations

Computations involving integer quantities can easily be performed exactly if the number of available bits to represent the integers is large enough. The use of rational fractions described above is the most common way of achieving exact computations. Gregory and Krishnamurthy (1984) discuss in detail these and other methods for performing error-free computations.

Another way of performing exact computations is by using *residue arithmetic*, in which each quantity is represented as a vector of residues, all from a vector of relatively prime integer moduli. For details of the use of residue arithmetic in numerical computations, we refer the reader to Szabó and Tanaka (1967), and for examples of applications of this technology in matrix computations, we refer to Stallings and Boullion (1972) and Keller-McNulty and Kennedy (1986).

10.2.5.1 Exact Dot Product (EDP)

As mentioned before, a common computational problem that motivates efforts for exact (or at least highly accurate) computations is computing a dot product. The ability to compute an exact dot product, or EDP, is desirable.

One approach, as indicated above, is to store every bit of the input floating-point numbers in very long vectors of bits, called accumulators. Kulisch (2011) proposed accumulators of 4288 bits each. The Kulisch accumulator can handle the exact accumulation of products of 64-bit IEEE floating-point values. This approach incurs a large memory overhead, however. Also, unless it is accomplished directly in the hardware, vectorization of computations would be very difficult. The IFIP Working Group 2.5 on Numerical Software has proposed that the EDP be incorporated into a new IEEE 754 standard (see page 474).

10.3 Numerical Algorithms and Analysis

We will use the term "algorithm" rather loosely but always in the general sense of a *method* or a *set of instructions* for doing something. (Formally, an "algorithm" must terminate; however, respecting that definition would not allow us to refer to a method as an algorithm until it has been proven to terminate.) Algorithms are sometimes distinguished as "numerical", "semi-numerical", and "nonnumerical", depending on the extent to which operations on real numbers are simulated.

10.3.1 Algorithms and Programs

Algorithms are expressed by means of a flowchart, a series of steps, or in a computer language or pseudolanguage. The expression in a computer language is a source program or module; hence, we sometimes use the words "algorithm" and "program" synonymously.

The program is the set of computer instructions that implement the algorithm. A poor implementation can render a good algorithm useless. A good implementation will preserve the algorithm's accuracy and efficiency and will detect data that are inappropriate for the algorithm. Robustness is more a property of the program than of the algorithm.

The exact way an algorithm is implemented in a program depends of course on the programming language, but it also may depend on the computer and associated system software. A program that will run on most systems without modification is said to be *portable*.

The two most important aspects of a computer algorithm are its accuracy and its efficiency. Although each of these concepts appears rather simple on the surface, both are actually fairly complicated, as we shall see.

10.3.2 Error in Numerical Computations

An "accurate" algorithm is one that gets the "right" answer. Knowing that the right answer may not be representable and that rounding within a set of operations may result in variations in the answer, we often must settle for an answer that is "close". As we have discussed previously, we measure error, or closeness, as either the absolute error or the relative error of a computation.

Another way of considering the concept of "closeness" is by looking backward from the computed answer and asking what perturbation of the original problem would yield the computed answer exactly. This approach, developed by Wilkinson (1963), is called *backward error analysis*. The backward analysis is followed by an assessment of the effect of the perturbation on the solution. Although backward error analysis may not seem as natural as "forward" analysis (in which we assess the difference between the computed and true solutions), it is easier to perform because all operations in the backward

analysis are performed in $\mathbb{F}$ instead of in $\mathbb{R}$. Each step in the backward analysis involves numbers in the set $\mathbb{F}$, that is, numbers that could actually have participated in the computations that were performed. Because the properties of the arithmetic operations in $\mathbb{R}$ do not hold and, at any step in the sequence of computations, the result in $\mathbb{R}$ may not exist in $\mathbb{F}$, it is very difficult to carry out a forward error analysis.

There are other complications in assessing errors. Suppose the answer is a vector, such as a solution to a linear system. What norm do we use to compare the closeness of vectors? Another, more complicated situation for which assessing correctness may be difficult is random number generation. It would be difficult to assign a meaning to "accuracy" for such a problem.

The basic source of error in numerical computations is the inability to work with the reals. The field of reals is simulated with a finite set. This has several consequences. A real number is rounded to a floating-point number; the result of an operation on two floating-point numbers is rounded to another floating-point number; and passage to the limit, which is a fundamental concept in the field of reals, is not possible in the computer.

Rounding errors that occur just because the result of an operation is not representable in the computer's set of floating-point numbers are usually not too bad. Of course, if they accumulate through the course of many operations, the final result may have an unacceptably large rounding error.

A natural approach to studying errors in floating-point computations is to define random variables for the rounding at all stages, from the initial representation of the operands, through any intermediate computations, to the final result. Given a probability model for the rounding error in the representation of the input data, a statistical analysis of rounding errors can be performed. Wilkinson (1963) introduced a uniform probability model for rounding of input and derived distributions for computed results based on that model. Linnainmaa (1975) discusses the effects of accumulated errors in floating-point computations based on a more general model of the rounding for the input. This approach leads to a forward error analysis that provides a probability distribution for the error in the final result.

The obvious probability model for floating-point representations is that the reals within an interval between any two floating-point numbers have a uniform distribution (see Fig. 10.4 on page 471 and Calvetti 1991). A probability model for the real line can be built up as a mixture of the uniform distributions (see Exercise 10.10 on page 519). The density is obviously 0 in the tails. While a model based on simple distributions may be appropriate for the rounding error due to the finite-precision *representation* of real numbers, probability models for rounding errors in floating point *computations* are not so simple. This is because the rounding errors in computations are not random. See Chaitin-Chatelin and Frayssé (1996) for a further discussion of probability models for rounding errors. Dempster and Rubin (1983) discuss the application of statistical methods for dealing with grouped data to the data resulting from rounding in floating-point computations.

Another, more pernicious, effect of rounding can occur in a single operation, resulting in catastrophic cancellation, as we have discussed previously (see page 488).

10.3.2.1 Measures of Error and Bounds for Errors

For the simple case of representing the real number r by an approximation $\tilde{r}$, we define absolute error, $|\tilde{r} - r|$, and relative error, $|\tilde{r} - r|/|r|$ (so long as $r \neq 0$). These same types of measures are used to express the errors in numerical computations. As we indicated above, however, the result may not be a simple real number; it may consist of several real numbers. For example, in statistical data analysis, the numerical result, $\tilde{r}$, may consist of estimates of several regression coefficients, various sums of squares and their ratio, and several other quantities. We may then be interested in some more general measure of the difference of $\tilde{r}$ and r,

$$\Delta(\tilde{r}, r),$$

where $\Delta(\cdot, \cdot)$ is a nonnegative, real-valued function. This is the absolute error, and the relative error is the ratio of the absolute error to $\Delta(r, r_0)$, where r_0 is a baseline value, such as 0. When r, instead of just being a single number, consists of several components, we must measure error differently. If r is a vector, the measure may be based on some norm, and in that case, $\Delta(\tilde{r}, r)$ may be denoted by $\|(\tilde{r} - r)\|$. A norm tends to become larger as the number of elements increases, so instead of using a raw norm, it may be appropriate to scale the norm to reflect the number of elements being computed.

However the error is measured, for a given algorithm, we would like to have some knowledge of the amount of error to expect or at least some bound on the error. Unfortunately, almost any measure contains terms that depend on the quantity being evaluated. Given this limitation, however, often we can develop an upper bound on the error. In other cases, we can develop an estimate of an "average error" based on some assumed probability distribution of the data comprising the problem.

In a Monte Carlo method, we estimate the solution based on a "random" sample, so one source of error in addition to any numerical computational errors, is the "sampling error". Just as in ordinary statistical estimation, we are concerned about the variance of the estimate, which we relate to the sampling error. We can usually derive expressions for the variance of the estimator in terms of the quantity being evaluated, and of course we can estimate the variance of the estimator using the realized random sample. The standard deviation of the estimator provides an indication of the distance around the computed quantity within which we may have some confidence that the true value lies. The standard deviation is sometimes called the "standard error", or the "probabilistic error bound".

It is often useful to identify the "order of the error" whether we are concerned about error bounds, average expected error, or the standard deviation

of an estimator. In general, we speak of the order of one function in terms of another function as a common argument of the functions approaches a given value. A function $f(t)$ is said to be of order $g(t)$ at t_0, written $O(g(t))$ ("big O of $g(t)$"), if there exists a positive constant M such that

$$|f(t)| \leq M|g(t)| \quad \text{as } t \to t_0.$$

This is the *order of convergence* of one function to another function at a given point.

If our objective is to compute $f(t)$ and we use an approximation $\tilde{f}(t)$, the order of the *error due to the approximation* is the order of the convergence. In this case, the argument of the order of the error may be some variable that defines the approximation. For example, if $\tilde{f}(t)$ is a finite series approximation to $f(t)$ using, say, k terms, we may express the error as $O(h(k))$ for some function $h(k)$. Typical orders of errors due to the approximation may be $O(1/k)$, $O(1/k^2)$, or $O(1/k!)$. An approximation with order of error $O(1/k!)$ is to be preferred over one order of error $O(1/k)$ because the error is decreasing more rapidly. The order of error due to the approximation is only one aspect to consider; roundoff error in the representation of any intermediate quantities must also be considered.

We will discuss the order of error in iterative algorithms further in Sect. 10.3.4 beginning on page 510. (We will discuss order also in measuring the speed of an algorithm in Sect. 10.3.3.)

The special case of convergence to the constant zero is often of interest. A function $f(t)$ is said to be "little o of $g(t)$" at t_0, written $o(g(t))$, if

$$f(t)/g(t) \to 0 \quad \text{as } t \to t_0.$$

If the function $f(t)$ approaches 0 at t_0, $g(t)$ can be taken as a constant and $f(t)$ is said to be $o(1)$.

Big O and little o convergences are defined in terms of dominating functions. In the analysis of algorithms, it is often useful to consider analogous types of convergence in which the function of interest dominates another function. This type of relationship is similar to a lower bound. A function $f(t)$ is said to be $\Omega(g(t))$ ("big omega of $g(t)$") if there exists a positive constant m such that

$$|f(t)| \geq m|g(t)| \quad \text{as } t \to t_0.$$

Likewise, a function $f(t)$ is said to be "little omega of $g(t)$" at t_0, written $\omega(g(t))$, if

$$g(t)/f(t) \to 0 \quad \text{as } t \to t_0.$$

Usually the limit on t in order expressions is either 0 or ∞, and because it is obvious from the context, mention of it is omitted. The order of the error in numerical computations usually provides a measure in terms of something that can be controlled in the algorithm, such as the point at which an infinite series is truncated in the computations. The measure of the error usually also contains expressions that depend on the quantity being evaluated, however.

10.3.2.2 Error of Approximation

Some algorithms are exact, such as an algorithm to multiply two matrices that just uses the definition of matrix multiplication. In this case, there may be numerical errors due to rounding or other computational events, but no errors due to the computational approach. Here we focus on additional errors that are due to the algorithms not being exact.

Algorithms may be approximate either because the algorithm itself is iterative, such as one using the Gauss-Seidel method (page 280), or because the algorithm makes use of approximations to the desired value, possibly because the result to be computed does not have a finite closed-form expression. An example of the latter is the evaluation of the normal cumulative distribution function. One way of evaluating this is by using a rational polynomial approximation to the distribution function. Such an expression may be evaluated with very little rounding error, but the expression has an *error of approximation*. On the other hand, the component of the error in an iterative algorithm due to eventually having to halt the iterations is an *error of truncation*. The truncation error is also an error of approximation.

When solving a differential equation on the computer, the differential equation is often approximated by a difference equation. Even though the differences used may not be constant, they are finite and the passage to the limit can never be effected. This kind of approximation leads to a *discretization error*. The amount of the discretization error has nothing to do with rounding error. If the last differences used in the algorithm are δt, then the error is usually of order $O(\delta t)$, even if the computations are performed exactly.

The type of error of approximation that occurs when an algorithm uses a series expansion is similar to the error that occurs in using an iterative algorithm. The series may be exact, and in principle the evaluation of all terms would yield an exact result. The algorithm uses only a finite number of terms, and the resulting component of the error is *truncation error*. This is the type of error we discussed in connection with Fourier expansions on pages 42 and 99. Often the exact expansion is an infinite series, and we approximate it with a finite series. When a truncated Taylor series is used to evaluate a function at a given point x_0, the order of the truncation error is the derivative of the function that would appear in the first unused term of the series, evaluated at x_0.

We need to have some knowledge of the magnitude of the error. For algorithms that use approximations, it is often useful to express the order of the error in terms of some quantity used in the algorithm or in terms of some aspect of the problem itself. We must be aware, however, of the limitations of such measures of the errors or error bounds. For an oscillating function, for example, the truncation error may never approach zero over any nonzero interval.

10.3.2.3 Algorithms and Data

The performance of an algorithm may depend on the data. We have seen that even the simple problem of computing the roots of a quadratic polynomial, $ax^2 + bx + c$, using the quadratic formula, equation (10.3), can lead to severe cancellation. For many values of a, b, and c, the quadratic formula works perfectly well. Data that are likely to cause computational problems are referred to as ill-conditioned data, and, more generally, we speak of the "condition" of data. The concept of condition is understood in the context of a particular set of operations. Heuristically, data for a given problem are ill-conditioned if small changes in the data may yield large changes in the solution.

Consider the problem of finding the roots of a high-degree polynomial, for example. Wilkinson (1959) gave an example of a polynomial that is very simple on the surface yet whose solution is very sensitive to small changes of the values of the coefficients:

$$\begin{aligned} f(x) &= (x-1)(x-2)\cdots(x-20) \\ &= x^{20} - 210x^{19} + \cdots + 20!. \end{aligned} \tag{10.4}$$

While the solution is easy to see from the factored form, the solution is very sensitive to perturbations of the coefficients. For example, changing the coefficient 210 to $210 + 2^{-23}$ changes the roots drastically; in fact, ten of them are now complex. Of course, the extreme variation in the magnitudes of the coefficients should give us some indication that the problem may be ill-conditioned.

10.3.2.4 Condition of Data

We attempt to quantify the condition of a set of data for a particular set of operations by means of a *condition number*. Condition numbers are defined to be positive and in such a way that large values of the numbers mean that the data or problems are ill-conditioned. A useful condition number for the problem of finding roots of a differentiable function can be defined to be increasing as the reciprocal of the absolute value of the derivative of the function in the vicinity of a root.

In the solution of a linear system of equations, the coefficient matrix determines the condition of the problem. The most commonly used condition number for this problem is the one based on ratio of eigenvalues (or singular values) that we discussed in Sect. 6.1.1 on page 267.

Condition numbers are only indicators of possible numerical difficulties for a given problem. They must be used with some care. For example, according to the condition number for finding roots based on the derivative, Wilkinson's polynomial is well-conditioned.

10.3.2.5 Robustness of Algorithms

The ability of an algorithm to handle a wide range of data and either to solve the problem as requested or to determine that the condition of the data does not allow the algorithm to be used is called the *robustness* of the algorithm.

10.3.2.6 Stability of Algorithms

Another concept that is quite different from robustness is *stability*. An algorithm is said to be *stable* if it always yields a solution that is an *exact* solution to a perturbed problem; that is, for the problem of computing $f(x)$ using the input data x, an algorithm is stable if the result it yields, $\tilde{f}(x)$, is

$$f(x + \delta x)$$

for some (bounded) perturbation δx of x. Stated another way, an algorithm is stable if small perturbations in the input or in intermediate computations do not result in large differences in the results.

The concept of stability for an algorithm should be contrasted with the concept of condition for a problem or a dataset. If a problem is ill-conditioned, a stable algorithm (a "good algorithm") will produce results with large differences for small differences in the specification of the problem. This is because the exact results have large differences. An algorithm that is not stable, however, may produce large differences for small differences in the computer description of the problem, which may involve rounding, truncation, or discretization, or for small differences in the intermediate computations performed by the algorithm.

The concept of stability arises from backward error analysis. The stability of an algorithm may depend on how continuous quantities are discretized, such as when a range is gridded for solving a differential equation. See Higham (2002) for an extensive discussion of stability.

10.3.2.7 Reducing the Error in Numerical Computations

An objective in designing an algorithm to evaluate some quantity is to avoid accumulated rounding error and to avoid catastrophic cancellation. In the discussion of floating-point operations above, we have seen two examples of how an algorithm can be constructed to mitigate the effect of accumulated rounding error (using equations (10.2) on page 488 for computing a sum) and to avoid possible catastrophic cancellation in the evaluation of the expression (10.3) for the roots of a quadratic equation.

Another example familiar to statisticians is the computation of the sample sum of squares:

$$\sum_{i=1}^{n}(x_i - \bar{x})^2 = \sum_{i=1}^{n} x_i^2 - n\bar{x}^2. \tag{10.5}$$

This quantity is $(n-1)s^2$, where s^2 is the sample variance.

Either expression in equation (10.5) can be thought of as describing an algorithm. The expression on the left-hand side implies the "two-pass" algorithm:

$$\begin{aligned}
&a = x_1 \\
&\text{for } i = 2, \ldots, n \\
&\{ \\
&\quad a = x_i + a \\
&\} \\
&a = a/n \\
&b = (x_1 - a)^2 \\
&\text{for } i = 2, \ldots, n \\
&\{ \\
&\quad b = (x_i - a)^2 + b \\
&\}.
\end{aligned} \tag{10.6}$$

This algorithm yields $\bar{x} = a$ and $(n-1)s^2 = b$. Each of the sums computed in this algorithm may be improved by using equations (10.2). A problem with this algorithm is the fact that it requires two passes through the data. Because the quantities in the second summation are squares of residuals, they are likely to be of relatively equal magnitude. They are of the same sign, so there will be no catastrophic cancellation in the early stages when the terms being accumulated are close in size to the current value of b. There will be some accuracy loss as the sum b grows, but the addends $(x_i - a)^2$ remain roughly the same size. The accumulated rounding error, however, may not be too bad.

The expression on the right-hand side of equation (10.5) implies the "one-pass" algorithm:

$$\begin{aligned}
&a = x_1 \\
&b = x_1^2 \\
&\text{for } i = 2, \ldots, n \\
&\{ \\
&\quad a = x_i + a \\
&\quad b = x_i^2 + b \\
&\} \\
&a = a/n \\
&b = b - na^2.
\end{aligned} \tag{10.7}$$

This algorithm requires only one pass through the data, but if the x_is have magnitudes larger than 1, the algorithm has built up two relatively large quantities, b and na^2. These quantities may be of roughly equal magnitudes; subtracting one from the other may lead to catastrophic cancellation (see Exercise 10.17, page 520).

Another algorithm is shown in equations (10.8). It requires just one pass through the data, and the individual terms are generally accumulated fairly

accurately. Equations (10.8) are a form of the Kalman filter (see, for example, Grewal and Andrews 1993).

$$\begin{aligned} &a = x_1 \\ &b = 0 \\ &\text{for } i = 2, \ldots, n \\ &\{ \\ &\quad d = (x_i - a)/i \\ &\quad a = d + a \\ &\quad b = i(i-1)d^2 + b \\ &\}. \end{aligned} \tag{10.8}$$

A useful measure to quantify the sensitivity of s, the sample standard deviation, to the data, the x_is, is the condition number

$$\kappa = \frac{\sum_{i=1}^{n} x_i^2}{\sqrt{n-1}s}. \tag{10.9}$$

This is a measure of the "stiffness" of the data. It is clear that if the mean is large relative to the variance, this condition number will be large. (Recall that large condition numbers imply ill-conditioning, and also recall that condition numbers must be interpreted with some care.) Notice that this condition number achieves its minimum value of 1 for the data $x_i - \bar{x}$, so if the computations for $\bar{x}$ and $x_i - \bar{x}$ were exact, the data in the last part of the algorithm in equations (10.6) would be perfectly conditioned. A dataset with a large mean relative to the variance is said to be *stiff*.

Often when a finite series is to be evaluated, it is necessary to accumulate a set of terms of the series that have similar magnitudes, and then combine this with similar partial sums. It may also be necessary to scale the individual terms by some very large or very small multiplicative constant while the terms are being accumulated and then remove the scale after some computations have been performed.

Chan, Golub, and LeVeque (1982) propose a modification of the algorithm in equations (10.8) to use pairwise accumulations (as in the fan-in method discussed previously). Chan, Gene, and LeVeque (1983) make extensive comparisons of the methods and give error bounds based on the condition number.

10.3.3 Efficiency

The *efficiency* of an algorithm refers to its usage of computer resources. The two most important resources are the processing units and the memory. The amount of time the processing units are in use and the amount of memory required are the key measures of efficiency. A limiting factor for the time the processing units are in use is the number and type of operations required. Some operations take longer than others; for example, the operation of adding floating-point numbers may take more time than the operation of adding fixed-point numbers. This, of course, depends on the computer system and on what

kinds of floating-point or fixed-point numbers we are dealing with. If we have a measure of the size of the problem, we can characterize the performance of a given algorithm by specifying the number of operations of each type or just the number of operations of the slowest type.

10.3.3.1 Measuring Efficiency

Often, instead of the exact number of operations, we use the *order* of the number of operations in terms of the measure of problem size. If n is some measure of the size of the problem, an algorithm has order $O(f(n))$ if, as $n \to \infty$, the number of computations $\to cf(n)$, where c is some constant that does not depend on n. For example, to multiply two $n \times n$ matrices in the obvious way requires $O(n^3)$ multiplications and additions; to multiply an $n \times m$ matrix and an $m \times p$ matrix requires $O(nmp)$ multiplications and additions. In the latter case, n, m, and p are all measures of the size of the problem.

Notice that in the definition of order there is a constant c. Two algorithms that have the same order may have different constants and in that case are said to "differ only in the constant". The order of an algorithm is a measure of how well the algorithm "scales"; that is, the extent to which the algorithm can deal with truly large problems.

Let n be a measure of the problem size, and let b and q be positive constants. An algorithm of order $O(b^n)$ has *exponential order*, one of order $O(n^q)$ has *polynomial order*, and one of order $O(\log n)$ has *log order*. Notice that for log order it does not matter what the base is. Also, notice that $O(\log n^q) = O(\log n)$. For a given task with an obvious algorithm that has polynomial order, it is often possible to modify the algorithm to address parts of the problem so that in the order of the resulting algorithm one n factor is replaced by a factor of $\log n$.

Although it is often relatively easy to determine the order of an algorithm, an interesting question in algorithm design involves the *order of the problem*; that is, the order of the most efficient algorithm possible. A problem of polynomial order is usually considered tractable, whereas one of exponential order may require a prohibitively excessive amount of time for its solution. An interesting class of problems are those for which a solution can be verified in polynomial time yet for which no polynomial algorithm is known to exist. Such a problem is called a *nondeterministic polynomial*, or NP, problem. "Nondeterministic" does not imply any randomness; it refers to the fact that no polynomial algorithm for determining the solution is known. Most interesting NP problems can be shown to be equivalent to each other in order by reductions that require polynomial time. Any problem in this subclass of NP problems is equivalent in some sense to all other problems in the subclass and so such a problem is said to be *NP-complete*.

For many problems it is useful to measure the size of a *problem* in some standard way and then to identify the order of an *algorithm* for the problem

with separate components. A common measure of the size of a problem is L, the length of the stream of data elements. An $n \times n$ matrix would have length proportional to $L = n^2$, for example. To multiply two $n \times n$ matrices in the obvious way requires $O(L^{3/2})$ multiplications and additions, as we mentioned above.

In analyzing algorithms for more complicated problems, we may wish to determine the order in the form

$$O(f(n)g(L))$$

because L is an essential measure of the problem size and n may depend on how the computations are performed. For example, in the linear programming problem, with n variables and m constraints with a dense coefficient matrix, there are order nm data elements. Algorithms for solving this problem generally depend on the limit on n, so we may speak of a linear programming algorithm as being $O(n^3L)$, for example, or of some other algorithm as being $O(\sqrt{n}L)$. (In defining L, it is common to consider the magnitudes of the data elements or the precision with which the data are represented, so that L is the order of the total number of bits required to represent the data. This level of detail can usually be ignored, however, because the limits involved in the order are generally not taken on the magnitude of the data but only on the number of data elements.)

The order of an algorithm (or, more precisely, the "order of *operations* of an algorithm") is an asymptotic measure of the operation count as the size of the problem goes to infinity. The order of an algorithm is important, but in practice the actual count of the operations is also important. In practice, an algorithm whose operation count is approximately n^2 may be more useful than one whose count is $1000(n \log n + n)$, although the latter would have order $O(n \log n)$, which is much better than that of the former, $O(n^2)$. When an algorithm is given a fixed-size task many times, the finite efficiency of the algorithm becomes very important.

The number of computations required to perform some tasks depends not only on the size of the problem but also on the data. For example, for most sorting algorithms, it takes fewer computations (comparisons) to sort data that are already almost sorted than it does to sort data that are completely unsorted. We sometimes speak of the *average* time and the *worst-case* time of an algorithm. For some algorithms, these may be very different, whereas for other algorithms or for some problems these two may be essentially the same.

Our main interest is usually not in how many computations occur but rather in how long it takes to perform the computations. Because some computations can take place simultaneously, even if all kinds of computations required the same amount of time, the *order of time* could be different from the order of the number of computations.

The actual number of floating-point operations divided by the time in seconds required to perform the operations is called the FLOPS (floating-point operations per second) rate. Confusingly, "FLOP" also means "floating-point

operation", and "FLOPs" is the plural of "FLOP". Of course, as we tend to use lowercase more often, we must use the context to distinguish "flops" as a rate from "flops" the plural of "flop".

In addition to the actual processing, the data may need to be copied from one storage position to another. Data movement slows the algorithm and may cause it not to use the processing units to their fullest capacity. When groups of data are being used together, blocks of data may be moved from ordinary storage locations to an area from which they can be accessed more rapidly. The efficiency of a program is enhanced if all operations that are to be performed on a given block of data are performed one right after the other. Sometimes a higher-level language prevents this from happening. For example, to add two arrays (matrices) in modern Fortran, a single statement is sufficient:

```
A = B + C
```

Now, if we also want to add `B` to the array `E`, we may write

```
A = B + C
D = B + E
```

These two Fortran statements together may be less efficient than writing a traditional loop in Fortran or in C because the array `B` may be accessed a second time needlessly. (Of course, this is relevant only if these arrays are very large.)

10.3.3.2 Improving Efficiency

There are many ways to attempt to improve the efficiency of an algorithm. Often the best way is just to look at the task from a higher level of detail and attempt to construct a new algorithm. Many obvious algorithms are serial methods that would be used for hand computations, and so are not the best for use on the computer.

An effective general method of developing an efficient algorithm is called *divide and conquer*. In this method, the problem is broken into subproblems, each of which is solved, and then the subproblem solutions are combined into a solution for the original problem. In some cases, this can result in a net savings either in the number of computations, resulting in an improved order of computations, or in the number of computations that must be performed serially, resulting in an improved order of time.

Let the time required to solve a problem of size n be $t(n)$, and consider the recurrence relation

$$t(n) = pt(n/p) + cn$$

for p positive and c nonnegative. Then $t(n) \in \mathrm{O}(n \log n)$ (see Exercise 10.19, page 521). Divide and conquer strategies can sometimes be used together with a simple method that would be $\mathrm{O}(n^2)$ if applied directly to the full problem to reduce the order to $\mathrm{O}(n \log n)$.

The "fan-in algorithm" (see page 487) is an example of a divide and conquer strategy that allows $O(n)$ operations to be performed in $O(\log n)$ time if the operations can be performed simultaneously. The number of operations does not change materially; the improvement is in the time.

Although there have been orders of magnitude improvements in the speed of computers because the hardware is better, the order of time required to solve a problem is almost entirely dependent on the algorithm. The improvements in efficiency resulting from hardware improvements are generally differences only in the constant. The practical meaning of the order of the time must be considered, however, and so the constant may be important. In the fan-in algorithm, for example, the improvement in order is dependent on the unrealistic assumption that as the problem size increases without bound, the number of processors also increases without bound. Divide and conquer strategies do not require multiple processors for their implementation, of course.

Some algorithms are designed so that each step is as efficient as possible, without regard to what future steps may be part of the algorithm. An algorithm that follows this principle is called a *greedy algorithm*. A greedy algorithm is often useful in the early stages of computation for a problem or when a problem lacks an understandable structure.

10.3.3.3 Scalability

We expect to devote more resources to solving large-scale problems than to solving smaller problems. Whether or not the additional resources allow the larger problems to be solved in an acceptable length of time depends on the system or process being used to solve the problem, as well as on the nature of the problem itself. For example, in adding a set of numbers, the fan-in algorithm requires additional adders as the size of the problem grows. Assuming that these additional resources can be provided, the time required is of log order.

A system or process is called *scalable* if, as the size of the problem increases, additional resources can be provided to the system or process so that its performance is not badly degraded. The addition of resources is called scaling the system.

Scaling a system can be done in various ways. More random-access memory can be added to a computer or more cores can be added to the CPU, for example. This is sometimes called *scaling up* the system, because the basic structure of the system does not change. In very large-scale problems, the system itself can be expanded into one that consists of distributed computing systems. This is called *scaling out* the system

As a general concept, the words scalable and scalability are useful, but because of the imprecision, I do not use the words often. The order of computations or of time expressed as a function of the size of the problem and the resources available are more meaningful measures, although, as we have seen above, there may be inherent ambiguities in those measures.

10.3.3.4 Bottlenecks and Limits

There is a maximum FLOPS rate possible for a given computer system. This rate depends on how fast the individual processing units are, how many processing units there are, and how fast data can be moved around in the system. The more efficient an algorithm is, the closer its achieved FLOPS rate is to the maximum FLOPS rate.

For a given computer system, there is also a maximum FLOPS rate possible for a given problem. This has to do with the nature of the tasks within the given problem. Some kinds of tasks can utilize various system resources more easily than other tasks. If a problem can be broken into two tasks, T_1 and T_2, such that T_1 must be brought to completion before T_2 can be performed, the total time required for the problem depends more on the task that takes longer. This tautology (which is sometimes called someone's "law") has important implications for the limits of efficiency of algorithms. The speedup of problems that consist of both tasks that must be performed sequentially and tasks that can be performed in parallel is limited by the time required for the sequential tasks.

The efficiency of an algorithm may depend on the organization of the computer, the implementation of the algorithm in a programming language, and the way the program is compiled.

10.3.3.5 High-Performance Computing

In "high-performance" computing major emphasis is placed on computational efficiency. The architecture of the computer becomes very important, and the software is designed to take advantage of the particular characteristics of the computer on which it is to run.

The three main architectural elements are memory, central processing units, and communication paths. A controlling unit oversees how these elements work together.

There are various ways memory can be organized. There is usually a hierarchy of types of memory with different speeds of access. The various levels can also be organized into banks with separate communication links to the processing units.

The processing units can be constructed and organized in various ways. A single processor can be "vectorized", so that it can perform the same operation on all elements of two vectors at the same time. Vectorized processing is very important in numerical linear algebra, because so many of the computations naturally are performed on vectors. An EDP processor would also be very useful.

There may be multiple central processing units. The units may consist of multiple cores within the same processor. The processing units may include vector processors.

If more than one processing unit is available, it may be possible to perform different kinds of operations simultaneously. In this case, the amount of time required may be drastically smaller for an efficient parallel algorithm than it would for the most efficient serial algorithm that utilizes only one processor at a time. An analysis of the efficiency must take into consideration how many processors are available, how many computations can be performed in parallel, and how often they can be performed in parallel.

Levesque and Wagenbreth (2010) provide a good overview of the various designs and their relevance to high-performance computing.

10.3.3.6 Computations in Parallel

The most effective way of decreasing the time required for solving a computational problem is to perform the computations in parallel if possible. There are some computations that are essentially serial, but in almost any problem there are subtasks that are independent of each other and can be performed in any order. Parallel computing remains an important research area. See Nakano (2012) for a summary discussion of parallel computing. In an increasing number of problems in the data sciences, distributed computing, which can be considered an extreme form of parallel computing, is used because the data for the problems reside on different servers. See Kshemkalyani and Singhal (2011) for discussions of distributed computing.

10.3.4 Iterations and Convergence

Many numerical algorithms are iterative; that is, groups of computations form successive approximations to the desired solution. In a program this usually means a loop through a common set of instructions in which each pass through the loop changes the initial values of operands in the instructions.

We will generally use the notation $x^{(k)}$ to refer to the computed value of x at the k^{th} iteration.

An iterative algorithm terminates when some *convergence criterion* or *stopping criterion* is satisfied. An example is to declare that an algorithm has converged when

$$\Delta(x^{(k)}, x^{(k-1)}) \leq \epsilon,$$

where $\Delta(x^{(k)}, x^{(k-1)})$ is some measure of the difference of $x^{(k)}$ and $x^{(k-1)}$ and ϵ is a small positive number. Because x may not be a single number, we must consider general measures of the difference of $x^{(k)}$ and $x^{(k-1)}$. For example, if x is a vector, the measure may be some metric, such as we discuss in Chap. 2. In that case, $\Delta(x^{(k)}, x^{(k-1)})$ may be denoted by $\|x^{(k)} - x^{(k-1)}\|$.

An iterative algorithm may have more than one stopping criterion. Often, a maximum number of iterations is set so that the algorithm will be sure to terminate whether it converges or not. (Some people define the term "algorithm" to refer only to methods that converge. Under this definition, whether

or not a method is an "algorithm" may depend on the input data unless a stopping rule based on something independent of the data, such as the number of iterations, is applied. In any event, it is always a good idea, in addition to stopping criteria based on convergence of the solution, to have a stopping criterion that is independent of convergence and that limits the number of operations.)

The *convergence ratio* of the sequence $x^{(k)}$ to a constant x_0 is

$$\lim_{k\to\infty} \frac{\Delta(x^{(k+1)}, x_0)}{\Delta(x^{(k)}, x_0)}$$

if this limit exists. If the convergence ratio is greater than 0 and less than 1, the sequence is said to converge *linearly*. If the convergence ratio is 0, the sequence is said to converge *superlinearly*.

Other measures of the rate of convergence are based on

$$\lim_{k\to\infty} \frac{\Delta(x^{(k+1)}, x_0)}{(\Delta(x^{(k)}, x_0))^r} = c \tag{10.10}$$

(again, assuming the limit exists; i.e., $c < \infty$). In equation (10.10), the exponent r is called the *rate of convergence*, and the limit c is called the *rate constant*. If $r = 2$ (and c is finite), the sequence is said to converge *quadratically*. It is clear that for any $r > 1$ (and finite c), the convergence is superlinear.

Convergence defined in terms of equation (10.10) is sometimes referred to as "Q-convergence" because the criterion is a quotient. Types of convergence may then be referred to as "Q-linear", "Q-quadratic", and so on.

The convergence rate is often a function of k, say $h(k)$. The convergence is then expressed as an order in k, $\mathrm{O}(h(k))$.

10.3.4.1 Extrapolation

As we have noted, many numerical computations are performed on a discrete set that approximates the reals or $\mathbb{R}^d$, resulting in *discretization errors*. By "discretization error", we do not mean a rounding error resulting from the computer's finite representation of numbers. The discrete set used in computing some quantity such as an integral is often a grid. If h is the interval width of the grid, the computations may have errors that can be expressed as a function of h. For example, if the true value is x and, because of the discretization, the *exact value* that would be computed is x_h, then we can write

$$x = x_h + e(h).$$

For a given algorithm, suppose the error $e(h)$ is proportional to some power of h, say h^n, and so we can write

$$x = x_h + ch^n \tag{10.11}$$

for some constant c. Now, suppose we use a different discretization, with interval length rh having $0 < r < 1$. We have

$$x = x_{rh} + c(rh)^n$$

and, after subtracting from equation (10.11),

$$0 = x_h - x_{rh} + c(h^n - (rh)^n)$$

or

$$ch^n = \frac{(x_h - x_{rh})}{r^n - 1}. \tag{10.12}$$

This analysis relies on the assumption that the error in the discrete algorithm is proportional to h^n. Under this assumption, ch^n in equation (10.12) is the discretization error in computing x, using exact computations, and is an estimate of the error due to discretization in actual computations. A more realistic regularity assumption is that the error is $\mathrm{O}(h^n)$ as $h \to 0$; that is, instead of (10.11), we have

$$x = x_h + ch^n + \mathrm{O}(h^{n+\alpha})$$

for $\alpha > 0$.

Whenever this regularity assumption is satisfied, equation (10.12) provides us with an inexpensive improved estimate of x:

$$x_R = \frac{x_{rh} - r^n x_h}{1 - r^n}. \tag{10.13}$$

It is easy to see that $|x - x_R|$ is less than the absolute error using an interval size of either h or rh.

The process described above is called *Richardson extrapolation*, and the value in equation (10.13) is called the Richardson extrapolation estimate. Richardson extrapolation is also called "Richardson's deferred approach to the limit". It has general applications in numerical analysis, but is most widely used in numerical quadrature. Bickel and Joseph (1988) use Richardson extrapolation to reduce the computations in a bootstrap. Extrapolation can be extended beyond just one step, as in the presentation above.

Reducing the computational burden by using extrapolation is very important in higher dimensions. In many cases, for example in direct extensions of quadrature rules, the computational burden grows exponentially with the number of dimensions. This is sometimes called "the curse of dimensionality" and can render a fairly straightforward problem in one or two dimensions unsolvable in higher dimensions.

A direct extension of Richardson extrapolation in higher dimensions would involve extrapolation in each direction, with an exponential increase in the amount of computation. An approach that is particularly appealing in higher dimensions is splitting extrapolation, which avoids independent extrapolations in all directions. See Liem, Lü, and Shih (1995) for an extensive discussion of splitting extrapolation, with numerous applications.

10.3.5 Other Computational Techniques

In addition to techniques to improve the efficiency and the accuracy of computations, there are also special methods that relate to the way we build programs or store and access data.

10.3.5.1 Recursion

The algorithms for many computations perform some operation, update the operands, and perform the operation again.

1. perform operation
2. test for exit
3. update operands
4. go to 1

If we give this algorithm the name `doit` and represent its operands by x, we could write the algorithm as

Algorithm `doit`(x)
1. operate on x
2. test for exit
3. update x: x'
4. `doit`(x')

The algorithm for computing the mean and the sum of squares (10.8) on page 504 can be derived as a recursion. Suppose we have the mean a_k and the sum of squares s_k for k elements $x_1, x_2, \ldots, x_k$, and we have a new value x_{k+1} and wish to compute a_{k+1} and s_{k+1}. The obvious solution is

$$a_{k+1} = a_k + \frac{x_{k+1} - a_k}{k+1}$$

and

$$s_{k+1} = s_k + \frac{k(x_{k+1} - a_k)^2}{k+1}.$$

These are the same computations as in equations (10.8) on page 504.

Another example of how viewing the problem as an update problem can result in an efficient algorithm is in the evaluation of a polynomial of degree d,

$$p_d(x) = c_d x^d + c_{d-1} x^{d-1} + \cdots + c_1 x + c_0.$$

Doing this in a naive way would require $d-1$ multiplications to get the powers of x, d additional multiplications for the coefficients, and d additions. If we write the polynomial as

$$p_d(x) = x(c_d x^{d-1} + c_{d-1} x^{d-2} + \cdots + c_1) + c_0,$$

we see a polynomial of degree $d-1$ from which our polynomial of degree d can be obtained with but one multiplication and one addition; that is, the number of multiplications is equal to the increase in the degree—not two times the increase in the degree. Generalizing, we have

$$p_d(x) = x(\cdots x(x(c_d x + c_{d-1}) + \cdots) + c_1) + c_0, \tag{10.14}$$

which has a total of d multiplications and d additions. The method for evaluating polynomials in equation (10.14) is called *Horner's method*.

A computer subprogram that implements recursion invokes itself. Not only must the programmer be careful in writing the recursive subprogram, but the programming system must maintain call tables and other data properly to allow for recursion. Once a programmer begins to understand recursion, there may be a tendency to overuse it. To compute a factorial, for example, the inexperienced C programmer may write

```
float Factorial(int n)
   {
    if(n==0)
      return 1;
    else
      return n*Factorial(n-1);
   }
```

The problem is that this is implemented by storing a stack of statements. Because n may be relatively large, the stack may become quite large and inefficient. It is just as easy to write the function as a simple loop, and it would be a much better piece of code.

Fortran, R, Python, and C all allow for recursion.

10.3.5.2 Computations Without Storing Data

For computations involving large sets of data, it is desirable to have algorithms that sequentially use a single data record, update some cumulative data, and then discard the data record. Such an algorithm is called a *real-time* algorithm, and operation of such an algorithm is called *online* processing. An algorithm that has all of the data available throughout the computations is called a *batch* algorithm.

An algorithm that generally processes data sequentially in a similar manner as a real-time algorithm but may have subsequent access to the same data is called an *online* algorithm or an *"out-of-core"* algorithm. (This latter name derives from the erstwhile use of "core" to refer to computer memory.) Any real-time algorithm is an online or out-of-core algorithm, but an online or out-of-core algorithm may make more than one pass through the data. (Some people restrict "online" to mean "real-time" as we have defined it above.)

If the quantity t is to be computed from the data $x_1, x_2, \ldots, x_n$, a real-time algorithm begins with a quantity $t^{(0)}$, and from $t^{(0)}$ and x_1 computes $t^{(1)}$.

The algorithm proceeds to compute $t^{(2)}$ using x_2 and so on, never retaining more than just the current value, $t^{(k)}$. The quantities $t^{(k)}$ may of course consist of multiple elements, but the point is that the number of elements in each $t^{(k)}$ is independent of n.

Many summary statistics can be computed in online or real-time processes. For example, the algorithms discussed beginning on page 503 for computing the sample sum of squares are real-time algorithms. The algorithm in equations (10.6) requires two passes through the data so it is not a real-time algorithm, although it is out-of-core in each pass. There are stable online algorithms for other similar statistics, such as the sample variance-covariance matrix. The least squares linear regression estimates can also be computed by a stable one-pass algorithm that, incidentally, does not involve computation of the variance-covariance matrix (or the sums of squares and cross products matrix). There is no real-time algorithm for finding the median. The number of data records that must be retained and reexamined depends on n.

It is interesting to note that any sufficient statistic can be computed by an out-of-core algorithm.

In addition to the reduced storage burden, a real-time algorithm allows a statistic computed from one sample to be updated using data from a new sample. A real-time algorithm is necessarily $O(n)$.

10.3.5.3 MapReduce

In very-large-scale computational problems, the data may be stored in different locations and in different formats. In order to process the data in any systematic way, we need to scale out the processing system. To do this, we first need to map it into some common structure and then combine the individual pieces. One standard way of doing this is called *MapReduce*, because of these two separate types of operations.

In an application of MapReduce, it is first assumed that the problem consists of, or can be divided into, separate parts. The mapping phase of MapReduce operates on individual parts of the problem, and within each part of the problem, it assigns identifying keys to the separate values. The result of this phase is a collection of sets of key-value pairs.

The next step is to "shuffle" the elements in the sets of key-value pairs to form a new collection of sets each of which has a single key. These individual sets are then "reduced", that is, the actual computations are performed. Finally, the individual computed results are combined.

Here we will consider a simpler example, so that the individual steps are clear. Consider the problem of counting how many different numbers there are in a given matrix. In Fig. 10.13, we show a 4×4 matrix with three different elements, -1, 0, and 1. The problem is to determine how many elements of each value are in the matrix. In the MapReduce method, each key-value pair is an element count.

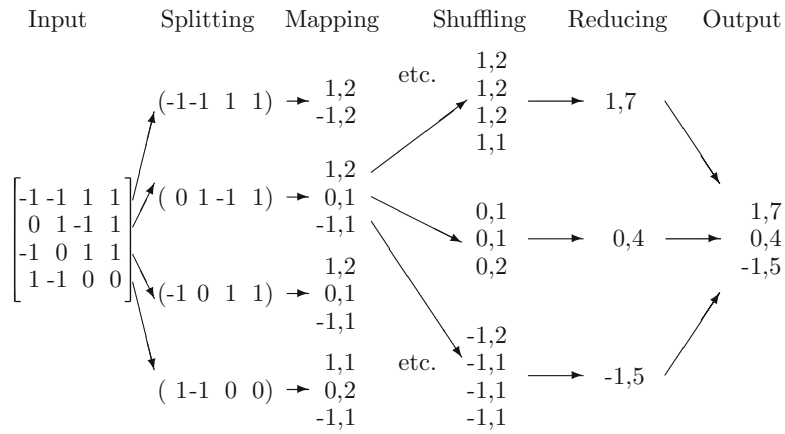


Figure 10.13. MapReduce to count the number of different elements in a matrix

The implementation of this procedure in a distributed computing environment would entail consideration of many details about the component computing environments and the interaction with the file system. As mentioned earlier, the Hadoop Distributed File System (HDFS) is designed for this kind of process.

This kind of problem does not arise often in numerical computations, but it illustrates a scalable approach that could be used in a similar problem in which the matrix is composed of elements distributed over multiple computer systems.

Our purpose here is only to get an overview of the big picture; not to discuss the details of the implementation. We will consider the MapReduce method in the specific context of matrix multiplication on page 533. Additional discussion of the MapReduce method is available in Parsian (2015).

Exercises

10.1. An important attitude in the computational sciences is that the computer is to be used as a tool for exploration and discovery. The computer should be used to check out "hunches" or conjectures, which then later should be subjected to analysis in the traditional manner. There are limits to this approach, however. An example is in limiting processes. Because the computer deals with finite quantities, the results of a computation may be misleading. Explore each of the situations below using C or Fortran. A few minutes or even seconds of computing should be enough to give you a feel for the nature of the computations. In these exercises, you may write computer programs in which you perform tests for equality. A word of warning is in order about such tests.

If a test involving a quantity `x` is executed soon after the computation of `x`, the test may be invalid within the set of floating-point numbers with which the computer nominally works. This is because the test may be performed using the extended precision of the computational registers.

a) Consider the question of the convergence of the series

$$\sum_{i=1}^{\infty} i.$$

Obviously, this series does not converge in $\mathbb{R}$. Suppose, however, that we begin summing this series using floating-point numbers. Will the computations overflow? If so, at what value of i (approximately)? Or will the series converge in $\mathbb{F}$? If so, to what value, and at what value of i (approximately)? In either case, state your answer in terms of the standard parameters of the floating-point model, b, p, $e_{\min}$, and $e_{\max}$ (page 469).

b) Consider the question of the convergence of the series

$$\sum_{i=1}^{\infty} 2^{-2i}$$

and answer the same questions as in Exercise 10.1a.

c) Consider the question of the convergence of the series

$$\sum_{i=1}^{\infty} \frac{1}{i}$$

and answer the same questions as in Exercise 10.1a.

d) Consider the question of the convergence of the series

$$\sum_{i=1}^{\infty} \frac{1}{i^x},$$

for $x \geq 1$. Answer the same questions as in Exercise 10.1a, except address the variable x.

10.2. We know, of course, that the harmonic series in Exercise 10.1c does not converge (although the naive program to compute it does). It is, in fact, true that

$$\begin{aligned} H_n &= \sum_{i=1}^{n} \frac{1}{i} \\ &= f(n) + \gamma + o(1), \end{aligned}$$

where f is an increasing function and γ is Euler's constant. For various n, compute H_n. Determine a function f that provides a good fit and obtain an approximation of Euler's constant.

10.3. Machine characteristics.
   a) Write a program to determine the smallest and largest relative spacings. Use it to determine them on the machine you are using.
   b) Write a program to determine whether your computer system implements gradual underflow.
   c) Write a program to determine the bit patterns of $+\infty$, $-\infty$, and NaN on a computer that implements the IEEE binary standard. (This may be more difficult than it seems.)
   d) i. Obtain the program MACHAR (Cody 1988) and use it to determine the smallest positive floating-point number on the computer you are using. (MACHAR is included in CALGO, which is available from `netlib`. See the Bibliography.)
      ii. Alternatively, or in addition, determine these values on the computer you are using by use of the Fortran intrinsic functions listed in Table 10.3.
      iii. Alternatively, or in addition, determine these values on the computer you are using by use of the R variable `.Machine`.

10.4. Write a program in Fortran or C to determine the bit patterns of fixed-point numbers, floating-point numbers, and character strings. Run your program on different computers, and compare your results with those shown in Figs. 10.1 through 10.3 and Figs. 10.11 and 10.12.

10.5. Install and load the `gmp` package in R. Store 11^{2000} and 35^{1500} as `bigz` values and compute their product. How many digits are in the product?

10.6. What is the numerical value of the rounding unit ($\frac{1}{2}$ ulp) in the IEEE Standard 754 double precision?

10.7. Consider the standard model (10.1) for the floating-point representation,

$$\pm 0.d_1 d_2 \cdots d_p \times b^e,$$

with $e_{\min} \le e \le e_{\max}$. Your answers to the following questions may depend on an additional assumption or two. Either choice of (standard) assumptions is acceptable.
   a) How many floating-point numbers are there?
   b) What is the smallest positive number?
   c) What is the smallest number larger than 1?
   d) What is the smallest number X such that $X + 1 = X$?
   e) Suppose $p = 4$ and $b = 2$ (and $e_{\min}$ is very small and $e_{\max}$ is very large). What is the next number after 20 in this number system?

10.8. a) Define parameters of a floating-point model so that the number of numbers in the system is less than the largest number in the system.
   b) Define parameters of a floating-point model so that the number of numbers in the system is greater than the largest number in the system.

10.9. Suppose that a certain computer represents floating-point numbers in base 10 using eight decimal places for the mantissa, two decimal places for the exponent, one decimal place for the sign of the exponent, and one decimal place for the sign of the number.
   a) What are the "smallest relative spacing" and the "largest relative spacing"? (Your answer may depend on certain additional assumptions about the representation; state any assumptions.)
   b) What is the largest number g such that $417 + g = 417$?
   c) Discuss the associativity of addition using numbers represented in this system. Give an example of three numbers, a, b, and c, such that using this representation $(a + b) + c \neq a + (b + c)$ unless the operations are chained. Then show how chaining could make associativity hold for some more numbers but still not hold for others.
   d) Compare the maximum rounding error in the computation $x+x+x+x$ with that in $4 * x$. (Again, you may wish to mention the possibilities of chaining operations.)

10.10. Consider the same floating-point system as in Exercise 10.9.
   a) Let X be a random variable uniformly distributed over the interval

$$[1 - .000001,\ 1 + .000001].$$

   Develop a probability model for the representation $[X]_c$. (This is a discrete random variable with 111 mass points.)
   b) Let X and Y be random variables uniformly distributed over the same interval as above. Develop a probability model for the representation $[X + Y]_c$. (This is a discrete random variable with 41 mass points.)
   c) Develop a probability model for $[X]_c\ [+]_c\ [Y]_c$. (This is also a discrete random variable with 41 mass points.)

10.11. Give an example to show that the sum of three floating-point numbers can have a very large relative error.

10.12. Write a single program in Fortran or C to compute the following
   a)

$$\sum_{i=0}^{5} \binom{10}{i} 0.25^i 0.75^{20-i}.$$

   b)

$$\sum_{i=0}^{10} \binom{20}{i} 0.25^i 0.75^{20-i}.$$

   c)

$$\sum_{i=0}^{50} \binom{100}{i} 0.25^i 0.75^{20-i}.$$

10.13. In standard mathematical libraries, there are functions for $\log(x)$ and $\exp(x)$ called `log` and `exp` respectively. There is a function in the IMSL Libraries to evaluate $\log(1+x)$ and one to evaluate $(\exp(x)-1)/x$. (The names in Fortran for single precision are `alnrel` and `exprl`.)
   a) Explain why the designers of the libraries included those functions, even though `log` and `exp` are available.
   b) Give an example in which the standard log loses precision. Evaluate it using `log` in the standard math library of Fortran or C. Now evaluate it using a Taylor series expansion of $\log(1+x)$.

10.14. Suppose you have a program to compute the cumulative distribution function for the chi-squared distribution. The input for the program is x and df, and the output is $\Pr(X \leq x)$. Suppose you are interested in probabilities in the extreme upper range and high accuracy is very important. What is wrong with the design of the program for this problem? What kind of program would be better?

10.15. Write a program in Fortran or C to compute e^{-12} using a Taylor series directly, and then compute e^{-12} as the reciprocal of e^{12}, which is also computed using a Taylor series. Discuss the reasons for the differences in the results. To what extent is truncation error a problem?

10.16. Errors in computations.
   a) Explain the difference in truncation and cancellation.
   b) Why is cancellation not a problem in multiplication?

10.17. Assume we have a computer system that can maintain seven digits of precision. Evaluate the sum of squares for the dataset $\{9000, 9001, 9002\}$.
   a) Use the algorithm in equations (10.6) on page 503.
   b) Use the algorithm in equations (10.7) on page 503.
   c) Now assume there is one guard digit. Would the answers change?

10.18. Develop algorithms similar to equations (10.8) on page 504 to evaluate the following.
   a) The weighted sum of squares

$$\sum_{i=1}^{n} w_i (x_i - \bar{x})^2.$$

   b) The third central moment

$$\sum_{i=1}^{n} (x_i - \bar{x})^3.$$

   c) The sum of cross products

$$\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y}).$$

Hint: Look at the difference in partial sums,

$$\sum_{i=1}^{j}(\cdot) - \sum_{i=1}^{j-1}(\cdot).$$

10.19. Given the recurrence relation

$$t(n) = pt(n/p) + cn$$

for p positive and c nonnegative, show that $t(n)$ is $\mathrm{O}(n \log n)$.
Hint: First assume n is a power of p.

10.20. In statistical data analysis, it is common to have some missing data. This may be because of nonresponse in a survey questionnaire or because an experimental or observational unit dies or discontinues participation in the study. When the data are recorded, some form of missing-data indicator must be used. Discuss the use of NaN as a missing-value indicator. What are some of its advantages and disadvantages?

10.21. Consider the four properties of a dot product listed on page 24. For each one, state whether the property holds in computer arithmetic. Give examples to support your answers.

10.22. Assuming the model (10.1) on page 469 for the floating-point number system, give an example of a nonsingular 2×2 matrix that is algorithmically singular.

10.23. A Monte Carlo study of condition number and size of the matrix.
For $n = 5, 10, \ldots, 30$, generate 100 $n \times n$ matrices whose elements have independent $\mathrm{N}(0, 1)$ distributions. For each, compute the L_2 condition number and plot the mean condition number versus the size of the matrix. At each point, plot error bars representing the sample "standard error" (the standard deviation of the sample mean at that point). How would you describe the relationship between the condition number and the size?
In any such Monte Carlo study we must consider the extent to which the random samples represent situations of interest. (How often do we have matrices whose elements have independent $\mathrm{N}(0, 1)$ distributions?)

11

Numerical Linear Algebra

Many scientific computational problems in various areas of application involve vectors and matrices. Programming languages such as C provide the capabilities for working with the individual elements but not directly with the arrays. Modern Fortran and higher-level languages such as Octave or Matlab and R allow direct manipulation of objects that represent vectors and matrices. The vectors and matrices are arrays of floating-point numbers.

The distinction between the set of real numbers, $\mathbb{R}$, and the set of floating-point numbers, $\mathbb{F}$, that we use in the computer has important implications for numerical computations. As we discussed in Sect. 10.2, beginning on page 483, an element x of a vector or matrix is approximated by a computer number $[x]_c$, and a mathematical operation $\circ$ is simulated by a computer operation $[\circ]_c$. The familiar laws of algebra for the field of the reals do not hold in $\mathbb{F}$, especially if uncontrolled parallel operations are allowed. These distinctions, of course, carry over to arrays of floating-point numbers that represent real numbers, and the properties of vectors and matrices that we discussed in earlier chapters may not hold for their computer counterparts. For example, the dot product of a nonzero vector with itself is positive (see page 24), but $\langle x_c, x_c \rangle_c = 0$ does not imply $x_c = 0$.

A good general reference on the topic of numerical linear algebra is Čížková and Čížek (2012).

11.1 Computer Storage of Vectors and Matrices

The elements of vectors and matrices are represented as ordinary numeric data, as we described in Sect. 10.1, in either fixed-point or floating-point representation.

J.E. Gentle, *Matrix Algebra*, Springer Texts in Statistics,
DOI 10.1007/978-3-319-64867-5_11

11.1.1 Storage Modes

The elements of vectors and matrices are generally stored in a logically contiguous area of the computer's memory. What is logically contiguous may not be physically contiguous, however.

Accessing data from memory in a single pipeline may take more computer time than the computations themselves. For this reason, computer memory may be organized into separate modules, or *banks*, with separate paths to the central processing unit. Logical memory is *interleaved* through the banks; that is, two consecutive logical memory locations are in separate banks. In order to take maximum advantage of the computing power, it may be necessary to be aware of how many interleaved banks the computer system has.

There are no convenient mappings of computer memory that would allow matrices to be stored in a logical rectangular grid, so matrices are usually stored either as columns strung end-to-end (a "column-major" storage) or as rows strung end-to-end (a "row-major" storage). In using a computer language or a software package, sometimes it is necessary to know which way the matrix is stored. The type of matrix computation to be performed may determine whether a vectorized processor should operate on rows or on columns.

For some software to deal with matrices of varying sizes, the user must specify the length of one dimension of the array containing the matrix. (In general, the user must specify the lengths of all dimensions of the array except one.) In Fortran subroutines, it is common to have an argument specifying the leading dimension (number of rows), and in C functions it is common to have an argument specifying the column dimension. (See the examples in Fig. 12.2 on page 563 and Fig. 12.3 on page 564 for illustrations of the leading dimension argument.)

11.1.2 Strides

Sometimes in accessing a partition of a given matrix, the elements occur at fixed distances from each other. If the storage is row-major for an $n \times m$ matrix, for example, the elements of a given column occur at a fixed distance of m from each other. This distance is called the "stride", and it is often more efficient to access elements that occur with a fixed stride than it is to access elements randomly scattered.

Just accessing data from the computer's memory contributes significantly to the time it takes to perform computations. A stride that is not a multiple of the number of banks in an interleaved bank memory organization can measurably increase the computational time in high-performance computing.

11.1.3 Sparsity

If a matrix has many elements that are zeros, and if the positions of those zeros are easily identified, many operations on the matrix can be speeded up.

Matrices with many zero elements are called *sparse matrices*. They occur often in certain types of problems; for example in the solution of differential equations and in statistical designs of experiments. The first consideration is how to represent the matrix and to store the matrix and the location information. Different software systems may use different schemes to store sparse matrices. The method used in the IMSL Libraries, for example, is described on page 550. An important consideration is how to preserve the sparsity during intermediate computations.

11.2 General Computational Considerations for Vectors and Matrices

All of the computational methods discussed in Chap. 10 apply to vectors and matrices, but there are some additional general considerations for vectors and matrices.

11.2.1 Relative Magnitudes of Operands

One common situation that gives rise to numerical errors in computer operations is when a quantity x is transformed to $t(x)$ but the value computed is unchanged:

$$[t(x)]_c = [x]_c; \tag{11.1}$$

that is, the operation actually accomplishes nothing. A type of transformation that has this problem is

$$t(x) = x + \epsilon, \tag{11.2}$$

where $|\epsilon|$ is much smaller than $|x|$. If all we wish to compute is $x + \epsilon$, the fact that $[x + \epsilon]_c = [x]_c$ is probably not important. Usually, of course, this simple computation is part of some larger set of computations in which ϵ was computed. This, therefore, is the situation we want to anticipate and avoid.

Another type of problem is the addition to x of a computed quantity y that overwhelms x in magnitude. In this case, we may have

$$[x + y]_c = [y]_c. \tag{11.3}$$

Again, this is a situation we want to anticipate and avoid.

11.2.1.1 Condition

A measure of the worst-case numerical error in numerical computation involving a given mathematical entity is the "condition" of that entity for the particular computations. The *condition number* of a matrix is the most generally useful such measure. For the matrix A, we denote the condition number as $\kappa(A)$. We discussed the condition number in Sect. 6.1 and illustrated it in

the toy example of equation (6.1). The condition number provides a bound on the relative norms of a "correct" solution to a linear system and a solution to a nearby problem. A specific condition number therefore depends on the norm, and we defined κ_1, κ_2, and κ_∞ condition numbers (and saw that they are generally roughly of the same magnitude). We saw in equation (6.10) that the L_2 condition number, $\kappa_2(A)$, is the ratio of magnitudes of the two extreme eigenvalues of A.

The condition of data depends on the particular computations to be performed. The relative magnitudes of other eigenvalues (or singular values) may be more relevant for some types of computations. Also, we saw in Sect. 10.3.2 that the "stiffness" measure in equation (10.3.2.7) is a more appropriate measure of the extent of the numerical error to be expected in computing variances.

11.2.1.2 Pivoting

Pivoting, discussed on page 277, is a method for avoiding a situation like that in equation (11.3). In Gaussian elimination, for example, we do an addition, $x+y$, where the y is the result of having divided some element of the matrix by some other element and x is some other element in the matrix. If the divisor is very small in magnitude, y is large and may overwhelm x as in equation (11.3).

11.2.1.3 "Modified" and "Classical" Gram-Schmidt Transformations

Another example of how to avoid a situation similar to that in equation (11.1) is the use of the correct form of the Gram-Schmidt transformations.

The orthogonalizing transformations shown in equations (2.56) on page 38 are the basis for Gram-Schmidt transformations of matrices. These transformations in turn are the basis for other computations, such as the QR factorization. (Exercise 5.10 required you to apply Gram-Schmidt transformations to develop a QR factorization.)

As mentioned on page 38, there are two ways we can extend equations (2.56) to more than two vectors, and the method given in Algorithm 2.1 is the correct way to do it. At the k^{th} stage of the Gram-Schmidt method, the vector $x_k^{(k)}$ is taken as $x_k^{(k-1)}$ and the vectors $x_{k+1}^{(k)}, x_{k+2}^{(k)}, \ldots, x_m^{(k)}$ are all made orthogonal to $x_k^{(k)}$. After the first stage, all vectors have been transformed. This method is sometimes called "modified Gram-Schmidt" because some people have performed the basic transformations in a different way, so that at the k^{th} iteration, starting at $k = 2$, the first $k-1$ vectors are unchanged (i.e., $x_i^{(k)} = x_i^{(k-1)}$ for $i = 1, 2, \ldots, k-1$), and $x_k^{(k)}$ is made orthogonal to the $k-1$ previously orthogonalized vectors $x_1^{(k)}, x_2^{(k)}, \ldots, x_{k-1}^{(k)}$. This method is called "classical Gram-Schmidt" for no particular reason. The "classical" method is not as stable, and should not be used; see Rice (1966) and Björck (1967) for

discussions. In this book, "Gram-Schmidt" is the same as what is sometimes called "modified Gram-Schmidt". In Exercise 11.1, you are asked to experiment with the relative numerical accuracy of the "classical Gram-Schmidt" and the correct Gram-Schmidt. The problems with the former method show up with the simple set of vectors $x_1 = (1, \epsilon, \epsilon)$, $x_2 = (1, \epsilon, 0)$, and $x_3 = (1, 0, \epsilon)$, with ϵ small enough that

$$[1 + \epsilon^2]_c = 1.$$

11.2.2 Iterative Methods

As we saw in Chap. 6, we often have a choice between direct methods (that is, methods that compute a closed-form solution) and iterative methods. Iterative methods are usually to be favored for large, sparse systems.

Iterative methods are based on a sequence of approximations that (it is hoped) converge to the correct solution. The fundamental trade-off in iterative methods is between the amount of work expended in getting a good approximation at each step and the number of steps required for convergence.

11.2.2.1 Preconditioning

In order to achieve acceptable rates of convergence for iterative algorithms, it is often necessary to precondition the system; that is, to replace the system $Ax = b$ by the system

$$M^{-1}Ax = M^{-1}b$$

for some suitable matrix M. As we indicated in Chaps. 6 and 7, the choice of M involves some art, and we will not consider any of the results here. Benzi (2002) provides a useful survey of the general problem and work up to that time, but this is an area of active research.

11.2.2.2 Restarting and Rescaling

In many iterative methods, not all components of the computations are updated in each iteration. An approximation to a given matrix or vector may be adequate during some sequence of computations without change, but then at some point the approximation is no longer close enough, and a new approximation must be computed. An example of this is in the use of quasi-Newton methods in optimization in which an approximate Hessian is updated, as indicated in equation (4.28) on page 202. We may, for example, just compute an approximation to the Hessian every few iterations, perhaps using second differences, and then use that approximate matrix for a few subsequent iterations.

Another example of the need to restart or to rescale is in the use of fast Givens rotations. As we mentioned on page 241 when we described the fast Givens rotations, the diagonal elements in the accumulated C matrices in

the fast Givens rotations can become widely different in absolute values, so to avoid excessive loss of accuracy, it is usually necessary to rescale the elements periodically. Anda and Park (1994, 1996) describe methods of doing the rescaling dynamically. Their methods involve adjusting the first diagonal element by multiplication by the square of the cosine and adjusting the second diagonal element by division by the square of the cosine. Bindel et al. (2002) discuss in detail techniques for performing Givens rotations efficiently while still maintaining accuracy. (The BLAS routines (see Sect. 12.2.1) `rotmg` and `rotm`, respectively, set up and apply fast Givens rotations.)

11.2.2.3 Preservation of Sparsity

In computations involving large sparse systems, we may want to preserve the sparsity, even if that requires using approximations, as discussed in Sect. 5.10.2. Fill-in (when a zero position in a sparse matrix becomes nonzero) would cause loss of the computational and storage efficiencies of software for sparse matrices.

In forming a preconditioner for a sparse matrix A, for example, we may choose a matrix $M = \widetilde{L}\widetilde{U}$, where $\widetilde{L}$ and $\widetilde{U}$ are approximations to the matrices in an LU decomposition of A, as in equation (5.51). These matrices are constructed as indicated in equation (5.52) so as to have zeros everywhere A has, and $A \approx \widetilde{L}\widetilde{U}$. This is called incomplete factorization, and often, instead of an exact factorization, an approximate factorization may be more useful because of computational efficiency.

11.2.2.4 Iterative Refinement

Even if we are using a direct method, it may be useful to refine the solution by one step computed in extended precision. A method for iterative refinement of a solution of a linear system is given in Algorithm 6.3.

11.2.3 Assessing Computational Errors

As we discuss in Sect. 10.2.2 on page 485, we measure error by a scalar quantity, either as *absolute error*, $|\tilde{r} - r|$, where r is the true value and $\tilde{r}$ is the computed or rounded value, or as *relative error*, $|\tilde{r} - r|/r$ (as long as $r \neq 0$). We discuss general ways of reducing them in Sect. 10.3.2.

11.2.3.1 Errors in Vectors and Matrices

The errors in vectors or matrices are generally expressed in terms of norms; for example, the relative error in the representation of the vector v, or as a result of computing v, may be expressed as $\|\tilde{v} - v\|/\|v\|$ (as long as $\|v\| \neq 0$), where $\tilde{v}$ is the computed vector. We often use the notation $\tilde{v} = v + \delta v$, and

so $\|\delta v\|/\|v\|$ is the relative error. The choice of which vector norm to use may depend on practical considerations about the errors in the individual elements. The L_∞ norm, for example, gives weight only to the element with the largest single error, while the L_1 norm gives weights to all magnitudes equally.

11.2.3.2 Assessing Errors in Given Computations

In real-life applications, the correct solution is not known, but we would still like to have some way of assessing the accuracy using the data themselves. Sometimes a convenient way to do this in a given problem is to perform internal consistency tests. An internal consistency test may be an assessment of the agreement of various parts of the output. Relationships among the output are exploited to ensure that the individually computed quantities satisfy these relationships. Other internal consistency tests may be performed by comparing the results of the solutions of two problems with a known relationship.

The solution to the linear system $Ax = b$ has a simple relationship to the solution to the linear system $Ax = b + ca_j$, where a_j is the j^{th} column of A and c is a constant. A useful check on the accuracy of a computed solution to $Ax = b$ is to compare it with a computed solution to the modified system. Of course, if the expected relationship does not hold, we do not know which solution is incorrect, but it is probably not a good idea to trust either. To test the accuracy of the computed regression coefficients for regressing y on $x_1, \ldots, x_m$, they suggest comparing them to the computed regression coefficients for regressing $y + dx_j$ on $x_1, \ldots, x_m$. If the expected relationships do not obtain, the analyst has strong reason to doubt the accuracy of the computations.

Another simple modification of the problem of solving a linear system with a known exact effect is the permutation of the rows or columns. Although this perturbation of the problem does not change the solution, it does sometimes result in a change in the computations, and hence it may result in a different computed solution. This obviously would alert the user to problems in the computations.

A simple internal consistency test that is applicable to many problems is to use two levels of precision in some of the computations. In using this test, one must be careful to make sure that the input data are the same. Rounding of the input data may cause incorrect output to result, but that is not the fault of the computational algorithm.

Internal consistency tests cannot confirm that the results are correct; they can only give an indication that the results are incorrect.

11.3 Multiplication of Vectors and Matrices

Arithmetic on vectors and matrices involves arithmetic on the individual elements. The arithmetic on the individual elements is performed as we have discussed in Sect. 10.2.

The way the storage of the individual elements is organized is very important for the efficiency of computations. Also, the way the computer memory is organized and the nature of the numerical processors affect the efficiency and may be an important consideration in the design of algorithms for working with vectors and matrices.

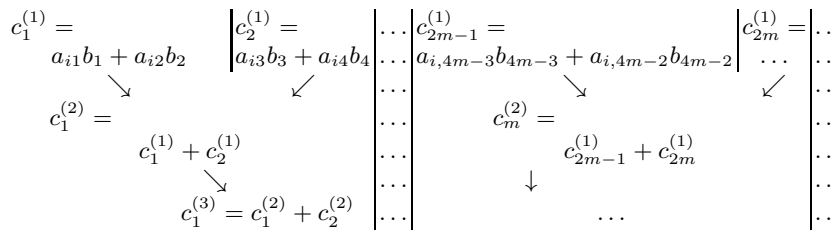
The best methods for performing operations on vectors and matrices in the computer may not be the methods that are suggested by the definitions of the operations.

In most numerical computations with vectors and matrices, there is more than one way of performing the operations on the scalar elements. Consider the problem of evaluating the matrix times vector product, $c = Ab$, where A is $n \times m$. There are two obvious ways of doing this:

- compute each of the n elements of c, one at a time, as an inner product of m-vectors, $c_i = a_i^{\mathrm{T}} b = \sum_j a_{ij} b_j$, or
- update the computation of all of the elements of c simultaneously as

1. For $i = 1, \ldots, n$, let $c_i^{(0)} = 0$.
2. For $j = 1, \ldots, m$,
   {
   for $i = 1, \ldots, n$,
   {
   let $c_i^{(i)} = c_i^{(i-1)} + a_{ij} b_j$.
   }
   }

If there are p processors available for parallel processing, we could use a fan-in algorithm (see page 487) to evaluate Ax as a set of inner products:



The order of the computations is nm (or n^2).

Multiplying two matrices A and B can be considered as a problem of multiplying several vectors b_i by a matrix A, as described above. In the following we will assume A is $n \times m$ and B is $m \times p$, and we will use the notation a_i to represent the i^{th} column of A, a_i^{T} to represent the i^{th} row of A, b_i to represent the i^{th} column of B, c_i to represent the i^{th} column of $C = AB$, and so on. (This notation is somewhat confusing because here we are not using a_i^{T} to represent the transpose of a_i as we normally do. The notation should be

clear in the context of the diagrams below, however.) Using the inner product method above results in the first step of the matrix multiplication forming

$$\left[\begin{array}{c} a_1^{\mathrm{T}} \\ \hline \cdots \\ \ddots \\ \cdots \end{array}\right] \left[\begin{array}{c|c} b_1 & \cdots \\ & \cdots \\ & \ddots \\ & \cdots \end{array}\right] \longrightarrow \left[\begin{array}{c|c} c_{11} = a_1^{\mathrm{T}} b_1 & \cdots \\ \hline & \cdots \\ \vdots & \ddots \\ & \cdots \end{array}\right].$$

Using the second method above, in which the elements of the product vector are updated all at once, results in the first step of the matrix multiplication forming

$$\left[\begin{array}{c|c} a_1 & \cdots \\ & \cdots \\ & \ddots \\ & \cdots \end{array}\right] \left[\begin{array}{c|c} b_{11} & \cdots \\ \hline & \cdots \\ \vdots & \ddots \\ & \cdots \end{array}\right] \longrightarrow \left[\begin{array}{c|c} c_{11}^{(1)} = a_{11} b_{11} & \cdots \\ c_{21}^{(1)} = a_{21} b_{11} & \cdots \\ \vdots & \ddots \\ c_{n1}^{(1)} = a_{n1} b_{11} & \cdots \end{array}\right].$$

The next and each successive step in this method are axpy operations:

$$c_1^{(k+1)} = b_{(k+1),1} a_1 + c_1^{(k)},$$

for k going to $m - 1$.

Another method for matrix multiplication is to perform axpy operations using all of the elements of b_1^{T} before completing the computations for any of the columns of C. In this method, the elements of the product are built as the sum of the outer products $a_i b_i^{\mathrm{T}}$. In the notation used above for the other methods, we have

$$\left[\begin{array}{c|c} a_1 & \cdots \\ & \cdots \\ & \ddots \\ & \cdots \end{array}\right] \left[\begin{array}{c} b_1^{\mathrm{T}} \\ \hline \cdots \\ \ddots \\ \cdots \end{array}\right] \longrightarrow \left[\begin{array}{c} c_{ij}^{(1)} = a_1 b_1^{\mathrm{T}} \\ \\ \end{array}\right],$$

and the update is

$$c_{ij}^{(k+1)} = a_{k+1} b_{k+1}^{\mathrm{T}} + c_{ij}^{(k)}.$$

The order of computations for any of these methods is $\mathrm{O}(nmp)$, or just $\mathrm{O}(n^3)$, if the dimensions are all approximately the same. Strassen's method, discussed next, reduces the order of the computations.

11.3.1 Strassen's Algorithm

Another method for multiplying matrices that can be faster for large matrices is the so-called *Strassen algorithm* (from Strassen 1969). Suppose A and B are square matrices with equal and even dimensions. Partition them into submatrices of equal size, and consider the block representation of the product,

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix},$$

where all blocks are of equal size. Form

$$\begin{aligned} P_1 &= (A_{11} + A_{22})(B_{11} + B_{22}), \\ P_2 &= (A_{21} + A_{22})B_{11}, \\ P_3 &= A_{11}(B_{12} - B_{22}), \\ P_4 &= A_{22}(B_{21} - B_{11}), \\ P_5 &= (A_{11} + A_{12})B_{22}, \\ P_6 &= (A_{21} - A_{11})(B_{11} + B_{12}), \\ P_7 &= (A_{12} - A_{22})(B_{21} + B_{22}). \end{aligned}$$

Then we have (see the discussion on partitioned matrices in Sect. 3.1)

$$\begin{aligned} C_{11} &= P_1 + P_4 - P_5 + P_7, \\ C_{12} &= P_3 + P_5, \\ C_{21} &= P_2 + P_4, \\ C_{22} &= P_1 + P_3 - P_2 + P_6. \end{aligned}$$

Notice that the total number of multiplications is 7 instead of the 8 it would be in forming

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

directly. Whether the blocks are matrices or scalars, the same analysis holds. Of course, in either case there are more additions. The addition of two $k \times k$ matrices is $O(k^2)$, so for a large enough value of n the total number of operations using the Strassen algorithm is less than the number required for performing the multiplication in the usual way.

The partitioning of the matrix factors can also be used recursively; that is, in the formation of the P matrices. If the dimension, n, contains a factor 2^e, the algorithm can be used directly e times, and then conventional matrix multiplication can be used on any submatrix of dimension $\leq n/2^e$.) If the dimension of the matrices is not even, or if the matrices are not square, it may be worthwhile to pad the matrices with zeros, and then use the Strassen algorithm recursively.

The order of computations of the Strassen algorithm is $O(n^{\log_2 7})$, instead of $O(n^3)$ as in the ordinary method ($\log_2 7 = 2.81$). The algorithm can be implemented in parallel (see Bailey et al. 1990), and this algorithm is actually used in some software systems.

Several algorithms have been developed that use similar ideas to Strassen's algorithm and are asymptotically faster; that is, with order of computations $O(n^k)$ where $k < \log_2 7$). (Notice that k must be at least 2 because there

are n^2 elements.) None of the algorithms that are asymptotically faster than Strassen's are competitive in practice, however, because they all have much larger start-up costs.

11.3.2 Matrix Multiplication Using MapReduce

While methods such as Strassen's algorithm achieve speedup by decreasing the total number of computations, other methods increase the overall speed by performing computations in parallel. Although not all computations can be performed in parallel and there is some overhead in additional computations for setting up the job, when multiple processors are available, the total number of computations may not be very important. One of the major tasks in parallel processing is just keeping track of the individual computations. MapReduce (see page 515) can sometimes be used in coordinating these operations.

For the matrix multiplication AB, in the view that the multiplication is a set of inner products, for i running over the indexes of the rows of A and j running over the indexes of the columns of B, we merely access the i^{th} row of A, a_{i*}, and the j^{th} column of B, b_{*j}, and form the inner product $a_{i*}^{\mathrm{T}} b_{*j}$ as the $(i,j)^{\text{th}}$ element of the product AB. In the language of relational databases in which the two matrices are sets of data with row and column identifiers, this amounts to accessing the rows of A and the columns of B one by one, matching the elements of the row and the column so that the column designator of the row element matches the row designator of the column element, summing the product of the A row elements and the B column elements, and then grouping the sums of the products (that is, the inner products) by the A row designators and the B column designators. In SQL, it is

```
SELECT A.row, B.col
   SUM(A.value*B.value) FROM A,B WHERE A.col=B.row
   GROUP BY A.row, B.col;
```

In a distributed computing environment, MapReduce could be used to perform these operations. However the matrices are stored, possibly each over multiple environments, MapReduce would first map the matrix elements using their respective row and column indices as keys. It would then make the appropriate associations of row element from A with the column elements from B and perform the multiplications and the sum. Finally, the sums of the multiplications (that is, the inner products) would be associated with the appropriate keys for the output. This process is described in many elementary descriptions of Hadoop, such as in Leskovec, Rajaraman, and Ullman (2014) (Chapter 2).

11.4 Other Matrix Computations

Many other matrix computations depend on a matrix factorization. The most useful factorization is the QR factorization. It can be computed stably using

either Householder reflections, Givens rotations, or the Gram-Schmidt procedure, as described respectively in Sects. 5.8.8, 5.8.9, and 5.8.10 (beginning on page 252). This is one time when the computational methods can follow the mathematical descriptions rather closely. Iterations using the QR factorization are used in a variety of matrix computations; for example, they are used in the most common method for evaluating eigenvalues, as described in Sect. 7.4, beginning on page 318.

Another very useful factorization is the singular value decomposition (SVD). The computations for SVD described in Sect. 7.7 beginning on page 322, are efficient and preserve numerical accuracy. A major difference in the QR factorization and the SVD is that the computations for SVD are necessarily iterative (recall the remarks at the beginning of Chap. 7).

11.4.1 Rank Determination

It is often easy to determine that a matrix is of full rank. If the matrix is not of full rank, however, or if it is very ill-conditioned, it is often difficult to determine its rank. This is because the computations to determine the rank eventually approximate 0. It is difficult to approximate 0; the relative error (if defined) would be either 0 or infinite. The rank-revealing QR factorization (equation (5.43), page 251) is the preferred method for estimating the rank. (Although I refer to this as "estimation", it more properly should be called "approximation". "Estimation" and the related term "testing", as used in statistical applications, apply to an unknown object, as in estimating or testing the rank of a model matrix as discussed in Sect. 9.5.5, beginning on page 433.) When this decomposition is used to estimate the rank, it is recommended that complete pivoting be used in computing the decomposition. The LDU decomposition, described on page 242, can be modified the same way we used the modified QR to estimate the rank of a matrix. Again, it is recommended that complete pivoting be used in computing the decomposition.

The singular value decomposition (SVD) shown in equation (3.276) on page 161 also provides an indication of the rank of the matrix. For the $n \times m$ matrix A, the SVD is

$$A = UDV^{\mathrm{T}},$$

where U is an $n \times n$ orthogonal matrix, V is an $m \times m$ orthogonal matrix, and D is a diagonal matrix of the singular values. The number of nonzero singular values is the rank of the matrix. Of course, again, the question is whether or not the singular values are zero. It is unlikely that the values computed are exactly zero.

A problem related to rank determination is to approximate the matrix A with a matrix A_r of rank $r \leq \mathrm{rank}(A)$. The singular value decomposition provides an easy way to do this,

$$A_r = UD_rV^{\mathrm{T}},$$

where D_r is the same as D, except with zeros replacing all but the r largest singular values. A result of Eckart and Young (1936) guarantees A_r is the rank r matrix closest to A as measured by the Frobenius norm,

$$\|A - A_r\|_{\mathrm{F}},$$

(see Sect. 3.10). This kind of matrix approximation is the basis for dimension reduction by principal components.

11.4.2 Computing the Determinant

The determinant of a square matrix can be obtained easily as the product of the diagonal elements of the triangular matrix in any factorization that yields an orthogonal matrix times a triangular matrix. As we have stated before, however, it is not often that the determinant need be computed.

One application in statistics is in optimal experimental designs. The D-optimal criterion, for example, chooses the design matrix, X, such that $|X^{\mathrm{T}}X|$ is maximized (see Sect. 9.3.2).

11.4.3 Computing the Condition Number

The computation of a condition number of a matrix can be quite involved. Clearly, we would not want to use the definition, $\kappa(A) = \|A\|\,\|A^{-1}\|$, directly. Although the choice of the norm affects the condition number, recalling the discussion in Sect. 6.1, we choose whichever condition number is easiest to compute or estimate.

Various methods have been proposed to estimate the condition number using relatively simple computations. Cline et al. (1979) suggest a method that is easy to perform and is widely used. For a given matrix A and some vector v, solve

$$A^{\mathrm{T}}x = v$$

and then

$$Ay = x.$$

By tracking the computations in the solution of these systems, Cline et al. conclude that

$$\frac{\|y\|}{\|x\|}$$

is approximately equal to, but less than, $\|A^{-1}\|$. This estimate is used with respect to the L_1 norm in the LINPACK software library (see page 558 and Dongarra et al. 1979), but the approximation is valid for any norm. Solving the two systems above probably does not require much additional work because the original problem was likely to solve $Ax = b$, and solving a system with

multiple right-hand sides can be done efficiently using the solution to one of the right-hand sides. The approximation is better if v is chosen so that $\|x\|$ is as large as possible relative to $\|v\|$.

Stewart (1980) and Cline and Rew (1983) investigated the validity of the approximation. The LINPACK estimator can underestimate the true condition number considerably, although generally not by an order of magnitude. Cline et al. (1982) give a method of estimating the L_2 condition number of a matrix that is a modification of the L_1 condition number used in LINPACK. This estimate generally performs better than the L_1 estimate, but the Cline/Conn/Van Loan estimator still can have problems (see Bischof 1990).

Hager (1984) gives another method for an L_1 condition number. Higham (1988) provides an improvement of Hager's method, given as Algorithm 11.1 below, which is used in the LAPACK software library (Anderson et al. 2000).

Algorithm 11.1 The Hager/Higham LAPACK condition number estimator γ of the $n \times n$ matrix A

Assume $n > 1$; otherwise set $\gamma = \|A\|$. (All norms are L_1 unless specified otherwise.)

0. Set $k = 1$; $v^{(k)} = \frac{1}{n}A1$; $\gamma^{(k)} = \|v^{(k)}\|$; and $x^{(k)} = A^{\mathrm{T}}\mathrm{sign}(v^{(k)})$.
1. Set $j = \min\{i, \text{ s.t. } |x_i^{(k)}| = \|x^{(k)}\|_\infty\}$.
2. Set $k = k + 1$.
3. Set $v^{(k)} = Ae_j$.
4. Set $\gamma^{(k)} = \|v^{(k)}\|$.
5. If $\mathrm{sign}(v^{(k)}) = \mathrm{sign}(v^{(k-1)})$ or $\gamma^{(k)} \leq \gamma^{(k-1)}$, then go to step 8.
6. Set $x^{(k)} = A^{\mathrm{T}}\mathrm{sign}(v^{(k)})$.
7. If $\|x^{(k)}\|_\infty \neq x_j^{(k)}$ and $k \leq k_{\max}$, then go to step 1.
8. For $i = 1, 2, \ldots, n$, set $x_i = (-1)^{i+1}\left(1 + \frac{i-1}{n-1}\right)$.
9. Set $x = Ax$.
10. If $\frac{2\|x\|}{(3n)} > \gamma^{(k)}$, set $\gamma^{(k)} = \frac{2\|x\|}{(3n)}$.
11. Set $\gamma = \gamma^{(k)}$. ∎

Higham (1987) compares Hager's condition number estimator with that of Cline et al. (1979) and finds that the Hager LAPACK estimator is generally more useful. Higham (1990) gives a survey and comparison of the various ways of estimating and computing condition numbers. You are asked to study the performance of the LAPACK estimate using Monte Carlo methods in Exercise 11.5 on page 538.

Exercises

11.1. Gram-Schmidt orthonormalization.

a) Write a program module (in Fortran, C, R, Octave or Matlab, or whatever language you choose) to implement Gram-Schmidt orthonormalization using Algorithm 2.1. Your program should be for an arbitrary order and for an arbitrary set of linearly independent vectors.

b) Write a program module to implement Gram-Schmidt orthonormalization using equations (2.56) and (2.57).

c) Experiment with your programs. Do they usually give the same results? Try them on a linearly independent set of vectors all of which point "almost" in the same direction. Do you see any difference in the accuracy? Think of some systematic way of forming a set of vectors that point in almost the same direction. One way of doing this would be, for a given x, to form $x + \epsilon e_i$ for $i = 1, \ldots, n-1$, where e_i is the i^{th} unit vector and ϵ is a small positive number. The difference can even be seen in hand computations for $n = 3$. Take $x_1 = (1, 10^{-6}, 10^{-6})$, $x_2 = (1, 10^{-6}, 0)$, and $x_3 = (1, 0, 10^{-6})$.

11.2. Given the $n \times k$ matrix A and the k-vector b (where n and k are large), consider the problem of evaluating $c = Ab$. As we have mentioned, there are two obvious ways of doing this: (1) compute each element of c, one at a time, as an inner product $c_i = a_i^{\mathrm{T}} b = \sum_j a_{ij} b_j$, or (2) update the computation of all of the elements of c in the inner loop.

a) What is the order of computation of the two algorithms?

b) Why would the relative efficiencies of these two algorithms be different for different programming languages, such as Fortran and C?

c) Suppose there are p processors available and the fan-in algorithm on page 530 is used to evaluate Ax as a set of inner products. What is the order of time of the algorithm?

d) Give a heuristic explanation of why the computation of the inner products by a fan-in algorithm is likely to have less roundoff error than computing the inner products by a standard serial algorithm. (This does not have anything to do with the parallelism.)

e) Describe how the following approach could be parallelized. (This is the second general algorithm mentioned above.)

```
for i = 1, ..., n
{
  c_i = 0
  for j = 1, ..., k
  {
    c_i = c_i + a_ij b_j
  }
}
```

f) What is the order of time of the algorithms you described?

11.3. Consider the problem of evaluating $C = AB$, where A is $n \times m$ and B is $m \times q$. Notice that this multiplication can be viewed as a set of matrix/vector multiplications, so either of the algorithms in Exercise 11.2d above would be applicable. There is, however, another way of performing this multiplication, in which all of the elements of C could be evaluated simultaneously.

a) Write pseudocode for an algorithm in which the nq elements of C could be evaluated simultaneously. Do not be concerned with the parallelization in this part of the question.

b) Now suppose there are nmq processors available. Describe how the matrix multiplication could be accomplished in $O(m)$ steps (where a step may be a multiplication and an addition).
Hint: Use a fan-in algorithm.

11.4. Write a Fortran or C program to compute an estimate of the L_1 LAPACK condition number γ using Algorithm 11.1 on page 536.

11.5. Design and conduct a Monte Carlo study to assess the performance of the LAPACK estimator of the L_1 condition number using your program from Exercise 11.4. Consider a few different sizes of matrices, say 5×5, 10×10, and 20×20, and consider a range of condition numbers, say 10, 10^4, and 10^8. In order to assess the accuracy of the condition number estimator, the random matrices in your study must have known condition numbers. It is easy to construct a diagonal matrix with a given condition number. The condition number of the diagonal matrix D, with nonzero elements $d_1, \ldots, d_n$, is $\max|d_i| / \min|d_i|$. It is not so clear how to construct a general (square) matrix with a given condition number. The L_2 condition number of the matrix UDV, where U and V are orthogonal matrices is the same as the L_2 condition number of U. We can therefore construct a wide range of matrices with given L_2 condition numbers. In your Monte Carlo study, use matrices with known L_2 condition numbers. The next question is what kind of random matrices to generate. Again, make a choice of convenience. Generate random diagonal matrices D, subject to fixed $\kappa(D) = \max|d_i| / \min|d_i|$. Then generate random orthogonal matrices as described in Exercise 4.10 on page 223. Any conclusions made on the basis of a Monte Carlo study, of course, must be restricted to the domain of the sampling of the study. (See Stewart, 1980, for a Monte Carlo study of the performance of the LINPACK condition number estimator.)

12

Software for Numerical Linear Algebra

There is a variety of computer software available to perform the operations on vectors and matrices discussed in Chap. 11 and previous chapters. We can distinguish software based on various dimensions, including the kinds of applications that the software emphasizes, the level of the objects it works with directly, and whether or not it is interactive. We can also distinguish software based on who "owns" the software and its availability to other users. Many commercial software systems are available from the developers/owners through licensing agreements, and the rights of the user are restricted by the terms of the license, in addition to any copyright.

Some software is designed only to perform a limited number of functions, such as for eigenanalysis, while other software provides a wide range of computations for linear algebra. Some software supports only real matrices and real associated values, such as eigenvalues. In some software systems, the basic units must be scalars, and so operations on matrices or vectors must be performed on individual elements. In these systems, higher-level functions to work directly on the arrays are often built and stored in libraries or supplemental software packages. In other software systems, the array itself is a fundamental operand. Finally, some software for linear algebra is interactive and computations are performed immediately in response to the user's input, while other software systems provide a language for writing procedural descriptions that are to be compiled for later invocation through some interface within the same system or through linkage from a different software system. We often refer to the languages of interactive systems as "interpretive" or "scripting" languages.

12.1 General Considerations

A person's interaction with a software system represents a series of investments. The first investment is in getting access to the system. This may involve spending money, spending time downloading files, and installing the software.

J.E. Gentle, *Matrix Algebra*, Springer Texts in Statistics,
DOI 10.1007/978-3-319-64867-5_12

The investment to acquire access involves "learning" the system: how to start it, how to input data, how to do something, how to output results, and how to shut it down. The next type of investment is usually just the time to use the system to solve problems. The returns on these investments are the solutions to the problems, or just the personal satisfaction in having solved them. A continuing investment is in becoming proficient in use of the software. The return is in time saved. Another important investment is the development of a set of reusable functions and scripts. Functions can often be used in a wide variety of settings far beyond the one that prompted the writing of the function in the first place. Scripts for a particular task can serve as templates for programs to solve a different problem. The return is in time saved and, possibly, improved accuracy, in solving future problems.

As with other investments, an objective is to maximize returns. Although the computing environment is constantly changing, as with other investments, some degree of stability of the investment environment is desirable in order to protect the returns. Formal standards (see pages 462 and 564) help to ensure this.

The software user's primary goal is not investment (usually), but nevertheless the user is making an investment, and with a little forethought can increase the returns on this investment.

Investors' choices depend on the stage of their investment career. If a software investor has not become proficient in any language, that investor has different investment choices from an investor who is very fluent in C, maybe knows a little Python, but doesn't know a word of Fortran.

12.1.1 Software Development and Open Source Software

An important distinction among software systems is the nature of the user's access to the system. Software, as intellectual property, is protected by copyright, just as any other written work (as opposed to an algorithm, which may be protected by patent). Copyright is an unusual type of legal title. In most legal jurisdictions, it is initiated without any explicit registration by the holder; it just comes into being at the time of the creation of the original work.

There are various ways that a copyright of software can be modified. At one extreme is complete revocation of copyright by the author; that is, the author puts the work in the "public domain". Even if the copyright is retained, the owner may allow free use, either with or without the rights to modify. We will not delve into the variations in these schemes, but rather use the phrase "open source" to refer to systems to which the authors have give users free access to the full source code of the system. Many important systems ranging from operating systems, such as Linux, to programming and application-oriented systems, such as R, are open source. Because software by its nature is often developed in open collaboration by many people who may have no connections other than just the work on the software, open-source software is often to be preferred.

12.1.2 Collaborative Research and Version Control

In collaborative development, version-control systems are especially useful. For software development, Git is probably the best. (It was built by Linus Torvalds, the creator of Linux.) The software itself is free and open source. A good interface to the software and to a hosting service is provided at GitHub:

`https://github.com/`

For general collaborative document development (including source programs), Overleaf is particularly useful. Overleaf also provides a TEX processor with most of the common packages attached. It also provides PDF rendering. The web site is

`https://www.overleaf.com/`

The Reproducible R Toolkit (see page 580) is useful for version control in collaborative work as well as for a single user to deal with different versions of R and R packages.

12.1.3 Finding Software

The quickest way to find software is usually to use a web search program.

The Guide to Available Mathematical Software (GAMS) is also a good source of information about software. This guide is organized by types of computations. Computations for linear algebra are in Class D, for example. The web site is

`http://gams.nist.gov/serve.cgi/Class/D/`

Much of the software is available through `statlib` or `netlib` (see page 619 in the Bibliography).

12.1.4 Software Design

There are many properties of software that affect its usefulness. Although we emphasize efficient use of computer resources (fast algorithms and minimal memory usage), it is more important that the user's time and efforts are not wasted. If the general design of the software is consistent with the user's ways of thinking about the application, then the user is more likely to be efficient in the use of the the software and to make fewer errors.

12.1.4.1 Interoperability

For some types of software, it is important to be aware of the way the data are stored in the computer, as we discussed in Sect. 11.1 beginning on page 523. This may include such things as whether the storage is row-major or column-major, which will determine the stride and may determine the details of an

algorithm so as to enhance the efficiency. Software written in a language such as Fortran or C often requires the specification of the number of rows (in Fortran) or columns (in C) that have been allocated for the storage of a matrix. The amount of space allocated for the storage of a matrix may not correspond exactly to the size of the matrix.

There are many issues to consider in evaluating software or to be aware of when developing software. The portability of the software is an important consideration because a user's programs are often moved from one computing environment to another.

12.1.4.2 Efficiency

Computational efficiency, the topic of Sect. 10.3.3, is a very important consideration in the development and use of numerical methods. In that context, efficiency refers to the use of computing resources, which ones are used and for how long they are used for a specific task. In the selection and use of software, the efficiency of the methods implemented in the software is an important consideration, but, depending on the task and the available resources, the user's time to set up the computations and access the results is often more important. Higher-level software and more application-specific software, while the actual computations may be slower, may be more efficient overall.

Some situations require special software that is more efficient than general-purpose software would be for the given task. Software for sparse matrices, for example, is specialized to take advantage of the zero entries.

12.1.4.3 Writing Mathematics and Writing Programs

Although one of the messages of this book is that we should not ignore computational details, there are some trivial details that should not have to enter our conscious thought processes. In well-designed software systems, these details flow naturally in the process of converting mathematics to computer code.

In writing either mathematics or programs, it is generally best to think of objects at the highest level that is appropriate for the problem at hand. The details of some computational procedure may be of the form

$$\sum_i \sum_j \sum_k a_{ki} x_{kj}. \tag{12.1}$$

We sometimes think of the computations in this form because we have programmed them in some low-level language at some time. In some cases, it is important to look at the computations in this form, but usually it is better to think of the computations at a higher level, say

$$A^{\mathrm{T}} X. \tag{12.2}$$

The compactness of the expression is not the issue (although it certainly is more pleasant to read). The issue is that expression (12.1) leads us to think of some nested computational loops, while expression (12.2) leads us to look for more efficient computational modules, such as the BLAS, which we discuss below. In a higher-level language system such as R, the latter expression is more likely to cause us to use the system more efficiently.

12.1.4.4 Numerical Mathematical Objects and Computer Objects

Some difficult design decisions must be made when building systems that provide objects that simulate mathematical objects. One issue is how to treat different mathematical objects that are based on the real number system, such as scalars, vectors, and matrices, when their sizes happen to coincide.

- Is a vector with one element a scalar?
- Is a 1×1 matrix a scalar?
- Is a $1 \times n$ matrix a "row vector"?
- Is an $n \times 1$ matrix a "column vector"?
- Is a "column vector" different from a "row vector"?

While the obvious answer to all these questions is "no", it is often convenient to design software systems as if the answer, at least to some of the questions some of the time, is "yes". The answer to any such software design question always must be made in the context of the purpose and intended use (and users) of the software. The issue is not the purity of a mathematical definition. We have already seen that most computer objects and operators do not behave exactly like the mathematical entities they simulate anyway.

The experience of most people engaged in scientific computations over many years has shown that the convenience resulting from the software's equivalent treatment of such different objects as a 1×1 matrix and a scalar outweighs the programming error detection that could be possible if the objects were made to behave as nearly as possible to the way the mathematical entities they simulate behave.

Consider, for example, the following arrays of numbers:

$$A = [1\ 2], \quad B = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \quad C = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}. \tag{12.3}$$

If these arrays are matrices with the usual matrix algebra, then ABC, where juxtaposition indicates Cayley multiplication, is not a valid expression.

If, however, we are willing to allow mathematical objects to change types, we come up with a reasonable interpretation of ABC. If the 1×1 matrix AB is interpreted as the scalar 5, then the expression $(AB)C$ can be interpreted as $5C$, that is, a scalar times a matrix. (Under Cayley multiplication, of course, we do not need to indicate the order of the operations because the operation is associative.)

There is no (reasonable) interpretation that would make the expression $A(BC)$ valid.

If A is a "row vector" and B is a "column vector", it hardly makes sense to define an operation on them that would yield another vector. A vector space cannot consist of such mixtures. Under a strict interpretation of the operations, $(AB)C$ is not a valid expression.

We often think of the "transpose" of a vector (although this may not be a viable concept in a vector space), and we denote a dot product in a vector space as $x^{\mathrm{T}}y$. If we therefore interpret a row vector such as A in (12.3) as x^{T} for some x in the vector space of which B is a member, then AB can be interpreted as a dot product (that is, as a scalar) and again $(AB)C$ is a valid expression.

Many software packages have a natural atomic numerical type. For example, in Fortran and C the atomic numeric type is a scalar. Other structures such as vectors and matrices are built as simple arrays from scalars. Other software systems are array-oriented. The basic atomic numeric type in Matlab is a matrix. In Matlab, a scalar is a 1×1 matrix. A one-dimensional array in Matlab is a matrix with either one row or one column (Fig. 12.1).

```
>> a = [1;2;3]
 a =
     1
     2
     3
>> b = [1,2,3]
 b =
     1 2 3
>> a(1,3)
 ** error **
>> a(3,1)
 ans =
       3
>> b(1,3)
 ans =
       3
```

Figure 12.1. Indexing in Matlab

The basic atomic numeric type in R is a vector. In R, a scalar is a vector of length 1. A $1 \times n$ matrix and an $n \times 1$ matrix may be cast as vectors (that is, one-dimensional arrays). This is often exactly what we would want, but sometimes we want to preserve the matrix structure (see page 576).

12.1.4.5 Other Mathematical Objects and Computer Objects

Most mathematical operations performed on a computer are based on the two operations of the field of the reals, and the design of scientific software has been focused on these operations. Operations on abstract variables are supported by some software packages. Maple and Mathematica are two such packages, and Matlab also supports some "symbolic computations", as these are called.

Most of our interests in numerical linear algebra are focused on numerical computations of course, but occasionally we want to do other kinds of mathematical operations. Operations on sets, for example, are also supported by some software systems, such as Python and R. The first issue in working with sets, of course, is how to initialize the set. R does not have an object class of "set", but it does provide functions for the standard set operations (see page 578).

12.1.4.6 Software for Statistical Applications

Statistical applications have requirements that go beyond simple linear algebra. The two most common additional requirements are for

- handling metadata and
- accommodating missing data.

Software packages designed for data analysis, such as SAS and R, generally provide for metadata and missing values. Fortran/C libraries generally do not provide for metadata or for handling missing data.

Two other needs that often arise in statistical analysis but often are not dealt with adequately in available software, are

- graceful handling of nonfull rank matrices and
- working with nonsquare matrices.

Aside from these general capabilities, of course, software packages for statistical applications, even if they are designed for some specific type of analysis, should provide the common operations such as computation of simple univariate statistics, linear regression computations, and some simple graphing capabilities.

12.1.4.7 Robustness

Operations on matrices are often viewed from the narrow perspective of the numerical analyst rather than from the broader perspective of a user with a task to perform. For example, the user may seek a solution to the linear system $Ax = b$. Many software packages to solve a linear system requires A to be square and of full rank. If this is not the case, then there are three possibilities: the system has no solution, the system has multiple solutions,

or the system has a unique solution. Software to solve a linear system of equations that requires A to be square and of full rank does not distinguish among these possibilities but rather always refuses to provide any solution. This can be quite annoying to a user who wants to solve a large number of different systems with different properties using the same code. A better design of the software would allow solution of consistent non-square systems and of consistent non-full-rank systems, and report an inconsistent of any structure as such.

A related problem occurs when A is a scalar. Some software to solve the linear system will fail on this degenerate case. (Note, for example, `solve` in R handles this correctly.)

Good software is robust both to extended cases and to degenerate cases.

12.1.4.8 Computing Paradigms

A computational task may involve several sequences of computations, some of which can be performed independently of others. We speak of the separate sequences of operations as *threads*. If the threads can be executed independently, they can easily be performed at the same time, that is, in parallel. Even if they cannot be performed independently, by sharing results among the threads, it may be possible to execute them in parallel. Parallel processing is one on the most important ways of speeding up computations.

Many computational tasks follow a very simple pattern; all data and all processors are together in the same system. The programming model is simple. For many interesting problems, however, the data are stored in different places and in a variety of formats. A simple approach, of course, is to collect and organize the data prior to any actual processing. In many large-scale problems this is not practical, however, due either to the size of the data or to the fact that the process of data generation is ongoing. If the data are not collected in one place, the processing may occur at different locations also. Such distributed computing, which can be thought of as an extreme form of parallel computing, requires a completely different paradigm.

Spark and Hadoop provide such a paradigm. Hadoop and MapReduce, discussed on page 515, together are ideally suited to distributed computing tasks. We will not discuss these methods further here, but White (2015) discusses Hadoop and gives several examples of its use, and Karau et al. (2015) provide a good introduction to Spark.

For most computations in numerical linear algebra, these considerations are not relevant, but as we address larger problems, these issues become much more important.

12.1.4.9 Array Structures and Indexes

Numerical operations on vectors and matrices in most general-purpose procedural languages are performed either within loops of operations on the in-

dividual elements or by invocation of a separate program module. A natural way of representing vectors and matrices is as array variables with indexes.

One of the main differences in various computing systems is how they handle arrays. The most common impact on the user is the indexing of the arrays, and here the most common differences are in where the index begins (0 or 1) and how sequences are handled. To process a mathematical expression $f(x_i)$ over n values of a vector x we use programming statements of the form of either

```
for i = 1:n, compute f(x[i])
```

or

```
for i = 0:(n-1), compute f(x[i])
```

depending on the beginning index.

Indexing in Fortran (by default), Matlab, and R begins with "1", just as in most mathematical notation (as throughout this book, for example). Fortran allows indexing to start at any integer. Indexing for ordinary arrays begins with "0" in Python, C, and C++.

Fortran handles arrays as multiply indexed memory locations, consistent with the nature of the object. The storage of two-dimensional arrays in Fortran is column-major; that is, the array A is stored as $\mathrm{vec}(A)$. To reference the contiguous memory locations, the first subscript varies fastest. In general-purpose software consisting of Fortran subprograms, it is often necessary to specify the lengths of all dimensions of a Fortran array except the last one.

An array in C is an ordered set of memory locations referenced by a pointer or by a name and an index. The indexes are enclosed in rectangular brackets following the variable name. An element of a multidimensional array in C is indexed by multiple indexes, each within rectangular brackets. If the 3×4 matrix A is as stored in the C array `A`, the $(2,3)$ element $A_{2,3}$ is referenced as `A[1][2]`. This disconnect between the usual mathematical representations and the C representations results from the historical development of C by computer scientists, who deal with arrays, rather than by mathematical scientists, who deal with matrices and vectors.

Multidimensional arrays in C are arrays of arrays, in which the array constructors operate from right to left. This results in two-dimensional C arrays being stored in row-major order, that is, the array A is stored as $\mathrm{vec}(A^{\mathrm{T}})$. To reference the contiguous memory locations, the last subscript varies fastest. In general-purpose software consisting of C functions, it is often necessary to specify the lengths of all dimensions of a C array except the first one.

Python provides an array construct similar to that of C, but in addition it also provides several other constructs for multiple values, including one that is effectively a mathematical set; that is, a collection of unique elements.

Subarrays are formed slightly differently in different systems. Consider the vector x with ten elements, which we have stored in a linear array called `x`. The number of elements in the array is `len`, 10 in this case. Suppose we want

to reference various elements within the array. In different systems this is done as shown in Table 12.1.

Table 12.1. Subarrays

	The last five elements	All but last five elements	First, third, ... elements
Fortran	`x(len-4:)`	`x(:len-5)`	`x(::2)`
Matlab	`x(len-4:len)`	`x(1:len-5)`	`x(1:2:len)`
R	`x[(len-4):len]`	`x[1:(len-5)]`	`x[seq(1,len,2)]`
Python	`x[len-5:]`	`x[:len-5]`	`x[::2]`

For people who use different systems, the differences in the indexing is not just annoying; the main problem is that it leads to programming errors. (The reader who does not know Python may also be somewhat perplexed by the first two entries for Python in Table 12.1.)

12.1.4.10 Matrix Storage Modes

There are various ways of indexing matrices that have special structure. Although one of the purposes of special storage modes is to use less computer memory, sometimes it is more natural to think of the elements of a matrix with a special structure in a way that conforms to that structure.

Matrices that have multiple elements with the same value can often be stored in the computer in such a way that the individual elements do not all have separate locations. Symmetric matrices and matrices with many zeros, such as the upper or lower triangular matrices of the various factorizations we have discussed, are examples of matrices that do not require full rectangular arrays for their storage. There are several special modes for storage of matrices with repeated elements, especially if those elements are zeroes.

- symmetric mode
- Hermitian mode
- triangular mode
- band mode
- sparse storage mode

For a symmetric or Hermitian matrix or a triangular matrix, only about half of the elements need to be stored. Although the amount of space saved by not storing the full symmetric matrix is only about one half of the amount of space required, the use of rank 1 arrays rather than rank 2 arrays can yield some reference efficiencies. (Recall that in discussions of computer software objects, "rank" usually means the number of dimensions.) For band matrices and other sparse matrices, the savings in storage can be much larger.

For a symmetric matrix such as

$$A = \begin{bmatrix} 1\ 2\ 4 \cdots \\ 2\ 3\ 5 \cdots \\ 4\ 5\ 6 \cdots \\ \cdots \end{bmatrix}, \tag{12.4}$$

only the unique elements

$$v = (1, 2, 3, 4, 5, 6, \cdots) \tag{12.5}$$

need to be stored.

A special indexing method for storing symmetric matrices, called *symmetric storage mode*, uses a linear array to store only the unique elements. Symmetric storage mode can be set up in various ways.

One form of symmetric storage mode corresponds directly to an arrangement of the vector v in equation (12.5) to hold the data in matrix A in equation (12.4). This symmetric storage mode is a much more efficient and useful method of storing a symmetric matrix than would be achieved by a vech(·) operator because with symmetric storage mode, the size of the matrix affects only the elements of the vector near the end. If the number of rows and columns of the matrix is increased, the length of the vector is increased, but the elements are not rearranged.

For an $n \times n$ symmetric matrix A, the correspondence with the $n(n+1)/2$-vector v is

$$v_{i(i-1)/2+j} = a_{i,j}, \quad \text{for } i \geq j.$$

Notice that the relationship does not involve n. For $i \geq j$, in Fortran, it is

```
v(i*(i-1)/2+j) = a(i,j)
```

In C, because the indexing begins at 0, the correspondence in C arrays is

```
v[i*(i+1)/2+j] = a[i][j]
```

In R, a symmetric matrix such as A in equation (12.4) is often represented in a vector of the form

$$\tilde{v} = (1, 2, 4, \cdots, 3, 5, \cdots, 6, \cdots). \tag{12.6}$$

Notice that this is exactly what the vech(·) operator yields. (There is no `vech` function in any of the common R packages, however.) For an $n \times n$ symmetric matrix A, the correspondence with the $n(n+1)/2$-vector $\tilde{v}$ is

$$\tilde{v}_{i+n(j-1)-j(j-1)/2} = a_{i,j}, \quad \text{for } i \geq j.$$

Notice that the relationship involves n. For a hollow matrix, such as a distance matrix (produced by the `dist` function in R, for example), the diagonal elements are not stored.

In a band storage mode, for the $n \times m$ band matrix A with lower band width w_l and upper band width w_u, an $w_l + w_u \times m$ array is used to store the

elements. The elements are stored in the same column of the array, say `A`, as they are in the matrix; that is,

$$\texttt{A}(i - j + w_u + 1,\, j) = a_{i,j}$$

for $i = 1, 2, \ldots, w_l + w_u + 1$. Band symmetric, band Hermitian, and band triangular modes are all defined similarly. In each case, only the upper or lower bands are referenced.

There are several different schemes for representing sparse matrices, but the most common way is the *index-index-value* method. This method uses three arrays, each of rank 1 and with length equal to the number of nonzero elements. The integer array `i` contains the row indicator, the integer array `j` contains the column indicator, and the floating-point array `a` contains the corresponding values; that is, the (`i`(k), `j`(k)) element of the matrix is stored in `a`(k). The level 3 BLAS (see page 557) for sparse matrices have an argument to allow the user to specify the type of storage mode.

12.1.5 Software Development, Maintenance, and Testing

In software development as in other production processes, good initial design of both the product and the process reduces the need for "acceptance sampling". Nevertheless, there is a need for thorough and systematic testing of software. Tests should be part of the development phase, but can continue throughout the life of the software.

12.1.5.1 Test Data

Testbeds for software consist of test datasets that vary in condition but have known solutions or for which there is an easy way of verifying the solution. Test data may be fixed datasets or randomly generated datasets over some population with known and controllable properties.

For testing software for some matrix computations, a very common matrix is the special Hankel matrix, called the *Hilbert matrix H* that has elements

$$h_{ij} = \frac{1}{i + j - 1}.$$

Hilbert matrices have large condition numbers; for example, the 10×10 Hilbert matrix has condition number of order 10^{13}. The Matlab function `hilb(n)` generates an $n \times n$ Hilbert matrix. A Hilbert matrix is also a special case of a square Cauchy matrix with $x = (n + 1, n + 2, \ldots, 2n)$, $y = (n, n - 1, \ldots, 1)$, $v = 1$, and $w = 1$ (see Sect. 8.8.8 on page 391).

Randomly generated test data can provide general information about the performance of a computational method over a range of datasets with specified characteristics. Examples of studies using randomly generated datasets are the paper by Birkhoff and Gulati (1979) on the accuracy of computed

solutions x_c of the linear system $Ax = b$, where A is $n \times n$ from a BMvN distribution, and the paper by Stewart (1980) using random matrices from a Haar distribution to study approximations to condition numbers (see page 221 and Exercise 4.10). As it turns out, matrices from the BMvN distribution are not sufficiently ill-conditioned often enough to be useful in studies of the accuracy of solutions of linear systems. Birkhoff and Gulati developed a procedure to construct arbitrarily ill-conditioned matrices from ones with a BMvN distribution and then used these matrices in their empirical studies.

Ericksen (1985) describes a method to generate matrices with known inverses in such a way that the condition numbers vary widely. To generate an $n \times n$ matrix A, choose $x_1, x_2, \ldots, x_n$ arbitrarily, except such that $x_1 \neq 0$, and then take

$$\begin{array}{ll} a_{1j} = x_1 & \text{for } j = 1, \ldots, n, \\ a_{i1} = x_i & \text{for } i = 2, \ldots, n, \\ a_{ij} = a_{i,j-1} + a_{i-1,j-1} & \text{for } i, j = 2, \ldots, n. \end{array} \tag{12.7}$$

To represent the elements of the inverse, first define $y_1 = x_1^{-1}$, and for $i = 2, \ldots, n$,

$$y_i = -y_1 \sum_{k=0}^{i-1} x_{i-k} y_k.$$

Then the elements of the inverse of A, $B = (b_{ij})$, are given by

$$\begin{array}{ll} b_{in} = (-1)^{i+k} \binom{n-1}{i-1} y_1 & \text{for } i = 1, \ldots, n, \\ b_{nj} = y_{n+1-j} & \text{for } j = 1, \ldots, n-1, \\ b_{ij} = x_1 b_{in} b_{nj} + \sum_{k=i+1}^{n} b_{k,j+1} & \text{for } i, j = 1, \ldots, n-1, \end{array} \tag{12.8}$$

where the binomial coefficient, $\binom{k}{m}$, is defined to be 0 if $k < m$ or $m < 0$.

The nonzero elements of L and U in the LU decomposition of A are easily seen to be $l_{ij} = x_{i+1-j}$ and $u_{ij} = \binom{j-1}{i-1}$. The nonzero elements of the inverses of L and U are then seen to have (i, j) elements y_{i+1-j} and $(-1)^{i-j}\binom{j-1}{i-1}$. The determinant of A is x_1^n.

For some choices of $x_1, \ldots, x_n$, it is easy to determine the condition numbers, especially with respect to the L_1 norm, of the matrices A generated in this way. Ericksen (1985) suggests that the xs be chosen as

$$x_1 = 2^m \quad \text{for } m \leq 0$$

and

$$x_i = \binom{k}{i-1} \quad \text{for } i = 2, \ldots, n \quad \text{and } k \geq 2,$$

in which case the L_1 condition number of 10×10 matrices will range from about 10^7 to 10^{17} as n ranges from 2 to 20 for $m = 0$ and will range from about 10^{11} to 10^{23} as n ranges from 2 to 20 for $m = -1$.

For testing algorithms for computing eigenvalues, a useful matrix is a *Wilkinson matrix*, which is a symmetric, tridiagonal matrix with 1s on the off-diagonals. For an $n \times n$ Wilkinson matrix, the diagonal elements are

$$\frac{n-1}{2}, \frac{n-3}{2}, \frac{n-5}{2}, \ldots, \frac{n-5}{2}, \frac{n-3}{2}, \frac{n-1}{2}.$$

If n is odd, the diagonal includes 0, otherwise all of the diagonal elements are positive. The 5×5 Wilkinson matrix, for example, is

$$\begin{bmatrix} 2 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 2 \end{bmatrix}.$$

The two largest eigenvalues of a Wilkinson matrix are very nearly equal. Other pairs are likewise almost equal to each other: the third and fourth largest eigenvalues are also close in size, the fifth and sixth largest are likewise, and so on. The largest pair is closest in size, and each smaller pair is less close in size. The Matlab function `wilkinson(n)` generates an $n \times n$ Wilkinson matrix.

Another test matrix available in Matlab is the Rosser test matrix, which is an 8×8 matrix with an eigenvalue of multiplicity 2 and three nearly equal eigenvalues. It is constructed by the Matlab function `rosser`.

A well-known, large, and wide-ranging set of test matrices for computational algorithms for various problems in linear algebra was compiled and described by Gregory and Karney (1969). Higham (1991, 2002) also describes a set of test matrices and provides Matlab programs to generate the matrices.

Another set of test matrices is available through the "Matrix Market", designed and developed by Boisvert, Pozo, and Remington of the U.S. National Institute of Standards and Technology with contributions by various other people. The test matrices can be accessed at

`http://math.nist.gov/MatrixMarket`

The database can be searched by specifying characteristics of the test matrix, such as size, symmetry, and so on. Once a particular matrix is found, its sparsity pattern can be viewed at various levels of detail, and other pertinent data can be reviewed. If the matrix seems to be what the user wants, it can be downloaded. The initial database for the Matrix Market is the approximately 300 problems from the Harwell-Boeing Sparse Matrix Collection.

A set of test datasets for statistical analyses has been developed by the National Institute of Standards and Technology. This set, called "statistical

reference datasets" (StRD), includes test datasets for linear regression, analysis of variance, nonlinear regression, Markov chain Monte Carlo estimation, and univariate summary statistics. It is available at

`http://www.itl.nist.gov/div898/strd/`

12.1.5.2 Assessing the Accuracy of a Computed Result

In real-life applications, the correct solution is not known, and this may also be the case for randomly generated test datasets. If the correct solution is not known, internal consistency tests as discussed in Sect. 11.2.3 may be used to assess the accuracy of the computations in a given problem.

12.1.5.3 Software Reviews

Reviews of available software play an important role in alerting the user to both good software to be used and bad software to be avoided. Software reviews also often have the salutary effect of causing the software producers to improve their products.

Software reviews vary in quality. Some reviews are rather superficial, and reflect mainly the overall impressions of the reviewer. For software described in the peer-reviewed literature, efforts like the ACM's Replicated Computational Results review process can improve the quality of reviews (see below).

12.1.6 Reproducible Research

As the research of statisticians, computer scientists, and applied mathematicians has become ever more computationally-oriented, the reproducibility of the research has become much more of an issue. Research involving only logical reasoning or mathematical manipulations is immediately reproducible by it essential nature. Research involving mathematical manipulations that also include some assumptions about underlying physical phenomena must be validated by observational studies. Experiments or observational studies in the natural sciences, however, by long-standing tradition, must also be reproducible by experimentation or through other observational studies in order to be accepted by other scientists.

Reproducibility of research based on computational results requires access to the original data or to data substantially similar, and access to the entire workflow, including particularly the computational methods. Data may have substantial value, and the owner of the data, in order to protect that value, may impose strict limits on its usage and distribution. Data may also have confidential, possibly personal, information that must be protected. Often the only way to ensure protection is to restrict the distribution of the data. Just because there are impediments to the sharing of data, however, does not mean

that sharing of data or at least of substantially similar data or anonymized data is not necessary.

The actual computations may depend on the computing system, both the hardware and the software. It is an easy matter to include all of the technical specifications in a description of the research. These include version numbers of published software and exact copies of any ad hoc software. In the case of simulation studies, reproducibility requires specification of the random number generator and any settings used. GitHub, mentioned above, is not only a convenient tool for collaborative research, it also provides a good system of version control and a repository for software (and other documents) to be part of the published research.

One of the main issues in ensuring that research is reproducible is the simple housekeeping involved in keeping correct versions of software and documents together. There are various tools for doing this. Two systems that are helpful in integrating R scripts with LaTeX documents are `Sweave` and `knitr`. Xie (2015) describes `knitr` in detail. Stodden et al. (2014) and Gandrud (2015) discuss `Sweave`, `knitr`, and other tools for integrating programs with documents, and also provide general recommendations for ensuring reproducibility of research. The Reproducible R Toolkit (see page 580) is useful for version control.

Another major way in which the quality of research is ensured is through the peer review and editorial process. There is a variety of publication outlets, and in many, this process is not carried out in an ideal fashion. The publication of algorithms and computer programs is often taken rather lightly. One form of publication, for example, is an R package and associated documentation in CRAN, the Comprehensive R Archive Network. This repository is extremely useful, not only because of the number of high-quality packages it contains, but in the maintenance of the system provided by the R Project team. As the various packages are used in different settings, and because of the open-source nature of the software, there is a certain amount of quality control. Software bugs, however, are notoriously difficult to ferret out, and some R packages are less than reliable.

The ACM, through its journals such as *Transactions on Mathematical Software (TOMS)*, publishes software that has at least been through a standard peer-review refereeing process. The refereeing has been enhanced with a "Replicated Computational Results" or "RCR" review process, in which a referee performs a more extensive review that includes execution of the program in various settings. The RCR reviewer writes a report that is accompanied by the name of the reviewer. The paper/software itself receives a special RCR designation in *TOMS*. See Heroux (2015) for a description of the process.

12.2 Software Libraries

There are a number of libraries of subprograms for carrying out common operations in linear algebra and other scientific computations. The libraries vary in several ways: free or with licensing costs or user fees; low-level computational modules or higher-level, more application-oriented programs; specialized or general purpose; and quality, from high to low. Many of these subprograms were originally written in Fortran and later translated into C. Now many of the common libraries are available in other general-purpose programming languages, such as Python. The basic libraries have also been adapted for use in higher-level software systems such as R and Matlab. In the higher-level systems and in Python, libraries are usually called "packages".

In any system, when multiple packages or libraries are installed, there may be multiple versions of some of the components. Management of the separate packages in a single system can be carried out by a package manager such as Conda. Conda is written in Python, and generally oriented toward Python packages but it can be used to manage packages for other software systems, including R. A version of Conda, called Anaconda, includes several Python packages in addition to the package manager. Anaconda and information about it are available at

```
https://www.continuum.io/
```

12.2.1 BLAS

There are several basic computations for vectors and matrices that are very common across a wide range of scientific applications. Computing the dot product of two vectors, for example, is a task that may occur in such diverse areas as fitting a linear model to data or determining the maximum value of a function. While the dot product is relatively simple, the details of how the computations are performed and the order in which they are performed can have effects on both the efficiency and the accuracy; see the discussion beginning on page 487 about the order of summing a list of numbers.

There is a widely-used standard set of routines called "basic linear algebra subprograms" (BLAS) implement many of the standard operations for vectors and matrices. The BLAS represent a very significant step toward software standardization because the definitions of the tasks and the user interface are the same on all computing platforms. The BLAS can be thought of as "software parts" (Rice 1993). They are described in terms of the interface to a general-purpose programming language (an "application programming interface", or API). A software part, similar to an electrical component in an electronic device, conforms to functional specifications, which allow an engineer to construct a larger system using that part, possibly together with other parts. The specific "binding", or name and list of arguments, may be different for different software systems, and certainly the actual coding may

be quite different to take advantage of special features of the hardware or underlying software such as compilers.

The operations performed by the BLAS often cause an input variable to be updated. For example, in a Givens rotation, two input vectors are rotated into two new vectors (see `rot` below). In this case, it is natural and efficient just to replace the input values with the output values. A natural implementation of such an operation is to use an argument that is both input and output. In some programming paradigms, such a "side effect" can be somewhat confusing, but the value of this implementation outweighs the undesirable properties.

There is a consistency of the interface among the BLAS routines. The nature of the arguments and their order in the reference are similar from one routine to the next. The general order of the arguments is:

1. the size or shape of the vector or matrix,
2. the array itself, which may be either input or output,
3. the stride, and
4. other input arguments.

The first and second types of arguments are repeated as necessary for each of the operand arrays and the resultant array.

A BLAS routine is identified by a root character string that indicates the operation, for example, `dot` or `axpy`. The name of the BLAS program module may depend on the programming language. In some languages, the root may be prefixed by `s` to indicate single precision, by `d` to indicate double precision, or by `c` to indicate complex, for example. If the language allows generic function and subroutine references, just the root of the name is used.

The axpy operation we referred to on page 12 multiplies one vector by a constant and then adds another vector $(ax + y)$. The BLAS routine `axpy` performs this operation. The interface is

```
axpy(n, a, x, incx, y, incy)
```

where

`n`,	the number of elements in each vector,
`a`,	the scalar constant,
`x`,	input/output one-dimensional array that contains the elements of the vector x,
`incx`,	the stride in the array `x` that defines the vector,
`y`,	the input/output one-dimensional array that contains the elements of the vector y, and
`incy`,	the stride in the array `y` that defines the vector.

As another example, the routine `rot` to apply a Givens rotation has the interface

```
rot(n, x, incx, y, incy, c, s)
```

where

`n`,	the number of elements in each vector,
`x`,	the input/output one-dimensional array that contains the elements of the vector x,
`incx`,	the stride in the array `x` that defines the vector,
`y`,	the input/output one-dimensional array that contains the elements of the vector y,
`incy`,	the stride in the array `y` that defines the vector,
`c`,	the cosine of the rotation, and
`s`,	the sine of the rotation.

This routine is invoked after `rotg` has been called to determine the cosine and the sine of the rotation (see Exercise 12.5, page 583).

Source programs and additional information about the BLAS can be obtained at

```
http://www.netlib.org/blas/
```

There is a software suite called ATLAS (Automatically Tuned Linear Algebra Software) that provides Fortran and C interfaces to a portable BLAS binding as well as to other software for linear algebra for various processors. Information about the ATLAS software can be obtained at

```
http://math-atlas.sourceforge.net/
```

Intel Math Kernel Library (MKL) is a library of optimized routines that takes advantage of the architecture of Intel processors, or other processors built on the same architecture. The functions include the BLAS, LAPACK (see next section), FFT, and various miscellaneous computations. The BLAS and LAPACK routines use the standard interfaces, so programs using BLAS and LAPACK can simply be re-linked with the MKL. The functions are designed to perform multithreaded operations using all of the available cores in an Intel CPU. There is an R package `RevoUtilsMath` which provides for the incorporation of the Math Kernel Library into R on systems running on Intel or compatible processors.

12.2.2 Level 2 and Level 3 BLAS, LAPACK, and Related Libraries

The level 1 BLAS or BLAS-1, the original set of the BLAS, are for vector operations. They were defined by Lawson et al. (1979). The basic operation is axpy, for vectors and scalars:

$$\alpha x + y.$$

(Here I have adopted the notation commonly used now by the numerical analysts who work on the BLAS.) General operations on a matrix and a vector

or on two matrices can be put together using the BLAS-1. Depending on the architecture of the computer, greater efficiency can be achieved by tighter coupling of the computations involving all elements in a matrix without regard to whether they are in the same row or same column. Dongarra et al. (1988) defined a set of the BLAS, called level 2 or the BLAS-2, for operations involving a matrix and a vector. The basic operation is "gemv" (general matrix-vector),

$$\alpha Ax + \beta y.$$

Later, a set called the level 3 BLAS or the BLAS-3, for operations involving two dense matrices, was defined by Dongarra et al. (1990), and a set of the level 3 BLAS for sparse matrices was proposed by Duff et al. (1997). An updated set of BLAS is described by Blackford et al. (2002). The basic operation is "gemm" (general matrix-matrix),

$$\alpha AB + \beta C.$$

Duff and Vömel (2002) provide a set of Fortran BLAS for sparse matrices.

When work was being done on the BLAS-1 in the 1970s, those lower-level routines were being incorporated into a higher-level set of Fortran routines for matrix eigensystem analysis called EISPACK (Smith et al. 1976) and into a higher-level set of Fortran routines for solutions of linear systems called LINPACK (Dongarra et al. 1979). As work progressed on the BLAS-2 and BLAS-3 in the 1980s and later, a unified set of Fortran routines for both eigenvalue problems and solutions of linear systems was developed, called LAPACK (Anderson et al. 2000). A Fortran 95 version, LAPACK95, is described by Barker et al. (2001). Current information about LAPACK is available at

`http://www.netlib.org/lapack/`

There is a graphical user interface to help the user navigate the LAPACK site and download LAPACK routines.

There are several other libraries that provide functions not only for numerical linear algebra but also for a wide range of scientific computations. Two of the most widely-used proprietary Fortran and C libraries are the IMSL Libraries and the Nag Library. The GNU Scientific Library (GSL) is a widely-used and freely-distributed C library. See Galassi et al. (2002) and the web site

`http://www.gnu.org/gsl/`

Many of the GSL functions, as well as others, have been incorporated into the Python package called `numpy`. For a person with limited resources, Python together with packages such as `numpy` is one of the cheapest and easiest ways to get up and running with numerical computations and graphics on a general-purpose personal computer. The free Anaconda system will quickly install Python along with the packages.

All of these libraries provide large numbers of routines for numerical linear algebra, ranging from very basic computations as provided in the BLAS through complete routines for solving various types of systems of equations and for performing eigenanalysis.

12.2.3 Libraries for High Performance Computing

Because computations for linear algebra are so pervasive in scientific applications, it is important to have very efficient software for carrying out these computations. We have discussed several considerations for software efficiency in previous chapters.

If the matrices in large-scale problems are sparse, it is important to take advantage of that sparsity both in the storage and in all computations. We discussed storage schemes on page 548. For sparse matrices it is necessary to have a scheme for identifying the locations of the nonzeros and for specifying their values. The nature of storage schemes varies from one software package to another, but the most common is index-index-value (see page 550). It is also important to preserve the sparsity during intermediate computations.

Software for performing computations on sparse matrices is specialized to take advantage of the zero entries. Duff, Heroux, and Pozo (2002) discuss special software for sparse matrices.

12.2.3.1 Libraries for Parallel Processing

In parallel processing, the main issues are how to divide up the computations and how to share the results. A widely-used standard for communication among processors is the Message-Passing Interface (MPI) developed in the 1990s. MPI allows for standardized message passing across languages and systems. See Gropp et al. (2014) for a description of the MPI system. Current information is available at

`https://www.open-mpi.org/`

IBM has built the Message Passing Library (MPL) in both Fortran and C, which provides message-passing kernels.

A system such as MPL that implements the MPI paradigm is essentially a library of functions. Another approach to parallel processing is to incorporate language constructs into the programming language itself, thus providing a higher-level facility. OpenMP is a language extension for expressing parallel operations. See Chapman et al. (2007) for a description of OpenMP. Current information is available at

`http://www.openmp.org/`

A standard set of library routines that are more application-oriented, called the BLACS (Basic Linear Algebra Communication Subroutines), provides a portable message-passing interface primarily for linear algebra computations

with a user interface similar to that of the BLAS. A slightly higher-level set of routines, the PBLAS, combine both the data communication and computation into one routine, also with a user interface similar to that of the BLAS. Filippone and Colajanni (2000) provide a set of parallel BLAS for sparse matrices. Their system, called PSBLAS, shares the general design of the PBLAS for dense matrices and the design of the level 3 BLAS for sparse matrices.

ScaLAPACK, described by Blackford et al. (1997b), is a distributed memory version of LAPACK that uses the BLACS and the PBLAS modules. The computations in ScaLAPACK are organized as if performed in a "distributed linear algebra machine" (DLAM), which is constructed by interconnecting BLAS with a BLACS network. The BLAS perform the usual basic computations and the BLACS network exchanges data using primitive message-passing operations. The DLAM can be constructed either with or without a host process. If a host process is present, it would act like a server in receiving a user request, creating the BLACS network, distributing the data, starting the BLAS processes, and collecting the results. ScaLAPACK has routines for LU, Cholesky, and QR decompositions and for computing eigenvalues of a symmetric matrix. The routines are similar to the corresponding routines in LAPACK (Table 12.2). Even the names are similar, for example, in Fortran:

Table 12.2. LAPACK and ScaLAPACK

LAPACK	ScaLAPACK	
`dgetrf`	`pdgetrf`	LU factorization
`dpotrf`	`pdpotrf`	Cholesky factorization
`dgeqrf`	`pdgeqrf`	QR factorization
`dsyevx`	`pdsyevx`	Eigenvalues/vectors of symmetric matrix

Three other packages for numerical linear algebra that deserve mention are Trilinos, Blaze, and PLASMA. Trilinos is a collection of compatible software packages that support parallel linear algebra computations, solution of linear and nonlinear equations and eigensystems of equations and related capabilities. The majority of packages are written in C++ using object-oriented techniques. All packages are self-contained, with the Trilinos top layer providing a common look and feel and infrastructure.

The main Trilinos web site is

`http://software.sandia.gov/trilinos/`

All of the Trilinos packages are available on a range of platforms, especially on high-performance computers.

Blaze is an open-source, high-performance C++ library for dense and sparse matrices. Blaze consists of a number of LAPACK wrapper functions that simplify the use of LAPACK.

The main Blaze web site is

```
http://bitbucket.org/blaze-lib/blaze/
```

There is also an R package `RcppBlaze` which provides for the incorporation of the Blaze software into R using the `Rcpp` package for use of C++ code in R.

The Parallel Linear Algebra for Scalable Multi-core Architectures (PLASMA) system is based on OpenMP multithreading directives. It includes linear systems solvers, mixed precision iterative refinement, matrix norms (fast, multithreaded), a full set of multithreaded BLAS 3, matrix inversion, least squares, and band linear solvers. See Buttari et al. (2009) for a description of the overall design. The functionality of PLASMA continues to increase.

The constructs of an array-oriented language such as R or modern Fortran are helpful in thinking of operations in such a way that they are naturally parallelized. While the addition of arrays in FORTRAN 77 or C is an operation that leads to loops of sequential scalar operations, in modern Fortran it is thought of as a single higher-level operation. How to perform operations in parallel efficiently is still not a natural activity, however. For example, the two modern Fortran statements to add the arrays `a` and `b` and then to add `a` and `c`

```
d = a + b
e = a + c
```

may be less efficient than loops because the array `a` may be accessed twice.

General references that describe parallel computations and software for linear algebra include Nakano (2012), Quinn (2003), and Roosta (2000).

12.2.3.2 Clusters of Computers

The software package PVM, or Parallel Virtual Machine, which was developed at Oak Ridge National Laboratory, the University of Tennessee, and Emory University, provides a set of C functions or Fortran subroutines that allow a heterogeneous collection of Unix or Linux computers to operate smoothly as a multicomputer (see Geist et al. 1994). The development of PVM is ongoing. There is currently a set of bindings to PVM from R, called RPVM.

A cluster of computers is a very cost-effective method for high-performance computing. A standard technology for building a cluster of Unix or Linux computers is called Beowulf (see Gropp, Lusk, and Sterling 2003). A system called Pooch is available for linking Apple computers into clusters (see Dauger and Decyk 2005).

12.2.3.3 Graphical Processing Units

A typical central processing unit (CPU) consists of registers into which operands are brought, manipulated as necessary, and then combined in a

manner simulating the mathematical operation to be performed. The registers themselves may be organized into a small number of "cores", each of which can perform separate arithmetical operations. In the late 1990s, Nvidia extended this type of design to put thousands of cores on a single unit. This was done initially so as to update graphical images, which only requires simple operations on data representing states of pixels. This type of processing unit is called a graphical processing unit or GPU.

Over time, the cores of a GPU were developed further so as to be able to perform more general arithmetic operations. Nvidia integrated GPUs with CPUs in a "compute unified device architecture", or CUDA. CUDA now refers to a programming interface at both a low and a high level to allow Fortran or C programs to utilize an architecture that combines GPUs and CPUs. Various CUDA libraries are available, such as BLAS (cuBLAS) and routines for computations with sparse matrices (cuSPARSE).

Cheng, Grossman, and McKercher (2014) provide an introduction to CUDA programming in C. They discuss matrix multiplication in CUDA on both shared-memory and unshared-memory architectures. In a CUDA implementation, each core within the GPU accumulates one entry of the product matrix.

12.2.4 The IMSL Libraries

The IMSL libraries are available in both Fortran and C versions and in both single and double precisions. These libraries use the BLAS and other software from LAPACK. Sparse matrices are generally stored in the index-index-value scheme, although other storage modes are also implemented.

12.2.4.1 Examples of Use of the IMSL Libraries

There are separate IMSL routines for single and double precisions. The names of the Fortran routines share a common root; the double-precision version has a `D` as its first character, usually just placed in front of the common root. Functions that return a floating-point number but whose mnemonic root begins with an `I` through an `N` have an `A` in front of the mnemonic root for the single-precision version and have a `D` in front of the mnemonic root for the double-precision version. Likewise, the names of the C functions share a common root. The function name is of the form `imsl_f_`*root_name* for single precision and `imsl_d_`*root_name* for double precision.

Consider the problem of solving the system of linear equations

$$\begin{aligned} x_1 + 4x_2 + 7x_3 &= 10, \\ 2x_1 + 5x_2 + 8x_3 &= 11, \\ 3x_1 + 6x_2 + 9x_3 &= 12. \end{aligned}$$

Write the system as $Ax = b$. The coefficient matrix A is real (not necessarily `REAL`) and square. We can use various IMSL subroutines to solve this problem. The two simplest basic routines are `lslrg`/`dlslrg` and `lsarg`/`dlsarg`. Both have the same set of arguments:

`n`,	the problem size.
`A`,	the coefficient matrix.
`lda`,	the leading dimension of `A` (`A` can be defined to be bigger than it actually is in the given problem).
`b`,	the right-hand sides.
`ipath`,	an indicator of whether $Ax = b$ or $A^{\mathrm{t}}x = b$ is to be solved.
`x`,	the solution.

The difference in the two routines is whether or not they do iterative refinement. A program to solve the system using `lsarg` (without iterative refinement) is shown in Fig. 12.2.

```
C  Fortran 77 program
      PARAMETER (ida=3)
      INTEGER   n, ipath
      REAL      A(ida, ida), b(ida), x(ida)
C  Storage is by column;
C  nonblank character in column 6 indicates continuation
      DATA      A/1.0,   2.0,  3.0,
     +            4.0,   5.0,  6.0,
     +            7.0,   8.0,  9.0/
      DATA      b/10.0, 11.0, 12.0/
      n     = 3
      ipath = 1
      CALL lsarg (n, A, lda, b, ipath, x)
      PRINT *, 'The solution is', x
      END
```

Figure 12.2. IMSL Fortran program to solve the system of linear equations

The IMSL C function to solve this problem is `lin_sol_gen`, which is available as *float* `*imsl_f_lin_sol_gen` or *double* `*imsl_d_lin_sol_gen`. The only required arguments for `*imsl_f_lin_sol_gen` are:

int `n`,	the problem size;
float `a[]`,	the coefficient matrix; and
float `b[]`,	the right-hand sides.

Either function will allow the array `a` to be larger than `n`, in which case the number of columns in `a` must be supplied in an optional argument. Other

optional arguments allow the specification of whether $Ax = b$ or $A^{\mathrm{T}}x = b$ is to be solved (corresponding to the argument `ipath` in the Fortran subroutines `lslrg`/`dlslrg` and `lsarg`/`dlsarg`), the storage of the LU factorization, the storage of the inverse, and so on. A program to solve the system is shown in Fig. 12.3. Note the difference between the column orientation of Fortran and the row orientation of C.

```
\*  C program *\
#include <imsl.h>
#include <stdio.h>
main()
{
    int   n = 3;
    float *x;
\* Storage is by row;
   statements are delimited by ';',
   so statements continue automatically. */
    float a[] = {1.0,  4.0,  7.0,
                 2.0,  5.0,  8.0,
                 3.0,  6.0,  9.0};
    float b[] = {10.0, 11.0, 12.0};
    x = imsl_f_lin_sol_gen (n, a, IMSL_A_COL_DIM, 3, b, 0);
    printf ("The solution is %10.4f%10.4f%10.4f\n",
                             x[0], x[1], x[2]);
}
```

Figure 12.3. IMSL C program to solve the system of linear equations

The argument `IMSL_A_COL_DIM` is optional, taking the value of `n`, the number of equations, if it is not specified. It is used in Fig. 12.3 only for illustration.

12.3 General Purpose Languages

Fortran, C, and Python are the most commonly used procedural languages for scientific computation. The International Organization for Standardization (ISO), has specified standard definitions of Fortran and C, so users' and developers' investments in these two languages have a certain amount of protection. A standard version of Python is controlled by the nonprofit Python Software Foundation.

The development of accepted standards for computer languages began with the American National Standards Institute (ANSI). Now its international counterpart, ISO, is the main body developing and promoting standards for

computer languages. Whenever ANSI and ISO both have a standard for a given version of a language, the standards are the same.

There are various dialects of both Fortran and C, most of which result from "extensions" provided by writers of compilers. While these extensions may make program development easier and occasionally provide modest enhancements to execution efficiency, a major effect of the extensions is to lock the user into a specific compiler. Because users usually outlive compilers, it is best to eschew the extensions and to program according to the ANSI/ISO standards. Several libraries of program modules for numerical linear algebra are available both in standard Fortran and in standard C.

C began as a low-level language that provided many of the capabilities of a higher-level language together with more direct access to the operating system. It still lacks some of the facilities that are very useful in scientific computation.

C++ is an object-oriented programming language built on C. The object-oriented features make it much more useful in computing with vectors and matrices or other arrays and more complicated data structures. Class libraries can be built in C++ to provide capabilities similar to those available in Fortran. There are also ANSI (or ISO) standard versions of C++.

An advantage of C over Fortran is that it provides for easier communication between program units, so it is often used when larger program systems are being put together. Other advantages of C are the existence of widely-available, inexpensive compilers, and the fact that it is widely taught as a programming language in beginning courses in computer science.

Fortran has evolved over many years of use by scientists and engineers. Both ANSI and ISO have specified standard definitions of various versions of Fortran. A version called FORTRAN was defined in 1977 (see ANSI 1978). We refer to this version along with a modest number of extensions as FORTRAN 77. If we mean to exclude any extensions or modifications, we refer to it as ANSI FORTRAN 77. (Although the formal name of that version of the language is written in upper case, most people and I generally write it in lower case except for the first letter.) A new standard (not a replacement standard) was adopted in 1990 by ANSI, at the insistence of ISO. This standard language is called ANSI Fortran 90 (written in lower case except for the first letter) or ISO Fortran 90 (see ANSI 1992). It has a number of features that extend its usefulness, especially in numerical linear algebra. There have been a few revisions of Fortran 90 in the past several years. There are only small differences between Fortran 90 and subsequent versions, which are called Fortran 95, Fortran 2000, Fortran 2003, Fortran 2008, and Fortran 2015. Fortran 2008 introduced the concept of multiple images for parallel processing. Each image can be thought of as a separate program executing in its own environment. Data are shared by a special construct, called a *coarray*.

Most of the features I discuss are in all versions after Fortran 95, which I will generally refer to just as "Fortran" or, possibly, "modern Fortran".

Modern Fortran provides additional facilities for working directly with arrays. For example, to add matrices A and B we can write the Fortran expression `A+B`.

Compilers for Fortran are often more expensive and less widely available than compilers for C/C++. An open-source free compiler for Fortran 95 is available at

`http://www.g95.org/`

This compiler also implements some of the newer features, such as coarrays, which were introduced in the later versions of Fortran 2003 and Fortran 2008.

A disadvantage of Fortran compared with C/C++ is that fewer people outside of the numerical computing community know the language.

12.3.1 Programming Considerations

Both users and developers of software need to be aware of a number of programming details.

Sometimes within the execution of an iterative algorithm it is necessary to perform some operation outside of the basic algorithm itself. A common example of this is in an online algorithm, in which more data must be brought in between the operations of the online algorithm. A simple example of this is the online computation of a correlation matrix using an algorithm similar to equations (10.8) on page 504. When the first observation is passed to the program doing the computations, that program must be told that this is the first observation (or, more generally, the first n_1 observations). Then, for each subsequent observation (or set of observations), the program must be told that these are intermediate observations. Finally, when the last observation (or set of observations, or even a null set of observations) is passed to the computational program, the program must be told that these are the last observations, and wrap-up computations must be performed (computing covariances and correlations from sums of squares). Between the first and last invocations of the computational program, it may preserve intermediate results that are not passed back to the calling program. In this simple example, the communication is one-way, from calling routine to called routine.

12.3.1.1 Reverse Communication in Iterative Algorithms

In more complicated cases using an iterative algorithm, the computational routine may need more general input or auxiliary computations, and hence there may be two-way communication between the calling routine and the called routine. A two-way communication between the routines is sometimes called reverse communication. An example is the repetition of a preconditioning step in a routine using a conjugate gradient method; as the computations proceed, the computational routine may detect a need for rescaling and so return to a calling routine to perform those services. Barrett et al. (1994) and

Dongarra and Eijkhout (2000) describe a variety of uses of reverse communication in software for numerical linear algebra.

12.3.1.2 Computational Efficiency

Two seemingly trivial things can have major effects on computational efficiency. One is movement of data between the computer's memory and the computational units. How quickly this movement occurs depends, among other things, on the organization of the data in the computer. Multiple elements of an array can be retrieved from memory more quickly if they are in contiguous memory locations. (Location in computer memory does not necessarily refer to a physical place; in fact, memory is often divided into banks, and adjacent "locations" are in alternate banks. Memory is organized to optimize access.) The main reason that storage of data in contiguous memory locations affects efficiency involves the different levels of computer memory. A computer often has three general levels of randomly accessible memory, ranging from "cache" memory, which is very fast, to "disk" memory, which is relatively slower. When data are used in computations, they may be moved in blocks, or pages, from contiguous locations in one level of memory to a higher level. This allows faster subsequent access to other data in the same page. When one block of data is moved into the higher level of memory, another block is moved out. The movement of data (or program segments, which are also data) from one level of memory to another is called "paging".

In Fortran, a column of a matrix occupies contiguous locations, so when paging occurs, elements in the same column are moved. Hence, a column of a matrix can often be operated on more quickly in Fortran than a row of a matrix. In C, a row can be operated on more quickly for similar reasons.

Some computers have array processors that provide basic arithmetic operations for vectors. The processing units are called vector registers and typically hold 128 or 256 full-precision floating-point numbers (see Sect. 10.1). For software to achieve high levels of efficiency, computations must be organized to match the length of the vector processors as often as possible.

Another thing that affects the performance of software is the execution of loops. In the simple loop

```
do i = 1, n
   sx(i) = sin(x(i))
end do
```

it may appear that the only computing is just the evaluation of the sine of the elements in the vector `x`. In fact, a nonnegligible amount of time may be spent in keeping track of the loop index and in accessing memory. A compiler on a vector computer may organize the computations so that they are done in groups corresponding to the length of the vector registers. On a computer that does not have vector processors, a technique called "unrolling do-loops"

is sometimes used. For the code segment above, unrolling the do-loop to a depth of 7, for example, would yield the following code:

```
do i = 1, n, 7
   sx(i)   = sin(x(i))
   sx(i+1) = sin(x(i+1))
   sx(i+2) = sin(x(i+2))
   sx(i+3) = sin(x(i+3))
   sx(i+4) = sin(x(i+4))
   sx(i+5) = sin(x(i+5))
   sx(i+6) = sin(x(i+6))
end do
```

plus a short loop for any additional elements in `x` beyond $7\lfloor n/7 \rfloor$. Obviously, this kind of programming effort is warranted only when n is large and when the code segment is expected to be executed many times. The extra programming is definitely worthwhile for programs that are to be widely distributed and used, such as the BLAS.

For matrices with special patterns, the storage mode (see page 548) can have a major effect on the computational efficiency, both in memory access and in the number of computations that can be avoided.

12.3.2 Modern Fortran

Fortran began as a simple language to translate common mathematical expressions into computer instructions. This heritage ensures that much of the language "looks like" what most people write in standard mathematics. Early versions of Fortran had a lot of "computerese" restrictions, however (parts of statements had to be in certain columns, and so on). Statement grouping was done in an old-fashioned way, using numbers. More seriously, many things that we might want to do were very difficult.

Fortran has evolved, and modern Fortran is now one of the most useful general-purpose compiler languages for numerical computations. Some books that describe modern Fortran are Clerman and Spector (2012), Hanson and Hopkins (2013), Lemmon and Schafer (2005), Markus (2012), and Metcalf, Reid, and Cohen (2011).

For the scientific programmer, one of the most useful features of modern Fortran is the provision of primitive constructs for vectors and matrices. Whereas all of the FORTRAN 77 intrinsics are scalar-valued functions, Fortran 95 provides intrinsic array-valued functions. For example, if `A` and `B` represent matrices conformable for addition, the statement `D = A+B` performs the operation; if they are conformable for multiplication, the statement

```
C = MATMUL(A, B)
```

yields the Cayley product in `C`. The `MATMUL` function also allows multiplication of vectors and matrices.

Indexing of arrays starts at 1 by default (any starting value can be specified, however), and storage is column-major.

Space must be allocated for arrays in modern Fortran, but this can be done at run time. An array can be initialized either in the statement allocating the space or in a regular assignment statement. A vector can be initialized by listing the elements between "(/" and "/)". This list can be generated in various ways. The `RESHAPE` function can be used to initialize matrices.

For example, a modern Fortran statement to declare that the variable `A` is to be used as a 3×4 array and to allocate the necessary space is

```
REAL, DIMENSION(3,4) :: A
```

A Fortran statement to initialize `A` with the matrix

$$\begin{bmatrix} 1 & 4 & 7 & 10 \\ 2 & 5 & 8 & 11 \\ 3 & 6 & 9 & 12 \end{bmatrix}$$

is

```
A = RESHAPE( (/ 1., 2., 3., &
                4., 5., 6., &
                7., 8., 9., &
               10.,11.,12./), &
             (/3,4/) )
```

Fortran has an intuitive syntax for referencing subarrays, shown in Table 12.3. (See also Table 12.1 on page 548.)

Table 12.3. Subarrays in modern Fortran

`A(2:3,1:3)`	The 2×3 submatrix in rows 2 and 3 and columns 1 to 3 of `A`
`A(:,1:4:2)`	Refers to the submatrix with all three rows and the first and third columns of `A`
`A(:,4)`	Refers to the column vector that is the fourth column of `A`

Notice that because the indexing starts with 1 (instead of 0) the correspondence between the computer objects and the mathematical objects is a natural one. The subarrays can be used directly in functions. For example, if `A` is as shown above, and `B` is the matrix

$$\begin{bmatrix} 1 & 5 \\ 2 & 6 \\ 3 & 7 \\ 4 & 8 \end{bmatrix},$$

the Fortran function reference

```
MATMUL(A(1:2,2:3), B(3:4,:))
```

yields the Cayley product

$$\begin{bmatrix} 4 & 7 \\ 5 & 8 \end{bmatrix} \begin{bmatrix} 3 & 7 \\ 4 & 8 \end{bmatrix}. \tag{12.9}$$

Fortran also contains some of the constructs, such as `FORALL`, that have evolved to support parallel processing.

More extensive later revisions (Fortran 2000 and subsequent versions) include such features as exception handling, better interoperability with C, allocatable components, parameterized derived types, object-oriented programming, and coarrays.

A good index of freely-available Fortran software (and also C and C++ software) is

```
http://www.netlib.org/utk/people/JackDongarra/la-sw.html
```

12.3.3 C and C++

C is a computer language ideally suited to computer systems operation and management. It is stable and widely used, meaning that an investment in C software is likely to continue being worthwhile. There are many libraries of C programs for a wide range of applications.

Indexes of arrays in C begin at 0. (That is the way that positions in hardware registers are normally counted, but not the way we usually index mathematical entities.) Two-dimensional arrays in C are row-major, and in general, the indexes in multi-dimensional arrays as a mapping to computer memory vary fastest from right to left. A programmer mixing Fortran code with C code must be constantly aware of these two properties.

C++ is an object-oriented dialect of C. In an object-oriented language, it is useful to define classes corresponding to matrices and vectors. Operators and/or functions corresponding to the usual operations in linear algebra can be defined so as to allow use of simple expressions to perform these operations.

A class library in C++ can be defined in such a way that the computer code corresponds more closely to mathematical code. The indexes to the arrays can be defined to start at 1, and the double index of a matrix can be written within a single pair of parentheses. For example, in a C++ class defined for use in scientific computations, the $(10, 10)$ element of the matrix A (that is, $a_{10,10}$) can be referenced in a natural way as

```
A(10,10)
```

instead of as

```
A[9][9]
```

as it would be in ordinary C. Many computer engineers prefer the latter notation, however.

There are various C++ class libraries or templates for matrix and vector computations. One is the Armadillo C++ Matrix Library, which provides a syntax similar to that of Matlab.

The Template Numerical Toolkit

`http://math.nist.gov/tnt/`

and the Matrix Template Library

`http://www.osl.iu.edu/research/mtl/`

are templates based on the design approach of the C++ Standard Template Library

`http://www.sgi.com/tech/stl/`

Use of a C++ class library for linear algebra computations may carry a computational overhead that is unacceptable for large arrays. Both the Template Numerical Toolkit and the Matrix Template Library are very efficient computationally, however (see Siek and Lumsdaine 2000). Eubank and Kupresanin (2012) discuss many computational details relating to the usage of C++ in numerical linear algebra.

12.3.4 Python

Python, which is an interpretive system, is one of the most popular languages. It is currently the most commonly-used language in academic courses to introduce students to programming. There are several free Python interpreters for a wide range of platforms. There is an active development community that has produced a large number of open-source Python functions for scientific applications. While all language systems, such as Fortran and C, undergo revisions from time to time, the rate and nature of the revisions to Python have exceeded those of other systems. Some of these changes have been disruptive because of lack of backward compatibility. Many changes are useful enhancements, of course. In a recent update, a Cayley multiplication operator, "`@`" (of all symbols!), was introduced.

For persons who program in other languages, the many differences in symbols and in syntax are annoying and induce programming errors.

Python has a number of array constructs, some that act similarly to vectors, some that are like sets, and so on. The native index for all of them begins at 0.

One of the most useful freely-distributed Python libraries is `numpy`, which supports a wide range of functionality including general numerical linear algebra, statistical applications, optimization, solutions of differential equations, and so on. The free Anaconda system will quickly install Python along with the packages. (See page 555.)

There is also a set of Python wrappers for the Armadillo C++ Matrix Library.

12.4 Interactive Systems for Array Manipulation

Many of the computations for linear algebra are implemented as simple operators on vectors and matrices in some interactive systems. Some of the more common interactive systems that provide for direct array manipulation are Octave or Matlab, R, SAS IML, APL, Lisp-Stat, Gauss, Python, IDL, and PV-Wave. There is no need to allocate space for the arrays in these systems as there is for arrays in Fortran and C.

Occasionally we need to operate on vectors or matrices whose elements are variables. Software for symbolic manipulation, such as Maple or Mathematica, can perform vector/matrix operations on variables. See Exercise 12.11 on page 585.

12.4.1 R

The software system called S was developed at Bell Laboratories, primarily by John Chambers, in the mid-1970s. S is both a data analysis system and an object-oriented programming language. In the mid-1990s, Robert Gentleman and Ross Ihaka produced an open-source and freely available system called R that provided generally the same functionality in the same language as S. This system has become one of the most widely-used statistical software packages. Its development and maintenance has been subsumed by a group of statisticians and computer scientists called the R Project Team. The system, along with various associated packages, is available at

`http://www.r-project.org/`

There are graphical interfaces for installation and maintenance of R that interact well with the operating system, whether Unix/Linus, Microsoft Windows, or MacOS.

In this section I discuss some of the salient properties of R. I describe some features at a high-level, and I also mention some more obtuse properties. The purpose is neither to be a tutorial nor to be a general reference for the language. Venables and Ripley (2003) is one of several good books to use for learning R. Wickham (2015) and Chambers (2016) provide many more details and general insights to go much deeper in understanding and using R.

12.4.1.1 General Properties

The basic numeric object in R is a vector. The most important R entity is the function. In R, all actions are "functions", and R has an extensive set of

functions. Many functions are provided through packages that, although not part of the core R, can be easily installed.

R statements are line-oriented but continue from one line to another until a syntactically complete statement is terminated by an end-of-line.

Assignment is made by "`<-`" or by "`=`".

A comment statement in R begins with a pound sign, "`#`".

R has a number of built-in object classes, and the user can define additional classes that determine how generic functions operate on the objects.

R has functions for printing, but if a statement consists of just the name of an object, the object is printed to the standard output device (which is likely to be the monitor).

One of the most important and useful properties of R is the ways it can handle missing data or values that do not exist as ordinary numbers or characters, such as ∞ or $0/0$. (See Sect. 9.5.6, beginning on page 437, and pages 464 and 475.) Often when data contain missing values, we wish to ignore those values and process all of the valid data. Many functions in R provide a standard way of requesting that this be done, by means of the logical argument `na.rm`. If `na.rm` is true, then the function ignores the fact that some data may be missing, and performs its operations only on the valid data.

12.4.1.2 Documentation

There are a number of books and tutorials on R, and a vast array of sources of information on the internet. R also has an integrated help system. Documentation for R functions is generally stored in the form of `man` pages and can be accessed through the R `help` function or by the R "?" operator.

12.4.1.3 Basic Operations with Vectors and Matrices and for Subarrays

A list is constructed by the `c` function. A list of scalar numeric values can be treated as a vector without modification. A matrix is constructed from a list of numeric values by the `matrix` function. A matrix can also be constructed by binding vectors as the columns of the matrix (the `cbind` function) or by binding vectors as the rows of the matrix (the `rbind` function).

Indexing of arrays starts at 1, and storage is column-major. Indexes are indicated by "`[ ]`"; for example, `x[1]` refers to the first element of the one-dimensional array `x`. Any set of valid indexes can be specified; for example, `x[c(3,1,3)]` refers to the third, the first, and again the third elements of the one-dimensional array `x`. Negative values can be used to indicate removal of specified elements; for example, `x[c(-1,-3)]` refers to the same one-dimensional array `x` with the first and third elements removed. The order of negative indexes or the repetition of negative indexes has no effect; for example, `x[c(-3,-1,-3)]` is the same as `x[c(-1,-3)]`. Positive and negative values cannot be mixed as indexes.

Cayley multiplication is indicated by the symbol "`%*%`". Most operators with array operands are applied elementwise; for example, the symbol "`*`" indicates the Hadamard product of two matrices. The expression

```
A %*% B
```

indicates the Cayley product of the matrices, where the number of columns of `A` must be the same as the number of rows of `B`. The expression

```
A * B
```

indicates the Hadamard product of the matrices, where the number of rows and columns of `A` must be the same as the number of rows and columns of `B`.

The transpose of a vector or matrix is obtained by using the function "`t`":

```
t(A)
```

Although R stores matrices by columns and converts lists into matrices in column order by default, a matrix can be constructed by rows, and doing so often results in more readable code:

```
A <- matrix(c( 1, 4, 7, 10,
               2, 5, 8, 11,
               3, 6, 9, 12),
            nrow=3, byrow=T)
```

To the extent that R distinguishes between row vectors and column vectors, a vector is considered to be a column vector. In many cases, however, it does not distinguish. For example, if

```
x <- c(1,2)
y <- c(1,2)
```

the expression `x %*% y` is the dot product; that is,

```
x %*% y
```

is the same as

```
t(x) %*% y
```

The transpose operator is not required in this case.

The outer product, however, requires either explicit transposition or use of a special binary operator. The outer product is formed by `x %*% t(y)` or by using the special outer product operator `%o%`; thus,

```
x %o% y
```

is the same as

```
x %*% t(y)
```

There is also a useful function, `outer`, that allows more general combinations of the elements of two vectors. For example, if `func` is a scalar function of two scalar variables, `outer(x,y,FUN=func)` forms a matrix with the rows corresponding to `x` and the columns corresponding to `y`, and whose $(ij)^{\text{th}}$ element corresponds to `func(x[i],y[j])`. Strings can be used as the argument `FUN`; thus,

```
outer(x,y,FUN="*")
```

is also the same as

```
x %o% y
```

In the expressions

```
y <- a %*% x
```

and

```
y <- x[c(1,2,3)] %*% a
```

the vector is interpreted as a row or column as appropriate for the multiplication to be defined.

Like many other software systems for array manipulation, R usually does not distinguish between scalars and arrays of size 1. For example, if

```
x <- c(1,2)
y <- c(1,2)
z <- c(1,2,3)
```

the expression `x %*% y %*% z` yields the same value as `5*z` because the expression `x %*% y %*% z` is interpreted as `(x %*% y) %*% z` and `(x %*% y)` is a scalar. The expression `x %*% (y %*% z)` is invalid because `y` and `z` are not conformable for multiplication. In the case of one-dimensional and two-dimensional arrays, R sometimes allows the user to treat them in a general fashion, and sometimes makes a hard distinction. For example, the function `length` returns the number of elements in an array, whether it is one-dimensional or two-dimensional. The function `dim`, however, returns the numbers of rows and columns in a two-dimensional array, but returns `NULL` for a one-dimensional array. See page 576 for some additional comments about one-dimensional and two-dimensional arrays in R.

Examples of subarray references in R are shown in Table 12.4. Compare these with the Fortran references shown in Table 12.3. In R, a missing index indicates that the entire corresponding dimension is to be used. Groups of indices can be formed by the `c` function or the `seq` function, which is similar to the `i:j:k` notation of Fortran.

The subarrays can be used directly in expressions. For example, the expression

```
A[c(1,2),c(2,3)] %*% B[c(3,4),]
```

Table 12.4. Subarrays in R

`A[c(2,3),c(1,3)]`	The 2×3 submatrix in rows 2 and 3 and columns 1 to 3 of `A`
`A[,seq(1,4,2)]`	The submatrix with all 3 rows and the 1^{st} and 3^{rd} columns of `A`
`A[,4]`	The column vector that is the 4^{th} column of `A`

yields the product

$$\begin{bmatrix} 4 & 7 \\ 5 & 8 \end{bmatrix} \begin{bmatrix} 3 & 7 \\ 4 & 8 \end{bmatrix}$$

as on page 570.

The basic atomic numeric type in R is a vector, and a $1 \times n$ matrix or an $n \times 1$ matrix may be cast as vectors (that is, one-dimensional arrays). This is often exactly what we would want. There are, however, cases where this casting is disastrous, for example if we use the `dim` function, as shown in Fig. 12.4.

```
> A <- matrix(c(1,2,3,4,5,6), nrow=3)
> dim(A)
[1] 3 2
> b <- A[,2]
> dim(b)
NULL
> length(b)
[1] 3
```

Figure 12.4. Downcasting in R

To prevent this casting, we can use the `drop` keyword in the subsetting operator, as in Fig. 12.5.

```
> C <- A[,2, drop=FALSE]
> dim(C)
[1] 3 1
```

Figure 12.5. Preserving the class in R

12.4.1.4 R Functions for Numerical Linear Algebra

R has functions for many of the basic operations on vectors and matrices. Some of the R functions are shown in Table 12.5.

Table 12.5. Some R functions for vector/matrix computations

`norm`	Matrix norm.
	The L_1, L_2, L_∞, and Frobenius norms are available.
`t`	Transpose of a matrix or preparation of a vector for multiplication.
`diag`	Principal diagonal of a matrix.
`tr {psych}`	Trace of a square matrix.
`vecnorm`	Vector L_p norm.
`det`	Determinant.
`rcond.Matrix`	Matrix condition number.
`lu`	LU decomposition.
`qr`	QR decomposition.
`chol`	Cholesky factorization.
`svd`	Singular value decomposition.
`eigen`	Eigenvalues and eigenvectors.
`as.vector`	vec(·).
`solve.Matrix`	Solve system of linear equations,
	or compute matrix inverse or pseudoinverse.
`lsfit`	Ordinary or weighted least squares fit.
`l1fit`	Least absolute values fit.
`nnls.fit`	Nonnegative least squares fit.

12.4.1.5 Other Mathematical Objects and Operations

In addition to numerical computations, there are other operations we may wish to perform. R provides facilities for many such operations. One of the most common non-computational operations is sorting. R does sorting very efficiently. A useful R function is `is.unsorted`, which checks if an object is sorted without sorting it.

R does not have an object class of "set" (although we could create one). R, however, does provide functions for the standard set operations such as union and intersection, and the logical operators of set equality and inclusion, as illustrated in Fig. 12.6.

The R function `unique` is useful for removing duplicate elements in an R object.

12.4.1.6 Extensibility

On of the most important features of R is its extensibility. Operations in R are performed by a *function* by invoking the function by its name. New functions

```
> S1 <- c(1,2,3,2)
> S2 <- c(3,2,1)
> S3 <- c(4,3,2,1)
> setequal(S1,S2)
[1] TRUE
> union(S1,S3)
[1] 1 2 3 4
> union(S1,S1)
[1] 1 2 3
> intersect(S1,S3)
[1] 1 2 3
> intersect(S1,S1)
[1] 1 2 3
> 1 %in% S1
[1] TRUE
> 5 %in% S1
[1] FALSE
```

Figure 12.6. Set functions in R: `setequal`, `union`, `intersect`, `%in%`

can be added to an R session with ease. (Multiple functions may have the same name, but only one is active at a time; thus, the function name is unique.)

A simple way a user can extend R is to write a function, using the R function constructor called `function`:

```
myFun <- function( ...){
   ...
}
```

In a simple case, the computations of the function can be expressed in ordinary R statements. In other cases, the computations can be expressed in a compiler language such as Fortran or C, compiled, and then linked into R. In either case, the user's function is invoked by its name, just as if it were part of the original R system.

12.4.1.7 Packages

A group of functions written either in R or in a compiler language can be packaged together in a "library" or "package", and then all of the functions, together with associated documentation, can be loaded into R just by loading the package.

Previously developed packages can be installed with the local system, and then can be loaded in an R session. These packages can be stored in the Comprehensive R Archive Network (CRAN) and an R system program can install them on a local system.

The R system generally consists of a set of packages, a "`base`" package, a "`stats`" package, and so on. When the R program is invoked, a standard set

of packages are loaded. Beyond those, the user can load any number of other packages that have been installed on the local system.

Within an R session, the available functions include all of those in the loaded packages. Sometimes, there may be name conflicts (two functions with the same name). The conflicts are resolved in the order of the loading of the packages; last-in gets preference. Reference to a function name often includes the package that includes the function; for example, "`tr {psych}`" refers to the `tr` function in the `psych` package. (This notation is not used in the R statement itself; the R statement invoking `tr` brings in the currently active version of `tr`.)

Documentation for a new package can be developed (using the R package `roxygen` or by other methods) and when the new package is loaded, the documentation becomes available to the user. This package, which is available on CRAN, can be described as a documentation *system*. It is also available at

`http://roxygen.org/`

There are very many R packages available on CRAN. Some packages address a fairly narrow area of application, but some provide a wide range of useful functions. Two notable ones of the latter type are `MASS`, developed primarily by Bill Venables and Brian Ripley, and `Hmisc`, developed primarily by Frank Harrell.

A problem with many of the packages at CRAN is that some of them were not written, developed, or tested by people with knowledge and skill in software development. Some of these packages are not of high quality. Another problem with packages on CRAN is that they may change from time to time, so that computed results may not be easily replicated.

12.4.1.8 Sharable Libraries and Rcpp

The functionality of R can also be extended by linking functions written in general-purpose compiled languages into R. The general way this is done is by building a sharable library (or "dynamic" library) in the language and then loading it into R using the R function `dyn.load`. (The sharable library can be built within the R script by use of `R CMD`.) For Fortran and C there are R functions, `.Fortran` and `.C`, to access any program unit in the shared library after it is loaded. The arguments of these functions are the program unit name followed by the names of the program unit's arguments set to the appropriately-cast R variables. For example, if a Fortran subroutine is named `myFun` and it has a double precision argument `x` and an integer argument `n`, and it has been compiled and linked into a sharable library called `myLib.so`, it could be invoked in R by the statements

```
dyn.load("myLib.so")
   ...  initialize relevant R variables, say xr and nr
ans <- .Fortran("myFun", x=as.double(xr), n=as.integer(nr))
```

There is an R package, `Rcpp`, that facilitates interoperability between C++ and R. This package allows the R user to write C++ code directly in the R script, so it is much simpler than the sharable library approach described above. This package can be used to produce other R packages written in C++ or other compiler languages. See Eddelbuettel (2013), and also Chambers (2016) and Wickham (2015) for discussion and examples of the use of `Rcpp`. There is also an R package, `RcppArmadillo`, that provides access to the functions in the Armadillo C++ Matrix Library.

Eubank and Kupresanin (2012) discuss use of C++ functions in R and various programming issues for both C++ and R.

There is a Python package, `RPy`, that allows access of the R system from Python.

12.4.1.9 Other Versions of R and Other Distributions

In addition to the standard distribution versions of R available from CRAN, there are various add-ons and enhancements, some also available from CRAN. One of the most notable of these is RStudio, which is an integrated R development system including a sophisticated program editor.

Spotfire S+® is an enhancement of S, originally developed as S-PLUS® by StatSci, Inc. It is now distributed and maintained by TIBCO Software Inc. The enhancements include graphical interfaces with menus for common analyses, more statistical analysis functionality, and support.

Microsoft R Open (MRO), formerly known as Revolution R Open (RRO), is an enhanced distribution of R from Microsoft Corporation. An important feature of Microsoft R Open is the use of the Intel Math Kernel Library (MKL) to allow multiple threads if the CPU follows the Intel multi-core architecture. This can result in significant speedup of standard computations for linear algebra.

Another useful feature of Microsoft R Open is the Reproducible R Toolkit, which can ensure that the same version of all R packages are used consistently. This is done by means of a CRAN repository snapshot, so that even if some packages are changed, the same computations of an R program on a given system at a given time can be performed again on a different system at a different time. A related component of the Reproducible R Toolkit is the `checkpoint` package, which allows the user to go back and forth to retrieve the exact versions of R packages. The user can specify the version of a package directly in the R code.

12.4.2 MATLAB and Octave

MATLAB®, or Matlab®, is a proprietary software package distributed by The Mathworks, Inc. It is built on an interactive, interpretive expression language, and is one of the most widely-used computer systems supporting numerical linear algebra and other scientific computations

Octave is a freely-available package that provides essentially the same core functionality in the same language as Matlab. The graphical interfaces for Octave are more primitive than those for Matlab and do not interact as seamlessly with the operating system. Most of the discussion in this section applies to Octave as well as to Matlab.

The basic object in Matlab and Octave is a rectangular array of numbers (possibly complex). Scalars (even indices) are 1×1 matrices; equivalently, a 1×1 matrix can be treated as a scalar. A vector is a matrix with one column.

Statements in Matlab are line-oriented. A statement is assumed to end at the end of the line, unless the last three characters on the line are periods (...) or a matrix is being initialized. If an assignment statement in Matlab is not terminated with a semicolon, the matrix on the left-hand side of the assignment is printed. If a statement consists only of the name of a matrix, the object is printed to the standard output device (which is likely to be the monitor).

The indexing of arrays in Matlab starts with 1, and indexes are indicated by "( )".

A matrix is initialized in Matlab by listing the elements row-wise within brackets and with a semicolon marking the end of a row. (Matlab also has a `reshape` function similar to that of Fortran that treats the matrix in a column-major fashion.)

Matlab has a number of functions to operate on vectors and matrices to perform such operations as various matrix factorizations, computations of eigenvalues and eigenvectors, solution of linear systems, and so on. Many operations are performed by operators of the language. In general, the operators in the Matlab language refer to the common vector/matrix operations. For example, the usual multiplication symbol, "`*`", means Cayley multiplication when the operands are matrices. The meaning of an operator can often be changed to become the corresponding element-by-element operation by preceding the operator with a period; for example, the symbol "`.*`" indicates the Hadamard product of two matrices. The expression

```
A * B
```

indicates the Cayley product of the matrices, where the number of columns of `A` must be the same as the number of rows of `B`; and the expression

```
A .* B
```

indicates the Hadamard product of the matrices, where the number of rows and columns of `A` must be the same as the number of rows and columns of `B`. The transpose of a vector or matrix is obtained by using a postfix operator "'", which is the same ASCII character as the apostrophe:

```
A'
```

Matlab has special operators "\" and "/" for solving linear systems or for multiplying one matrix by the inverse of another. The operator "/" means

"divide", as in $A/B = AB^{-1}$, while "\" means "divide from the left", as in $A\backslash B = A^{-1}B$.

While the statement

```
A\B
```

refers to a quantity that has the same value as the quantity indicated by

```
inv(A)*B
```

the computations performed are different (and, hence, the values produced may be different). The second expression is evaluated by performing the two operations indicated: `A` is inverted, and the inverse is then used as the left factor in matrix or matrix/vector multiplication. The first expression, `A\B`, indicates that the appropriate computations to evaluate x in $Ax = b$ should be performed to evaluate the expression. (Here, x and b may be matrices or vectors.) Another difference between the two expressions is that `inv(A)` requires `A` to be square and algorithmically nonsingular, whereas `A\B` produces a value that simulates $A^{-}b$.

Matlab distinguishes between row vectors and column vectors. A row vector is a matrix whose first dimension is 1, and a column vector is a matrix whose second dimension is 1. In either case, an element of the vector is referenced by a single index.

Subarrays in Matlab are defined in much the same way as in modern Fortran. The subarrays can be used directly in expressions. For example, the expression

```
A(1:2,2:3) * B(3:4,:)
```

yields the product

$$\begin{bmatrix} 4 & 7 \\ 5 & 8 \end{bmatrix} \begin{bmatrix} 3 & 7 \\ 4 & 8 \end{bmatrix}$$

as on page 570. There is one major difference in how Matlab and Fortran denote sequences with a stride greater than 1, however. The upper limit and the stride are reversed in the triplet. For example, if `x` contains the elements 1,2,3,4,5, in Matlab, `x(1:2:5)` is the list 1,3,5. The same subarray in Fortran is obtained by `x(1:5:2)`.

There are a number of books on Matlab, including, for example, Attaway (2016). Many books on numerical linear algebra use the Matlab language in the expositions. For example, the widely-used book by Coleman and Van Loan (1988), while not specifically on Matlab, shows how to perform matrix computations in Matlab.

Exercises

12.1. Write a function in R to find the square root of a symmetric nonnegative definite matrix (see page 160). Make sure that your function will

determine the square root of a nonnegative 1×1 matrix. Also, write your function so that if the matrix is not square or is not nonnegative definite, it prints an appropriate message and returns an appropriate value.

12.2. Write a recursive function in Fortran, C, or C++ to multiply two square matrices using the Strassen algorithm (page 531). Write the function so that it uses an ordinary multiplication method if the size of the matrices is below a threshold that is supplied by the user.

12.3. Set up an account on GitHub, and upload the function you wrote in Exercise 12.2 to your account. Share this function with another person. (This would probably be the instructor if you are using this book as a textbook in a course.)

12.4. There are various ways to evaluate the efficiency of a program: counting operations, checking the "wall time", using a shell level timer, and using a call within the program. In C, the timing routine is `ctime`, and in modern Fortran it is the subroutine `system_clock`. FORTRAN 77 does not have a built-in timing routine, but the IMSL Fortran Library provides one. For this exercise, you are to write six short C programs and six short Fortran programs. The programs in all cases are to initialize an $n \times m$ matrix so that the entries are equal to the column numbers; that is, all elements in the first column are 1s, all in the second column are 2s, etc. The six programs arise from three matrices of different sizes $10{,}000 \times 10{,}000$, $100 \times 1{,}000{,}000$, and $1{,}000{,}000 \times 100$; and from two different ways of nesting the loops: for each size matrix, first nest the row loop within the column loop and then reverse the loops. The number of operations is the same for all programs. For each program, use both a shell level timer (e.g., in Unix or Linux, use `time`) and a timer called from within your program. Make a table of the times:

		10000×10000	100×1000000	1000000×100
Fortran	column-in-row	–	–	–
	row-in-column	–	–	–
C	column-in-row	–	–	–
	row-in-column	–	–	–

12.5. Obtain the BLAS routines `rotg` and `rot` for constructing and applying a Givens rotation. These routines exist in both Fortran and C; they are available in the IMSL Libraries or from *CALGO* (*Collected Algorithms of the ACM*; see the Bibliography).

a) Using these two routines, apply a Givens rotation to the matrix used in Exercise 5.9 in Chap. 5,

$$A = \begin{bmatrix} 3 & 5 & 6 \\ 6 & 1 & 2 \\ 8 & 6 & 7 \\ 2 & 3 & 1 \end{bmatrix},$$

so that the second column becomes $(5, \tilde{a}_{22}, 6, 0)$.

b) Write a routine in Fortran or C that accepts as input a matrix and its dimensions and uses the BLAS routines `rotg` and `rot` to produce its QR decomposition. There are several design issues you should address: how the output is returned (for purposes of this exercise, just return two arrays or pointers to the arrays in full storage mode), how to handle nonfull rank matrices (for this exercise, assume that the matrix is of full rank, so return an error message in this case), how to handle other input errors (what do you do if the user inputs a negative number for a dimension?), and others.

12.6. Using the BLAS routines `rotg` and `rot` for constructing and applying a Givens rotation and the program you wrote in Exercise 12.5, write a Fortran or C routine that accepts a simple symmetric matrix and computes its eigenvalues using the mobile Jacobi scheme. The outer loop of your routine consists of the steps shown on page 318, and the multiple actions of each of those steps can be implemented in a loop in serial mode. The importance of this algorithm, however, is realized when the actions in the individual steps on page 318 are performed in parallel.

12.7. Sparse matrices.

a) Represent the order 10 identity matrix I_{100} using the index-index-value notation. (You may want to use R or the IMSL libraries to ensure the correctness of the representation.)

b) Represent the 10×10 type 2 tridiagonal matrix of equation (8.92) on page 385 using the index-index-value notation.

12.8. a) Show that the Ericksen matrix (expression (12.7)) has the inverse shown (expression (12.8)).

b) Show that the Ericksen matrix has determinant x_1^n.

c) For $m = -1$ and $k = 2$, form a 10×10 Ericksen matrix using the suggested values for the xs. (Recall the condition on the binomial coefficients.)
Compute the inverse, the determinant, and the L_2 condition number.

12.9. Compute the two largest eigenvalues of the 21×21 Wilkinson matrix to 15 digits.

12.10. Develop a class library in C++ for matrix and vector operations. Discuss carefully the issues you consider in designing the class constructors. Design them in such a way that the references

```
xx(1)
YY(1,1)
```

refer to the implied mathematical entities. Design the operators "`+`" and "`*`" so that the references

```
aa + bb
aa * bb
```

will determine whether `a` and `b` are matrices and/or vectors conformable for the implied mathematical operations and, if so, will produce the object corresponding to the implied mathematical entity represented by the expression.
See Eubank and Kupresanin (2012), Appendix D, for a full-blown solution to this exercise.

12.11. Use a symbolic manipulation software package such as Maple to determine the inverse of the matrix:

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}.$$

Determine conditions for which the matrix would be singular. (You can use the `solve()` function in Maple on certain expressions in the symbolic solution you obtained.)

12.12. Consider the 3×3 symmetric Toeplitz matrix with elements a, b, and c; that is, the matrix that looks like this:

$$\begin{bmatrix} a & b & c \\ b & a & b \\ c & b & a \end{bmatrix}.$$

See Exercise 8.12 on page 398.

a) Use a symbolic manipulation software package such as Maple to determine the inverse of this matrix.
See page 385.

b) Determine conditions for which the matrix would be singular.

12.13. a) Incorporate the function you developed in Exercise 12.2 into R. If you used Fortran or C, first build a sharable library. (You generally do this with a compiler directive.) Then use `dyn.load` and `.Fortran` or `.C`. If you used C++ in Exercise 12.2, use `Rcpp`.

b) Now write an R function to multiple two matrices using the Strassen algorithm as in Exercise 12.2.

c) Now compare the efficiency of your R function with the Fortran, C, or C++ function invoked from R and with the standard R multiplication operator, `%*%`. (The `proc.time` function in the R base package is probably the easiest way to do the timing.)

Appendices

A

Notation and Definitions

All notation used in this work is "standard", and I have endeavored to use notation consistently. I have opted for simple notation, which, of course, results in a one-to-many map of notation to object classes. Within a given context, however, the overloaded notation is generally unambiguous.

This appendix is not intended to be a comprehensive listing of definitions. The Index is a more reliable set of pointers to definitions, except for symbols that are not words.

A.1 General Notation

Uppercase italic Latin and Greek letters, such as A, B, E, Λ, etc., are generally used to represent either matrices or random variables. Random variables are usually denoted by letters nearer the end of the Latin alphabet, such X, Y, and Z, and by the Greek letter E. Parameters in models (that is, unobservables in the models), whether or not they are considered to be random variables, are generally represented by lowercase Greek letters. Uppercase Latin and Greek letters are also used to represent cumulative distribution functions. Also, uppercase Latin letters are used to denote sets.

Lowercase Latin and Greek letters are used to represent ordinary scalar or vector variables and functions. **No distinction in the notation is made between scalars and vectors**; thus, β may represent a vector and β_i may represent the i^{th} element of the vector β. In another context, however, β may represent a scalar. All vectors are considered to be column vectors, although we may write a vector as $x = (x_1, x_2, \ldots, x_n)$. Transposition of a vector or a matrix is denoted by the superscript "T".

Uppercase calligraphic Latin letters, such as $\mathcal{D}$, $\mathcal{V}$, and $\mathcal{W}$, are generally used to represent either vector spaces or transforms (functionals).

J.E. Gentle, *Matrix Algebra*, Springer Texts in Statistics,
DOI 10.1007/978-3-319-64867-5

Subscripts generally represent indexes to a larger structure; for example, x_{ij} may represent the $(i,j)^{\text{th}}$ element of a matrix, X. A subscript in parentheses represents an order statistic. A superscript in parentheses represents an iteration; for example, $x_i^{(k)}$ may represent the value of x_i at the k^{th} step of an iterative process.

x_i The i^{th} element of a structure (including a sample, which is a multiset).

$x_{(i)}$ The i^{th} order statistic.

$x^{(i)}$ The value of x at the i^{th} iteration.

Realizations of random variables and placeholders in functions associated with random variables are usually represented by lowercase letters corresponding to the uppercase letters; thus, ϵ may represent a realization of the random variable E.

A single symbol in an italic font is used to represent a single variable. A Roman font or a special font is often used to represent a standard operator or a standard mathematical structure. Sometimes a string of symbols in a Roman font is used to represent an operator (or a standard function); for example, $\exp(\cdot)$ represents the exponential function. But a string of symbols in an italic font on the same baseline should be interpreted as representing a composition (probably by multiplication) of separate objects; for example, *exp* represents the product of e, x, and p. Likewise a string of symbols in a Roman font (usually a single symbol) is used to represent a fundamental constant; for example, e represents the base of the natural logarithm, while e represents a variable.

A fixed-width font is used to represent computer input or output, for example,

```
a = bx + sin(c).
```

In computer text, a string of letters or numerals with no intervening spaces or other characters, such as `bx` above, represents a single object, and there is no distinction in the font to indicate the type of object.

Some important mathematical structures and other objects are:

$\mathbb{R}$ The field of reals or the set over which that field is defined.

$\mathbb{R}_+$ The set of positive reals.

$\bar{\mathbb{R}}_+$ The nonnegative reals; $\bar{\mathbb{R}}_+ = \mathbb{R}_+ \cup \{0\}$.

$\mathbb{R}^d$	The usual d-dimensional vector space over the reals or the set of all d-tuples with elements in $\mathbb{R}$.
$\mathbb{R}^{n\times m}$	The vector space of real $n \times m$ matrices.
$\mathbb{Z}$	The ring of integers or the set over which that ring is defined.
$\mathcal{G}L(n)$	The general linear group; that is, the group of $n \times n$ full rank (real) matrices with Cayley multiplication.
$\mathcal{O}(n)$	The orthogonal group; that is, the group of $n \times n$ orthogonal (orthonormal) matrices with Cayley multiplication.
e	The base of the natural logarithm. This is a constant; e may be used to represent a variable. (Note the difference in the font.)
i	The imaginary unit, $\sqrt{-1}$. This is a constant; i may be used to represent a variable. (Note the difference in the font.)

A.2 Computer Number Systems

Computer number systems are used to simulate the more commonly used number systems. It is important to realize that they have different properties, however. Some notation for computer number systems follows.

$\mathbb{F}$	The set of floating-point numbers with a given precision, on a given computer system, or this set together with the four operators `+`, `-`, `*`, and `/`. ($\mathbb{F}$ is similar to $\mathbb{R}$ in some useful ways; see Sect. 10.1.2 and Table 10.4 on page 492.)
$\mathbb{I}$	The set of fixed-point numbers with a given length, on a given computer system, or this set together with the four operators `+`, `-`, `*`, and `/`. ($\mathbb{I}$ is similar to $\mathbb{Z}$ in some useful ways; see Sect. 10.1.1.)
$e_{\min}$ and $e_{\max}$	The minimum and maximum values of the exponent in the set of floating-point numbers with a given length (see page 470).
$\epsilon_{\min}$ and $\epsilon_{\max}$	The minimum and maximum spacings around 1 in the set of floating-point numbers with a given length (see page 472).

ϵ or ϵ_{mach} The machine epsilon, the same as ϵ_{min} (see page 472).

$[\cdot]_c$ The computer version of the object $\cdot$ (see page 484).

NA Not available; a missing-value indicator.

NaN Not-a-number (see page 475).

A.3 General Mathematical Functions and Operators

Functions such as sin, max, span, and so on that are commonly associated with groups of Latin letters are generally represented by those letters in a Roman font.

Operators such as d (the differential operator) that are commonly associated with a Latin letter are generally represented by that letter in a Roman font.

Note that some symbols, such as $|\cdot|$, are overloaded; such symbols are generally listed together below.

$\times$ Cartesian or cross product of sets, or multiplication of elements of a field or ring.

$|x|$ The modulus of the real or complex number x; if x is real, $|x|$ is the absolute value of x.

$\lceil x \rceil$ The ceiling function evaluated at the real number x: $\lceil x \rceil$ is the smallest integer greater than or equal to x. For any x, $\lfloor x \rfloor \leq x \leq \lceil x \rceil$.

$\lfloor x \rfloor$ The floor function evaluated at the real number x: $\lfloor x \rfloor$ is the largest integer less than or equal to x.

$x!$ The factorial of x. If x is a positive integer, $x! = x(x-1)\cdots 2 \cdot 1$. For all other values except negative integers, $x!$ is defined by $x! = \Gamma(x+1)$.

$\mathrm{O}(f(n))$ The order class big O with respect to $f(n)$.

$$g(n) \in \mathrm{O}(f(n))$$

means there exists some fixed c such that $\|g(n)\| \leq c\|f(n)\| \; \forall n$. In particular, $g(n) \in \mathrm{O}(1)$ means $g(n)$ is bounded.
In one special case, we will use $\mathrm{O}(f(n))$ to represent some unspecified scalar or vector $x \in \mathrm{O}(f(n))$. This is the case of a convergent series. An example is

$$s = f_1(n) + \cdots + f_k(n) + \mathrm{O}(f(n)),$$

where $f_1(n), \ldots, f_k(n)$ are finite constants.
We may also express the order class defined by convergence as $x \to a$ as $\mathrm{O}(f(x))_{x \to a}$ (where a may be infinite). Hence, $g \in \mathrm{O}(f(x))_{x \to a}$ iff

$$\limsup_{x \to a} \|g(n)\|/\|f(n)\| < \infty.$$

$\mathrm{o}(f(n))$ Little o; $g(n) \in \mathrm{o}(f(n))$ means for all $c > 0$ there exists some fixed N such that $0 \leq g(n) < cf(n) \; \forall n \geq N$. (The functions f and g and the constant c could all also be negative, with a reversal of the inequalities.) Hence, $g(n) \in \mathrm{o}(f(n))$ means $\|g(n)\|/\|f(n)\| \to 0$ as $n \to \infty$.
In particular, $g(n) \in \mathrm{o}(1)$ means $g(n) \to 0$.
We also use $\mathrm{o}(f(n))$ to represent some unspecified scalar or vector $x \in \mathrm{o}(f(n))$ in special case of a convergent series, as above:

$$s = f_1(n) + \cdots + f_k(n) + \mathrm{o}(f(n)).$$

We may also express this kind of convergence in the form $g \in \mathrm{o}(f(x))_{x \to a}$ as $x \to a$ (where a may be infinite).

$\mathrm{O}_P(f(n))$ Bounded convergence in probability; $X(n) \in \mathrm{O}_P(f(n))$ means that for any positive ϵ, there is a constant C_ϵ such that $\sup_n \Pr(\|X(n)\| \geq C_\epsilon \|f(n)\|) < \epsilon$.

$\mathrm{o}_P(f(n))$ Convergent in probability; $X(n) \in \mathrm{o}_P(f(n))$ means that for any positive ϵ, $\Pr(\|X(n) - f(n)\| > \epsilon) \to 0$ as $n \to \infty$.

d The differential operator.

Δ or δ A perturbation operator; Δx (or δx) represents a perturbation of x and not a multiplication of x by Δ (or δ), even if x is a type of object for which a multiplication is defined.

$\Delta(\cdot,\cdot)$ A real-valued difference function; $\Delta(x,y)$ is a measure of the difference of x and y. For simple objects, $\Delta(x,y) = |x-y|$. For more complicated objects, a subtraction operator may not be defined, and Δ is a generalized difference.

$\tilde{x}$ A perturbation of the object x; $\Delta(x,\tilde{x}) = \Delta x$.

$\tilde{x}$ An average of a sample of objects generically denoted by x.

$\bar{x}$ The mean of a sample of objects generically denoted by x.

$\bar{x}$ The complex conjugate of the complex number x; that is, if $x = r + \mathrm{i}c$, then $\bar{x} = r - \mathrm{i}c$.

$\mathrm{sign}(x)$ For the vector x, a vector of units corresponding to the signs:

$$\begin{aligned} \mathrm{sign}(x)_i &= 1 \quad \text{if } x_i > 0, \\ &= 0 \quad \text{if } x_i = 0, \\ &= -1 \quad \text{if } x_i < 0, \end{aligned}$$

with a similar meaning for a scalar.

A.3.1 Special Functions

A good general reference on special functions in mathematics is the *NIST Handbook of Mathematical Functions*, edited by Olver et al. (2010). Another good reference on special functions is the venerable book edited by Abramowitz and Stegun (1964), which has been kept in print by Dover Publications.

$\log x$ The natural logarithm evaluated at x.

$\sin x$ The sine evaluated at x (in radians) and similarly for other trigonometric functions.

$\Gamma(x)$ The complete gamma function: $\Gamma(x) = \int_0^\infty t^{x-1}\mathrm{e}^{-t}\mathrm{d}t$. (This is called Euler's integral.) Integration by parts immediately gives the replication formula $\Gamma(x+1) = x\Gamma(x)$, and so if x is a positive integer, $\Gamma(x+1) = x!$, and more generally, $\Gamma(x+1)$ defines $x!$ for all x except negative integers. Direct evaluation of the integral yields $\Gamma(1/2) = \sqrt{\pi}$. Using this and the replication formula, with some manipulation we get for the positive integer j

$$\Gamma(j+1/2) = \frac{1 \cdot 2 \cdots (2j-1)}{2^j}\sqrt{\pi}.$$

The integral does not exist, and thus, the gamma function is not defined at the nonpositive integers.
The notation $\Gamma_d(x)$ denotes the multivariate gamma function (page 221), although in other literature this notation denotes the incomplete univariate gamma function, $\int_0^d t^{x-1}\mathrm{e}^{-t}\mathrm{d}t$.

A.4 Linear Spaces and Matrices

$\mathcal{V}(G)$ For the set of vectors (all of the same order) G, the vector space generated by that set.

$\mathcal{V}(X)$ For the matrix X, the vector space generated by the columns of X.

$\dim(\mathcal{V})$ The dimension of the vector space $\mathcal{V}$; that is, the maximum number of linearly independent vectors in the vector space.

$\mathrm{span}(Y)$ For Y either a set of vectors or a matrix, the vector space $\mathcal{V}(Y)$.

$\perp$ Orthogonality relationship (vectors, see page 33; vector spaces, see page 34).

$\mathcal{V}^\perp$ The orthogonal complement of the vector space $\mathcal{V}$ (see page 34).

$\mathcal{N}(A)$ The null space of the matrix A; that is, the set of vectors generated by all solutions, z, of the homogeneous system $Az = 0$; $\mathcal{N}(A)$ is the orthogonal complement of $\mathcal{V}(A^{\mathrm{T}})$.

$\mathrm{tr}(A)$ The trace of the square matrix A, that is, the sum of the diagonal elements.

$\mathrm{rank}(A)$ The rank of the matrix A, that is, the maximum number of independent rows (or columns) of A.

$\sigma(A)$	The spectrum of the matrix A (the set of (unique) eigenvalues).
$\rho(A)$	The spectral radius of the matrix A (the maximum absolute value of its eigenvalues).
$A > 0$ $A \geq 0$	If A is a matrix, this notation means, respectively, that each element of A is positive or nonnegative.
$A \succ 0$ $A \succeq 0$	This notation means that A is a symmetric matrix and that it is, respectively, positive definite or nonnegative definite.
A^{T}	For the matrix A, its transpose (also used for a vector to represent the corresponding row vector).
A^{H}	The conjugate transpose, also called the adjoint, of the matrix A; $A^{\mathrm{H}} = \bar{A}^{\mathrm{T}} = \overline{A^{\mathrm{T}}}$.
A^{-1}	The inverse of the square, nonsingular matrix A.
$A^{-\mathrm{R}}$	The right inverse of the $n \times m$ matrix A, of rank n; $AA^{-\mathrm{R}} = I_n$. The right inverse is $m \times n$ and of full column rank.
$A^{-\mathrm{L}}$	The left inverse of the $n \times m$ matrix A, of rank m; $A^{-\mathrm{L}}A = I_m$. The right inverse is $m \times n$ and of full row rank.
$A^{-\mathrm{T}}$	The inverse of the transpose of the square, nonsingular matrix A.
A^{+}	The g_4 inverse, the Moore-Penrose inverse, or the pseudoinverse of the matrix A (see page 128).
A^{-}	A g_1, or generalized, inverse of the matrix A (see page 128).
$A^{\frac{1}{2}}$	The square root of a nonnegative definite or positive definite matrix A; $(A^{\frac{1}{2}})^2 = A$.
$A^{-\frac{1}{2}}$	The square root of the inverse of a positive definite matrix A; $(A^{-\frac{1}{2}})^2 = A^{-1}$.

$\odot$ Hadamard multiplication (see page 94).

$\otimes$ Kronecker multiplication (see page 95).

$\oplus$ The direct sum of two matrices; $A \oplus B = \text{diag}(A, B)$ (see page 63).

$\oplus$ Direct sum of vector spaces (see page 18).

A.4.1 Norms and Inner Products

L_p For real $p \geq 1$, a norm formed by accumulating the p^{th} powers of the moduli of individual elements in an object and then taking the $(1/p)^{\text{th}}$ power of the result (see page 27).

$\|\cdot\|$ In general, the norm of the object $\cdot$.

$\|\cdot\|_p$ In general, the L_p norm of the object $\cdot$.

$\|x\|_p$ For the vector x, the L_p norm

$$\|x\|_p = \left(\sum |x_i|^p\right)^{\frac{1}{p}}$$

(see page 27).

$\|X\|_p$ For the matrix X, the L_p norm

$$\|X\|_p = \max_{\|v\|_p=1} \|Xv\|_p$$

(see page 165).

$\|X\|_{\text{F}}$ For the matrix X, the Frobenius norm

$$\|X\|_{\text{F}} = \sqrt{\sum_{i,j} x_{ij}^2}$$

(see page 167).

$\|X\|_{\mathrm{F}_p}$ For the matrix X, the Frobenius p norm

$$\|X\|_{\mathrm{F}_p} = \left(\sum_{i,j} |x_{ij}|^p\right)^{1/p}.$$

$\|X\|_{\mathrm{S}_p}$ For the $n \times m$ matrix X, the Schatten p norm

$$\|X\|_{\mathrm{S}_p} = \left(\sum_{i=1}^{\min(n,m)} d_i^p\right)^{1/p},$$

where the d_i are the singular values of X.

$\langle x, y \rangle$ The inner product or dot product of x and y (see page 23; and see page 97 for matrices).

$\kappa_p(A)$ The L_p condition number of the nonsingular square matrix A with respect to inversion (see page 269).

A.4.2 Matrix Shaping Notation

$\mathrm{diag}(A)$ or $\mathrm{vecdiag}(A)$ For the vector A, the vector consisting of the elements of the principal diagonal of A;

$$\mathrm{diag}(A) = \mathrm{vecdiag}(A) = (a_{11}, \ldots, a_{kk}),$$

where k is the minimum of the number of rows and the number of columns of A.

$\mathrm{diag}(v)$ For the vector v, the diagonal matrix whose nonzero elements are those of v; that is, the square matrix, A, such that $A_{ii} = v_i$ and for $i \neq j$, $A_{ij} = 0$.

$\mathrm{diag}(A_1, A_2, \ldots, A_k)$ The block diagonal matrix whose submatrices along the diagonal are $A_1, A_2, \ldots, A_k$.

$\text{vec}(A)$ — The vector consisting of the columns of the matrix A all strung into one vector; if the column vectors of A are $a_1, a_2, \ldots, a_m$, then

$$\text{vec}(A) = (a_1^{\mathrm{T}}, a_2^{\mathrm{T}}, \ldots, a_m^{\mathrm{T}}).$$

$\text{vech}(A)$ — For the $m \times m$ symmetric matrix A, the vector consisting of the lower triangular elements all strung into one vector:

$$\text{vech}(A) = (a_{11}, a_{21}, \ldots, a_{m1}, a_{22}, \ldots, a_{m2}, \ldots, a_{mm}).$$

$A_{(i_1,\ldots,i_k)}$ — The matrix formed from rows $i_1, \ldots, i_k$ and columns $i_1, \ldots, i_k$ from a given matrix A. This kind of submatrix and the ones below occur often when working with determinants (for square matrices). If A is square, the determinants of these submatrices are called *minors* (see page 67). Because the principal diagonal elements of this matrix are principal diagonal elements of A, it is called a principal submatrix of A. Generally, but not necessarily, $i_j < i_{j+1}$.

$A_{(i_1,\ldots,i_k)(j_1,\ldots,j_l)}$ — The submatrix of a given matrix A formed from rows $i_1, \ldots, i_k$ and columns $j_1, \ldots, j_l$ from A.

$A_{(i_1,\ldots,i_k)(*)}$ or $A_{(*)(j_1,\ldots,j_l)}$ — The submatrix of a given matrix A formed from rows $i_1, \ldots, i_k$ and all columns or else all rows and columns $j_1, \ldots, j_l$ from A.

$A_{-(i_1,\ldots,i_k)(j_1,\ldots,j_l)}$ — The submatrix formed from a given matrix A by deleting rows $i_1, \ldots, i_k$ and columns $j_1, \ldots, j_l$.

$A_{-(i_1,\ldots,i_k)()}$ or $A_{-()(j_1,\ldots,j_l)}$ — The submatrix formed from a given matrix A by deleting rows $i_1, \ldots, i_k$ (and keeping all columns) or else by deleting columns $j_1, \ldots, j_l$ from A.

A.4.3 Notation for Rows or Columns of Matrices

a_{i*} The vector that corresponds to the i^{th} row of the matrix A. As with all vectors, this is a column vector, so it often appears in the form a_{i*}^{T}.

a_{*j} The vector that corresponds to the j^{th} column of the matrix A.

A.4.4 Notation Relating to Matrix Determinants

$|A|$ The determinant of the square matrix A, $|A| = \det(A)$.

$\det(A)$ The determinant of the square matrix A, $\det(A) = |A|$.

$\det\left(A_{(i_1,\ldots,i_k)}\right)$ A principal minor of a square matrix A; in this case, it is the minor corresponding to the matrix formed from rows $i_1, \ldots, i_k$ and columns $i_1, \ldots, i_k$ from a given matrix A. The notation $|A_{(i_1,\ldots,i_k)}|$ is also used synonymously.

$\det\left(A_{-(i)(j)}\right)$ The minor associated with the $(i,j)^{\text{th}}$ element of a square matrix A. The notation $|A_{-(i)(j)}|$ is also used synonymously.

$a_{(ij)}$ The cofactor associated with the $(i,j)^{\text{th}}$ element of a square matrix A; that is, $a_{(ij)} = (-1)^{i+j}|A_{-(i)(j)}|$.

$\text{adj}(A)$ The adjugate, also called the classical adjoint, of the square matrix A: $\text{adj}(A) = (a_{(ji)})$; that is, the matrix of the same size as A formed from the cofactors of the elements of A^{T}.

A.4.5 Matrix-Vector Differentiation

$\mathrm{d}t$ The differential operator on the scalar, vector, or matrix t. This is an operator; d may be used to represent a variable. (Note the difference in the font.)

g_f or ∇f For the scalar-valued function f of a vector variable, the vector whose i^{th} element is $\partial f/\partial x_i$. This is the gradient, also often denoted as g_f.

∇f For the vector-valued function f of a vector variable, the matrix whose element in position (i, j) is

$$\frac{\partial f_j(x)}{\partial x_i}.$$

This is also written as $\partial f^{\mathrm{T}}/\partial x$ or just as $\partial f/\partial x$. This is the transpose of the Jacobian of f.

J_f For the vector-valued function f of a vector variable, the Jacobian of f denoted as J_f. The element in position (i, j) is

$$\frac{\partial f_i(x)}{\partial x_j}.$$

This is the transpose of (∇f): $\mathrm{J}_f = (\nabla f)^{\mathrm{T}}$.

H_f or $\nabla\nabla f$ or $\nabla^2 f$ The Hessian of the scalar-valued function f of a vector variable. The Hessian is the transpose of the Jacobian of the gradient. Except in pathological cases, it is symmetric. The element in position (i, j) is

$$\frac{\partial^2 f(x)}{\partial x_i \partial x_j}.$$

The symbol $\nabla^2 f$ is sometimes also used to denote the trace of the Hessian, in which case it is called the Laplace operator.

A.4.6 Special Vectors and Matrices

1 or 1_n A vector (of length n) whose elements are all 1s.

0 or 0_n A vector (of length n) whose elements are all 0s.

I or I_n The $(n \times n)$ identity matrix.

e_i The i^{th} unit vector (with implied length) (see page 16).

A.4.7 Elementary Operator Matrices

E_{pq} The $(p, q)^{\text{th}}$ elementary permutation matrix (see page 81).

$E_{(\pi)}$ The permutation matrix that permutes the rows according to the permutation π.

$E_p(a)$ The p^{th} elementary scalar multiplication matrix (see page 82).

$E_{pq}(a)$ The $(p,q)^{\text{th}}$ elementary axpy matrix (see page 83).

A.5 Models and Data

A form of model used often in statistics and applied mathematics has three parts: a left-hand side representing an object of primary interest; a function of another variable and a parameter, each of which is likely to be a vector; and an adjustment term to make the right-hand side equal the left-hand side. The notation varies depending on the meaning of the terms. One of the most common models used in statistics, the linear regression model with normal errors, is written as

$$Y = \beta^{\mathrm{T}} x + E. \tag{A.1}$$

The adjustment term is a random variable, denoted by an uppercase epsilon. The term on the left-hand side is also a random variable. This model does not represent observations or data. A slightly more general form is

$$Y = f(x;\theta) + E. \tag{A.2}$$

A single observation or a single data item that corresponds to model (A.1) may be written as

$$y = \beta^{\mathrm{T}} x + \epsilon,$$

or, if it is one of several,

$$y_i = \beta^{\mathrm{T}} x_i + \epsilon_i.$$

Similar expressions are used for a single data item that corresponds to model (A.2).

In these cases, rather than being a random variable, ϵ or ϵ_i may be a realization of a random variable, or it may just be an adjustment factor with no assumptions about its origin.

A set of n such observations is usually represented in an n-vector y, a matrix X with n rows, and an n-vector ϵ:

$$y = X\beta + \epsilon$$

or

$$y = f(X;\theta) + \epsilon.$$

B

Solutions and Hints for Selected Exercises

Exercises Beginning on Page 52

2.3b. Let one vector space consist of all vectors of the form $(a, 0)$ and the other consist of all vectors of the form $(0, b)$. The vector (a, b) is not in the union if $a \neq 0$ and $b \neq 0$.

2.8. Give a counterexample to the triangle inequality; for example, let $x = (9, 25)$ and $y = (16, 144)$.

2.11a. We first observe that if $\|x\|_p = 0$ or $\|y\|_q = 0$, we have $x = 0$ or $y = 0$, and so the inequality is satisfied because both sides are 0; hence, we need only consider the case $\|x\|_p > 0$ and $\|y\|_q > 0$. We also observe that if $p = 1$ or $q = 1$, we have the Manhattan and Chebyshev norms and the inequality is satisfied; hence we need only consider the case $1 < p < \infty$.

Now, for p and q as given, for any numbers a_i and b_i, there are numbers s_i and t_i such that $|a_i| = \mathrm{e}^{s_i/p}$ and $|b_i| = \mathrm{e}^{t_i/q}$. Because e^x is a convex function, we have $\mathrm{e}^{s_i/p+t_i/q} \leq \frac{1}{p}\mathrm{e}^s_i + \frac{1}{q}\mathrm{e}^t_i$, or

$$a_i b_i \leq |a_i||b_i| \leq |a_i|^p/p + |b_i|^q/q.$$

Now let

$$a_i = \frac{x_i}{\|x\|_p} \quad \text{and} \quad b_i = \frac{y_i}{\|y\|_q},$$

and so

$$\frac{x_i}{\|x\|_p}\frac{y_i}{\|y\|_q} \leq \frac{1}{p}\frac{|x_i|^p}{\|x\|_p^p} + \frac{1}{q}\frac{|y_i|^q}{\|y\|_q^q}.$$

J.E. Gentle, *Matrix Algebra*, Springer Texts in Statistics,
DOI 10.1007/978-3-319-64867-5

Now, summing these equations over i, we have

$$\begin{aligned}\frac{\langle x, y\rangle}{\|x\|_p \|y\|_q} &\leq \frac{1}{p}\frac{\|x\|_p^p}{\|x\|_p^p} + \frac{1}{q}\frac{\|y\|_q^q}{\|y\|_q^q} \\ &= 1.\end{aligned}$$

Hence, we have the desired result.
As we see from this proof, the inequality is actually a little stronger than stated. If we define u and v by $u_i = |x_i|$ and $v_i = |y_i|$, we have

$$\langle x, y\rangle \leq \langle u, v\rangle \leq \|x\|_p \|y\|_q.$$

We observe that equality occurs if and only if

$$\left(\frac{|x_i|}{\|x\|_p}\right)^{\frac{1}{q}} = \left(\frac{|y_i|}{\|y\|_q}\right)^{\frac{1}{p}}$$

and

$$\operatorname{sign}(x_i) = \operatorname{sign}(y_i)$$

for all i.
We note a special case by letting $y = 1$:

$$\bar{x} \leq \|x\|_p,$$

and with $p = 2$, we have a special case of the Cauchy-Schwarz inequality,

$$n\bar{x}^2 \leq \|x\|_2^2,$$

which guarantees that $\mathrm{V}(x) \geq 0$.

2.11b. Using the triangle inequality for the absolute value, we have $|x_i + y_i| \leq |x_i| + |y_i|$. This yields the result for $p = 1$ and $p = \infty$ (in the limit). Now assume $1 < p < \infty$. We have

$$\|x + y\|_p^p \leq \sum_{i=1}^n |x_i + y_i|^{p-1}|x_i| + \sum_{i=1}^n |x_i + y_i|^{p-1}|y_i|.$$

Now, letting $q = p/(p-1)$, we apply Hölder's inequality to each of the terms on the right:

$$\sum_{i=1}^n |x_i + y_i|^{p-1}|x_i| \leq \left(\sum_{i=1}^n |x_i + y_i|^{(p-1)q}\right)^{\frac{1}{q}} \left(\sum_{i=1}^n |x_i|^p\right)^{\frac{1}{p}}$$

and

$$\sum_{i=1}^n |x_i + y_i|^{p-1}|x_i| \leq \left(\sum_{i=1}^n |x_i + y_i|^{(p-1)q}\right)^{\frac{1}{q}} \left(\sum_{i=1}^n |y_i|^p\right)^{\frac{1}{p}},$$

so

$$\sum_{i=1}^{n}|x_i+y_i|^p \le \left(\sum_{i=1}^{n}|x_i+y_i|^{(p-1)q}\right)^{\frac{1}{q}}\left(\left(\sum_{i=1}^{n}|x_i|^p\right)^{\frac{1}{p}}+\left(\sum_{i=1}^{n}|y_i|^p\right)^{\frac{1}{p}}\right)$$

or, because $(p-1)q = p$ and $1-\frac{1}{q}=\frac{1}{p}$,

$$\left(\sum_{i=1}^{n}|x_i+y_i|^p\right)^{\frac{1}{p}} \le \left(\sum_{i=1}^{n}|x_i|^p\right)^{\frac{1}{p}}+\left(\sum_{i=1}^{n}|y_i|^p\right)^{\frac{1}{p}},$$

which is the same as

$$\|x+y\|_p \le \|x\|_p + \|y\|_p,$$

the triangle inequality.

2.19e. In $\mathbb{R}^3$,

$$\text{angle}(x,y) = \sin^{-1}\left(\frac{\|x\times y\|}{\|x\|\,\|y|}\right).$$

Because $x\times y = -y\times x$, this allows us to determine the angle from x to y; that is, the *direction* within $(-\pi,\pi]$ in which x would be rotated to y.

2.21. Just consider the orthogonal vectors $x=(1,0)$ and $y=(0,1)$. The centered vectors are $x_c=(\frac{1}{2},-\frac{1}{2})$ and $y_c=(-\frac{1}{2},\frac{1}{2})$. The angle between the uncentered vectors is $\pi/2$, while that between the centered vectors is π.

Exercises Beginning on Page 178

3.2. This exercise occurs in various guises in many different places, and the simple approach is to add and subtract $\bar{x}$:

$$\begin{aligned}(x-a)^{\mathrm{T}}(x-a) &= (x-\bar{x}-(a+\bar{x}))^{\mathrm{T}}(x-\bar{x}-(a+\bar{x}))\\ &= (x_c-(a+\bar{x}))^{\mathrm{T}}(x_c-(a+\bar{x}))\\ &= x_c^{\mathrm{T}}x_c+(a+\bar{x})^{\mathrm{T}}(a+\bar{x}))-2(a+\bar{x})^{\mathrm{T}}x_c\\ &= x_c^{\mathrm{T}}x_c+n(a+\bar{x})^2.\end{aligned}$$

Finally, we get the expressions in equation (3.92) by writing $x_c^{\mathrm{T}}x_c$ as $\mathrm{tr}(x_cx_c^{\mathrm{T}})$.

3.14. Write the $n\times m$ matrix A as

$$A=(a_{ij})=[a_{*1},\ldots,a_{*m}].$$

If A is of rank one, the maximum number of linearly independent columns is one; hence, for $k=2,\ldots,m$, $a_{*k}=c_ka_{*1}$, for some c_k.

Now let $x = a_{*1}$, which is an n-vector, and let y be an m-vector whose first element is 1 and whose $k = 2, \ldots, m$ elements are the c_ks. We see that $A = xy^{\mathrm{T}}$ by direct multiplication.
This decomposition is not unique, of course.

3.15. If the elements of the square matrix A are integers then each cofactor $a_{(ij)}$ is an integer, and hence the elements of $\mathrm{adj}(A)$ are integers. If $|A| = \pm 1$, then $A^{-1} = \pm\mathrm{adj}(A)$, and so the elements of A^{-1} are integers.
An easy way to form a matrix whose determinant is ± 1 is to form an upper triangular matrix with either 1 or -1 on the diagonal. A more "interesting matrix" that has the same determinant can then be formed by use of elementary operations. (People teaching matrix algebra find this useful!)

3.16. Because the inverse of a matrix is unique, we can verify each equation by multiplication by the inverse at the appropriate place. We can often reduce an expression to a desired form by multiplication by the identity MM^{-1}, where M is some appropriate matrix. For example, equation (3.177) can be verified by the equations

$$\begin{aligned}
(A+B)(A^{-1} - A^{-1}(A^{-1}+B^{-1})^{-1}A^{-1}) &= \\
I - (A^{-1}+B^{-1})^{-1}A^{-1} + BA^{-1} - BA^{-1}(A^{-1}+B^{-1})^{-1}A^{-1} &= \\
I + BA^{-1} - (I+BA^{-1})(A^{-1}+B^{-1})^{-1}A^{-1} &= \\
(I+BA^{-1})(I - (A^{-1}+B^{-1})^{-1}A^{-1}) &= \\
B(B^{-1}+A^{-1})(I - (A^{-1}+B^{-1})^{-1}A^{-1}) &= \\
B(B^{-1}+A^{-1}) - BA^{-1} &= I
\end{aligned}$$

3.19. Express the nonzero elements of row i in the $n \times n$ matrix A as $a_i b_k$ for $k = i, \ldots, n$, and the nonzero elements of column j as $a_k b_j$ for $k = j, \ldots, n$. Then obtain expressions for the elements of A^{-1}. Show, for example, that the diagonal elements of A^{-1} are $(a_i b_i)^{-1}$.

3.25. For property 8, let c be a nonzero eigenvalue of AB. Then there exists v ($\neq 0$) such that $ABv = cv$, that is, $BABv = Bcv$. But this means $BAw = cw$, where $w = Bv \neq 0$ (because $ABv \neq 0$) and so c is an eigenvalue of BA. We use the same argument starting with an eigenvalue of BA. For square matrices, there are no other eigenvalues, so the set of eigenvalues is the same.
For property 9, see the discussion of similarity transformations on page 146.

3.37. Let A and B be such that AB is defined.

$$\|AB\|_{\mathrm{F}}^2 = \sum_{ij} \left| \sum_k a_{ik} b_{kj} \right|^2$$

$$\begin{aligned}
&\leq \sum_{ij}\left(\sum_{k} a_{ik}^2\right)\left(\sum_{k} b_{kj}^2\right) \quad \text{(Cauchy-Schwarz)} \\
&= \left(\sum_{i,k} a_{ik}^2\right)\left(\sum_{k,j} b_{kj}^2\right) \\
&= \|A\|_{\mathrm{F}}^2 \|B\|_{\mathrm{F}}^2.
\end{aligned}$$

3.40. Hints.
For inequality (3.307), use the Cauchy-Schwartz inequality.
For inequality (3.309), use Hölder's inequality.

Exercises Beginning on Page 222

4.7b. The first step is to use the trick of equation (3.90), $x^{\mathrm{T}}Ax = \mathrm{tr}(Axx^{\mathrm{T}})$, again to undo the earlier expression, and write the last term in equation (4.44) as

$$-\frac{n}{2}\mathrm{tr}\left(\Sigma^{-1}(\bar{y}-\mu)(\bar{y}-\mu)^{\mathrm{T}}\right) = -\frac{n}{2}(\bar{y}-\mu)\Sigma^{-1}(\bar{y}-\mu)^{\mathrm{T}}.$$

Now Σ^{-1} is positive definite, so $(\bar{y}-\mu)\Sigma^{-1}(\bar{y}-\mu)^{\mathrm{T}} \geq 0$ and hence is minimized for $\hat{\mu} = \bar{y}$. Decreasing this term increases the value of $l(\mu, \Sigma; y)$, and so $l(\hat{\mu}, \Sigma; y) \geq l(\mu, \Sigma; y)$ for all positive definite Σ^{-1}.
Now, we consider the other term. Let $A = \sum_{i=1}^{n}(y_i - \bar{y})(y_i - \bar{y})^{\mathrm{T}}$. The first question is whether A is positive definite. We will refer to a text on multivariate statistics for the proof that A is positive definite with probability 1 (see Muirhead 1982, for example). We have

$$\begin{aligned}
l(\hat{\mu}, \Sigma; y) &= c - \frac{n}{2}\log|\Sigma| - \frac{1}{2}\mathrm{tr}\left(\Sigma^{-1}A\right) \\
&= c - \frac{n}{2}\left(\log|\Sigma| + \mathrm{tr}\left(\Sigma^{-1}A/n\right)\right).
\end{aligned}$$

Because c is constant, the function is maximized at the minimum of the latter term subject to Σ being positive definite, which, as shown for expression (4.61), occurs at $\widehat{\Sigma} = A/n$.

4.12. $2^{dn/2}\Gamma_d(n/2)|\Sigma|^{n/2}$.
Make the change of variables $W = 2\Sigma^{\frac{1}{2}}Y\Sigma^{\frac{1}{2}}$, determine the Jacobian, and integrate.

Exercises Beginning on Page 261

5.2. The R code that will produce the graph is

```
x<-c(0,1)
y<-c(0,1)
z<-matrix(c(0,0,1,1),nrow=2)
trans<-persp(x, y, z, theta = 45, phi = 30)
bottom<-c(.5,0,0,1)%*%trans
top<-c(.5,1,1,1)%*%trans
xends<-c(top[,1]/top[,4],bottom[,1]/bottom[,4])
yends<-c(top[,2]/top[,4],bottom[,2]/bottom[,4])
lines(xends,yends,lwd=2)
```

5.5. Let A is an $n \times n$ matrix, whose columns are the same as the vectors a_j, and let QR be the QR factorization of A. Because Q is orthogonal, $\det(Q) = 1$, and $\det(R) = \det(A)$. Hence, we have

$$\begin{aligned} |\det(A)| &= |\det(R)| \\ &= \prod_{j=1}^{n} |r_{jj}| \\ &\leq \prod_{j=1}^{n} \|r_j\|_2 \\ &= \prod_{j=1}^{n} \|a_j\|_2, \end{aligned}$$

where r_j is the vector whose elements are the same as the elements in the j^{th} column of R.

If equality holds, then either some a_j is zero, or else $r_{jj} = \|r_j\|$ for $j = 1, \ldots, n$. In the latter case, R is diagonal, and hence $A^{\mathrm{T}}A$ is diagonal, and so the columns of A are orthogonal.

Exercises Beginning on Page 305

6.1. First, show that

$$\max_{x \neq 0} \frac{\|Ax\|}{\|x\|} = \left(\min_{x \neq 0} \frac{\|A^{-1}x\|}{\|x\|} \right)^{-1}$$

and

$$\max_{x \neq 0} \frac{\|A^{-1}x\|}{\|x\|} = \left(\min_{x \neq 0} \frac{\|Ax\|}{\|x\|} \right)^{-1}.$$

6.2a. The matrix prior to the first elimination is

$$\begin{bmatrix} 2 & 5 & 3 & 19 \\ 1 & 4 & 1 & 12 \\ 1 & 2 & 2 & 9 \end{bmatrix}.$$

The solution is $(3, 2, 1)$.

6.2b. The matrix prior to the first elimination is

$$\begin{bmatrix} 5 & 2 & 3 & 19 \\ 4 & 1 & 1 & 12 \\ 2 & 1 & 2 & 9 \end{bmatrix},$$

and x_1 and x_2 have been interchanged.

6.2c.

$$D = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 2 \end{bmatrix},$$

$$L = \begin{bmatrix} 0 & 0 & 0 \\ 2 & 0 & 0 \\ 1 & 2 & 0 \end{bmatrix},$$

$$U = \begin{bmatrix} 0 & 4 & 1 \\ 0 & 0 & 3 \\ 0 & 0 & 0 \end{bmatrix},$$

$$\rho((D + L)^{-1}U) = 1.50.$$

6.2e. $\rho((\widetilde{D} + \widetilde{L})^{-1}\widetilde{U}) = 0.9045$.

6.2g. Some R code for this is

```
tildeD <- matrix(c(2, 0, 0,
                   0, 4, 0,
                   0, 0, 2), nrow=3, byrow=T)
tildeL <- matrix(c(0, 0, 0,
                   1, 0, 0,
                   1, 2, 0), nrow=3, byrow=T)
tildeU <- matrix(c(0, 5, 3,
                   0, 0, 1,
                   0, 0, 0), nrow=3, byrow=T)
b <- c(12,19,9)

omega <- 0.1
tildeDUadj <- (1-omega)*tildeD - omega*tildeU
tildeAk <- tildeD+omega*tildeL
badj <- omega*b
xk <- c(1,1,1)
```

```
nstep <- 2
for (i in 1:nstep){
   bk <- tildeDUadj%*%xk + badj
   xkp1 <- solve(tildeAk,bk)
   dif <- sqrt(sum((xkp1-xk)^2))
   print(dif)
   xk <- xkp1
}
```

6.4a. $nm(m+1) - m(m+1)/2$. (Remember $A^{\mathrm{T}}A$ is symmetric.)

6.4g. Using the normal equations with the Cholesky decomposition requires only about half as many flops as the QR, when n is much larger than m. The QR method oftens yields better accuracy, however.

6.5a.
1. $X(X^{\mathrm{T}}X)^{-1}X^{\mathrm{T}}X = X$
2. $(X^{\mathrm{T}}X)^{-1}X^{\mathrm{T}}X(X^{\mathrm{T}}X)^{-1}X^{\mathrm{T}} = (X^{\mathrm{T}}X)^{-1}X^{\mathrm{T}}$
3. $X(X^{\mathrm{T}}X)^{-1}X^{\mathrm{T}}$ is symmetric (take its transpose).
4. $(X^{\mathrm{T}}X)^{-1}X^{\mathrm{T}}X$ is symmetric.

Therefore, $(X^{\mathrm{T}}X)^{-1}X^{\mathrm{T}} = X^{+}$.

6.5b. We want to show $X^{\mathrm{T}}(y - XX^{+}y) = 0$. Using the properties of X^{+}, we have

$$\begin{aligned} X^{\mathrm{T}}(y - XX^{+}y) &= X^{\mathrm{T}}y - X^{\mathrm{T}}XX^{+}y \\ &= X^{\mathrm{T}}y - X^{\mathrm{T}}(XX^{+})^{\mathrm{T}}y \quad \text{because of symmetry} \\ &= X^{\mathrm{T}}y - X^{\mathrm{T}}(X^{+})^{\mathrm{T}}X^{\mathrm{T}}y \\ &= X^{\mathrm{T}}y - X^{\mathrm{T}}(X^{\mathrm{T}})^{+}X^{\mathrm{T}}y \quad \text{property of Moore-Penrose inverses and transposes} \\ &= X^{\mathrm{T}}y - X^{\mathrm{T}}y \quad \text{property of Moore-Penrose inverses} \\ &= 0 \end{aligned}$$

Exercises Beginning on Page 324

7.1a. 1.

7.1b. 1.

7.1d. 1. (All that was left was to determine the probability that $c_n \neq 0$ and $c_{n-1} \neq 0$.)

7.2a. 11.6315.

7.3.

$$\begin{bmatrix} 3.08 & -0.66 & 0 & 0 \\ -0.66 & 4.92 & -3.27 & 0 \\ 0 & -3.27 & 7.00 & -3.74 \\ 0 & 0 & -3.74 & 7.00 \end{bmatrix}.$$

Exercises Beginning on Page 396

8.10a.

$$p(c) = c^m - \alpha_1 c^{m-1} - \alpha_2\sigma_1 c^{m-2} - \alpha_3\sigma_1\sigma_2 c^{m-3} - \cdots - \alpha_m\sigma_1\sigma_2\cdots\sigma_{m-1}.$$

8.10b. Define

$$f(c) = 1 - \frac{p(c)}{c^m}.$$

This is a monotone decreasing continuous function in c, with $f(c) \to \infty$ as $c \to 0_+$ and $f(c) \to 0$ as $c \to \infty$. Therefore, there is a unique value c_* for which $f(c_*) = 1$. The uniqueness also follows from Descartes' rule of signs, which states that the maximum number of positive roots of a polynomial is the number of sign changes of the coefficients, and in the case of the polynomial $p(c)$, this is one.

8.18. This is a simple case of matrix multiplication.
To illustrate the use of R in complex matrices, I will show some code that is relevant to this problem, for a given order, of course.

```
omegajn <- function(n){
#####    Function to create the n^th roots of 1  #####
omegajn <- complex(n)
omegajn[1] <- 1
if (n>=2) omegajn[2] <- complex(re=cos(2*pi/n),
im=sin(2*pi/n))
if (n>=3) for (j in 3:n) omegajn[j] <- omegajn[2]
*omegajn[j-1]
return(omegajn)
}

Fn <- function(n){
#####    Function to create a Fourier matrix  #####
rts <- omegajn(n)
Fn <- matrix(c(rep(1,n),rts),nrow=n)
if (n>=3) for (j in 3:n) Fn <- cbind(Fn,rts^(j-1))
Fn <- Fn/sqrt(n)
return(Fn)
}

# perform multiplications to get the elementary
circulant matrix or order 5
round(Conj(t(F5))%*%diag(rts5)%*%F5)
```

8.19. $(-1)^{\lfloor n/2 \rfloor} n^n$, where $\lfloor \cdot \rfloor$ is the floor function (the greatest integer function). For $n = 1, 2, 3, 4$, the determinants are $1, -4, -27, 256$.

Exercises Beginning on Page 452

9.1. 1. This is because the subspace that generates a singular matrix is a lower dimensional space than the full sample space, and so its measure is 0.

9.4d. Assuming W is positive definite, we have

$$\widehat{\beta}_{W,C} = (X^{\mathrm{T}}WX)^{-1}X^{\mathrm{T}}Wy + (X^{\mathrm{T}}WX)^{-1}L^{\mathrm{T}}(L(X^{\mathrm{T}}WX)^{+}L^{\mathrm{T}})^{-1}(c - L(X^{\mathrm{T}}WX)^{+}X^{\mathrm{T}}Wy)\,.$$

9.12. Let $X = [X_{\mathrm{i}} \mid X_{\mathrm{o}}]$ and $Z = X_{\mathrm{o}}^{\mathrm{T}}X_{\mathrm{o}} - X_{\mathrm{o}}^{\mathrm{T}}X_{\mathrm{i}}(X_{\mathrm{i}}^{\mathrm{T}}X_{\mathrm{i}})^{-1}X_{\mathrm{i}}^{\mathrm{T}}X_{\mathrm{o}}$. Note that $X_{\mathrm{o}}^{\mathrm{T}}X_{\mathrm{i}} = X_{\mathrm{i}}^{\mathrm{T}}X_{\mathrm{o}}$. We have

$$\begin{aligned}
&X_{\mathrm{i}}^{\mathrm{T}}X(X^{\mathrm{T}}X)^{-1}X^{\mathrm{T}} \\
&= X_{\mathrm{i}}^{\mathrm{T}}[X_{\mathrm{i}} \mid X_{\mathrm{o}}]\left[\begin{array}{c|c} X_{\mathrm{i}}^{\mathrm{T}}X_{\mathrm{i}} & X_{\mathrm{i}}^{\mathrm{T}}X_{\mathrm{o}} \\ \hline X_{\mathrm{o}}^{\mathrm{T}}X_{\mathrm{i}} & X_{\mathrm{o}}^{\mathrm{T}}X_{\mathrm{o}} \end{array}\right]^{-1}[X_{\mathrm{i}} \mid X_{\mathrm{o}}]^{\mathrm{T}} \\
&= [X_{\mathrm{i}}^{\mathrm{T}}X_{\mathrm{i}} \mid X_{\mathrm{i}}^{\mathrm{T}}X_{\mathrm{o}}] \\
&\quad \left[\begin{array}{c|c} (X_{\mathrm{i}}^{\mathrm{T}}X_{\mathrm{i}})^{-1} - (X_{\mathrm{i}}^{\mathrm{T}}X_{\mathrm{i}})^{-1}(X_{\mathrm{o}}^{\mathrm{T}}X_{\mathrm{i}})Z^{-1}(X_{\mathrm{i}}^{\mathrm{T}}X_{\mathrm{o}})(X_{\mathrm{i}}^{\mathrm{T}}X_{\mathrm{i}})^{-1} & \\ -Z^{-1}(X_{\mathrm{o}}^{\mathrm{T}}X_{\mathrm{i}})(X_{\mathrm{i}}^{\mathrm{T}}X_{\mathrm{i}})^{-1} & -(X_{\mathrm{i}}^{\mathrm{T}}X_{\mathrm{i}})^{-1}(X_{\mathrm{i}}^{\mathrm{T}}X_{\mathrm{o}})Z^{-1} \\ \hline & Z^{-1} \end{array}\right] \\
&\quad \left[\begin{array}{c} X_{\mathrm{i}}^{\mathrm{T}} \\ X_{\mathrm{o}}^{\mathrm{T}} \end{array}\right] \\
&= [I - (X_{\mathrm{o}}^{\mathrm{T}}X_{\mathrm{i}})Z^{-1}(X_{\mathrm{i}}^{\mathrm{T}}X_{\mathrm{o}})(X_{\mathrm{i}}^{\mathrm{T}}X_{\mathrm{i}})^{-1} - X_{\mathrm{i}}^{\mathrm{T}}X_{\mathrm{o}}Z^{-1}(X_{\mathrm{o}}^{\mathrm{T}}X_{\mathrm{i}})(X_{\mathrm{i}}^{\mathrm{T}}X_{\mathrm{i}})^{-1} \quad \mid \\
&\qquad\qquad -X_{\mathrm{i}}^{\mathrm{T}}X_{\mathrm{o}}Z^{-1} + X_{\mathrm{i}}^{\mathrm{T}}X_{\mathrm{o}}Z^{-1}] \\
&\quad \left[\begin{array}{c} X_{\mathrm{i}}^{\mathrm{T}} \\ X_{\mathrm{o}}^{\mathrm{T}} \end{array}\right] \\
&= X_{\mathrm{i}}^{\mathrm{T}},
\end{aligned}$$

9.14. One possibility is

$$\left[\begin{array}{cc} 20 & 100 \\ 5 & 25 \\ 5 & 25 \\ 10 & \texttt{NA} \\ 10 & \texttt{NA} \\ 10 & \texttt{NA} \\ \texttt{NA} & 10 \\ \texttt{NA} & 10 \\ \texttt{NA} & 10 \end{array}\right].$$

The variance-covariance matrix computed from all pairwise complete observations is

$$\begin{bmatrix} 30 & 375 \\ 375 & 1230 \end{bmatrix},$$

while that computed only from complete cases is

$$\begin{bmatrix} 75 & 375 \\ 375 & 1875 \end{bmatrix}.$$

The correlation matrix computed from all pairwise complete observations is

$$\begin{bmatrix} 1.00 & 1.95 \\ 1.95 & 1.00 \end{bmatrix}.$$

Note that this example is not a pseudo-correlation matrix.
In the R software system, the `cov` and `cor` functions have an argument called "`use`", which can take the values "all.obs", "complete.obs", or "pairwise.complete.obs". The value "all.obs" yields an error if the data matrix contains any missing values. In `cov`, the values "complete.obs" and "pairwise.complete.obs" yield the variance-covariances shown above. The function `cor` with `use=‘‘pairwise.complete.obs’’` yields

$$\begin{bmatrix} 1.00 & 1.00 \\ 1.00 & 1.00 \end{bmatrix}.$$

However, if `cov` is invoked with `use=‘‘pairwise.complete.obs’’` and the function `cov2cor` is applied to the result, the correlations are 1.95, as in the first correlation matrix above.

9.17. This is an open question. If you get a proof of convergence, submit it for publication. You may wish to try several examples and observe the performance of the intermediate steps. I know of no case in which the method has not converged.

9.19b. Starting with the correlation matrix given above as a possible solution for Exercise 9.14, four iterations of equation (9.67) using $\delta = 0.05$ and $f(x) = \tanh(x)$ yield

$$\begin{bmatrix} 1.00 & 0.997 \\ 0.997 & 1.00 \end{bmatrix}.$$

9.21. We can develop a recursion for p_{11}^t based on p_{11}^{t-1} and p_{12}^{t-1},

$$p_{11}^t = p_{11}^{t-1}(1-\alpha) + p_{12}^{t-1}\beta,$$

and because $p_{11} + p_{12} = 1$, we have $p_{11}^t = p_{11}^{t-1}(1-\alpha-\beta) + \beta$. Putting this together, we have

$$\lim_{t\to\infty} P = \begin{bmatrix} \beta/(\alpha+\beta) & \alpha/(\alpha+\beta) \\ \beta/(\alpha+\beta) & \alpha/(\alpha+\beta) \end{bmatrix},$$

and so the limiting (and invariant) distribution is $\pi_s = (\beta/(\alpha+\beta),\ \alpha/(\alpha+\beta))$.

9.22c. From the exponential growth, we have $N^{(T)} = N^{(0)}e^{rT}$; hence,

$$r = \frac{1}{T}\log\left(N^{(T)}/N^{(0)}\right) = \frac{1}{T}\log(r_0).$$

Exercises Beginning on Page 516

10.1a. The computations do not overflow. The first floating-point number x such that $x + 1 = x$ is

$$0.10\cdots0 \times b^{p+1}.$$

Therefore, the series converges at the value of i such that $i(i+1)/2 = x$. Now solve for i.

10.2. The function is $\log(n)$, and Euler's constant is $0.57721\ldots$.

10.6. 2^{-56}. (The standard has 53 bits normalized, so the last bit is 2^{-55}, and half of that is 2^{-56}.)

10.7a. Normalized: $2b^{p-1}(b-1)(e_{\max} - e_{\min} + 1) + 1$.
Nonnormalized: $2b^{p-1}(b-1)(e_{\max} - e_{\min} + 1) + 1 + 2b^{p-1}$.

10.7b. Normalized: $b^{e_{\min}-1}$.
Nonnormalized: $b^{e_{\min}-p}$.

10.7c. $1 + b^{-p+1}$ or $1 + b^{-p}$ when $b = 2$ and the first bit is hidden.

10.7d. b^p.

10.7e. 22.

10.12. First of all, we recognize that the full sum in each case is 1. We therefore accumulate the sum from the direction in which there are fewer terms. After computing the first term from the appropriate direction, take a logarithm to determine a scaling factor, say s^k. (This term will be the smallest in the sum.) Next, proceed to accumulate terms until the sum is of a different order of magnitude than the next term. At that point, perform a scale adjustment by dividing by s. Resume summing, making similar scale adjustments as necessary, until the limit of the summation is reached.

10.14. The result is close to 1.
What is relevant here is that numbers close to 1 have only a very few digits of accuracy; therefore, it would be better to design this program so that it returns $1 - \Pr(X \leq x)$ (the "significance level"). The purpose and the anticipated use of a program determine how it should be designed.

10.17a. 2.

10.17b. 0.

10.17c. No (because the operations in the "for" loop are not chained).

10.18c.

$$
\begin{aligned}
&a = x_1\\
&b = y_1\\
&s = 0\\
&\text{for } i = 2, n\\
&\{\\
&\quad d = (x_i - a)/i\\
&\quad e = (y_i - b)/i\\
&\quad a = d + a\\
&\quad b = e + b\\
&\quad s = i(i-1)de + s\\
&\}.
\end{aligned}
$$

10.21. 1. No; 2. Yes; 3. No; 4. No.

10.22. A very simple example is

$$\begin{bmatrix} 1 & 1+\epsilon \\ 1 & 1 \end{bmatrix},$$

where $\epsilon < b^{-p}$, because in this case the matrix stored in the computer would be singular. Another example is

$$\begin{bmatrix} 1 & a(1+\epsilon) \\ a(1+\epsilon) & a^2(1+2\epsilon) \end{bmatrix},$$

where ϵ is the machine epsilon.

Exercises Beginning on Page 537

11.2a. $O(nk)$.

11.2c. At each successive stage in the fan-in, the number of processors doing the additions goes down by approximately one-half.
If $p \approx k$, then $O(n \log k)$ (fan-in on one element of c at a time)
If $p \approx nk$, then $O(\log k)$ (fan-in on all elements of c simultaneously)
If p is a fixed constant smaller than k, the order of time does not change; only the multiplicative constant changes.
Notice the difference in the order of time and the order of the number of computations. Often there is very little that can be done about the order of computations.

11.2d. Because in a serial algorithm the magnitudes of the summands become more and more different. In the fan-in, they are more likely to remain relatively equal. Adding magnitudes of different quantities results in benign roundoff, but many benign roundoffs become bad. (This is not

catastrophic cancellation.) Clearly, if all elements are nonnegative, this argument would hold. Even if the elements are randomly distributed, there is likely to be a drift in the sum (this can be thought of as a random walk). There is *no difference* in the number of computations.

11.2e. Case 1: $p \approx n$. Give each c_i a processor – do an outer loop on each. This would likely be more efficient because all processors are active at once.
Case 2: $p \approx nk$. Give each $a_{ij}b_j$ a processor – fan-in for each. This would be the same as the other.
If p is a fixed constant smaller than n, set it up as in Case 1, using n/p groups of c_i's.

11.2f. If $p \approx n$, then $O(k)$.
If $p \approx nk$, then $O(\log k)$.
If p is some small fixed constant, the order of time does not change; only the multiplicative constant changes.

Exercises Beginning on Page 582

12.2. Here is a recursive Matlab function for the Strassen algorithm due to Coleman and Van Loan. When it uses the Strassen algorithm, it requires the matrices to have even dimension.

```
   function C = strass(A,B,nmin)
%
% Strassen matrix multiplication C=AB
%       A, B must be square and of even dimension
% From Coleman and Van Loan
% If n <= nmin, the multiplication is done conventionally
%
   [n n ] = size(A);
   if n <= nmin
      C = A * B;   % n is small, get C conventionally
   else
      m = n/2; u = 1:m; v = m+1:n;
      P1 = strass(A(u,u)+A(v,v), B(u,u)+B(v,v), nmin);
      P2 = strass(A(v,u)+A(v,v), B(u,u),        nmin);
      P3 = strass(A(u,u),        B(u,v)-B(v,v), nmin);
      P4 = strass(A(v,v),        B(v,u)-B(u,u), nmin);
      P5 = strass(A(u,u)+A(u,v), B(v,v),        nmin);
      P6 = strass(A(v,u)-A(u,u), B(u,u)+B(u,v), nmin);
      P7 = strass(A(u,v)-A(v,v), B(v,u)+B(v,v), nmin);
      C = [P1+P4-P5+P7 P3+P5; P2+P4 P1+P3-P2+P6];
   end
```

12.5a.

```
real a(4,3)
data a/3.,6.,8.,2.,5.,1.,6.,3.,6.,2.,7.,1./
n = 4
m = 3
x1 = a(2,2) ! Temporary variables must be used
     because of
x2 = a(4,2) ! the side effects of srotg.
call srotg(x1, x2,, c, s)
call srot(m, a(2,1), n, a(4,1), n, c, s)
print *, c, s
print *, a
end
```

This yields 0.3162278 and 0.9486833 for c and s. The transformed matrix is

$$\begin{bmatrix} 3.000000 & 5.000000 & 6.000000 \\ 3.794733 & 3.162278 & 1.581139 \\ 8.000000 & 6.000000 & 7.000000 \\ -5.059644 & -0.00000002980232 & -1.581139 \end{bmatrix}.$$

12.7b. Using the Matrix package in R, after initializing `rho` and `sig2`, this is

```
Vinv <- sparseMatrix(i=c(1,1,2,2,2,3,3,3,4,4,4,5,5,5,6,6,6,7,
                         7,7,8,8,8,9,9,9,10,10),
                     j=c(1,2,1:3,2:4,3:5,4:6,5:7,6:8,7:9,8:10,
                         9,10),
                     x=c(1,-rho,rep(c(-rho,1+rho^2,-rho),8),
                         -rho,1))/((1-rho^2)*sig2)
```

12.9. 10.7461941829033 and 10.7461941829034.

12.11.

$$\frac{-(fh)+ei}{-(ceg)+bfg+cdh-afh-bdi+aei} \qquad \frac{ch-bi}{-(ceg)+bfg+cdh-afh-bdi+aei}$$

$$\frac{-(ce)+bf}{-(ceg)+bfg+cdh-afh-bdi+aei} \qquad \frac{fg-di}{-(ceg)+bfg+cdh-afh-bdi+aei}$$

$$\frac{-(cg)+ai}{-(ceg)+bfg+cdh-afh-bdi+aei} \qquad \frac{cd-af}{-(ceg)+bfg+cdh-afh-bdi+aei}$$

$$\frac{-(eg)+dh}{-(ceg)+bfg+cdh-afh-bdi+aei} \qquad \frac{bg-ah}{-(ceg)+bfg+cdh-afh-bdi+aei}$$

$$\frac{-(bd)+ae}{-(ceg)+bfg+cdh-afh-bdi+aei}$$

Bibliography

Many of the most useful background material is available on the internet. For statistics, one of the most useful sites on the internet is the electronic repository `statlib`, maintained at Carnegie Mellon University, which contains programs, datasets, and other items of interest. The URL is

`http://lib.stat.cmu.edu`.

The collection of algorithms published in *Applied Statistics* is available in `statlib`. These algorithms are sometimes called the *ApStat* algorithms.

Another very useful site for scientific computing is `netlib`,. The URL is

`http://www.netlib.org`

The *Collected Algorithms of the ACM* (*CALGO*), which are the Fortran, C, and Algol programs published in *ACM Transactions on Mathematical Software* (or in *Communications of the ACM* prior to 1975), are available in `netlib` under the TOMS link.

A wide range of software is used in the computational sciences. Some of the software is produced by a single individual who is happy to share the software, sometimes for a fee, but who has no interest in maintaining it. At the other extreme is software produced by large commercial companies whose continued existence depends on a process of production, distribution, and maintenance of the software. Information on much of the software can be obtained from GAMS, as we mentioned at the beginning of Chap. 12. Some of the free software can be obtained from `statlib` or `netlib`.

The following bibliography obviously covers a wide range of topics in statistical computing and computational statistics. Except for a few of the general references, all of these entries have been cited in the text.

J.E. Gentle, *Matrix Algebra*, Springer Texts in Statistics,
DOI 10.1007/978-3-319-64867-5

Abramowitz, Milton, and Irene A. Stegun, eds. 1964. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Washington: National Bureau of Standards (NIST). (Reprinted in 1965 by Dover Publications, Inc., New York.)

Alefeld, Göltz, and Jürgen Herzberger. (1983). *Introduction to Interval Computation*. New York: Academic Press.

Ammann, Larry, and John Van Ness. 1988. A routine for converting regression algorithms into corresponding orthogonal regression algorithms. *ACM Transactions on Mathematical Software* 14:76–87.

Anda, Andrew A., and Haesun Park. 1994. Fast plane rotations with dynamic scaling. *SIAM Journal of Matrix Analysis and Applications* 15:162–174.

Anda, Andrew A., and Haesun Park. 1996. Self-scaling fast rotations for stiff least squares problems. *Linear Algebra and Its Applications* 234:137–162.

Anderson, E., Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenhaum, S. Hammarling, A. McKenney, and D. Sorensen. 2000. *LAPACK Users' Guide*, 3rd ed. Philadelphia: Society for Industrial and Applied Mathematics.

Anderson, T. W. 1951. Estimating linear restrictions on regression coefficients for multivariate nomal distributions. *Annals of Mthematical Statistics* 22:327–351.

Anderson, T. W. 2003. *An Introduction to Multivariate Statistical Analysis*, 3rd ed. New York: John Wiley and Sons.

ANSI. 1978. *American National Standard for Information Systems — Programming Language FORTRAN*, Document X3.9-1978. New York: American National Standards Institute.

ANSI. 1989. *American National Standard for Information Systems — Programming Language C*, Document X3.159-1989. New York: American National Standards Institute.

ANSI. 1992. *American National Standard for Information Systems — Programming Language Fortran-90*, Document X3.9-1992. New York: American National Standards Institute.

ANSI. 1998. *American National Standard for Information Systems — Programming Language C++*, Document ISO/IEC 14882-1998. New York: American National Standards Institute.

Atkinson, A. C., and A. N. Donev. 1992. *Optimum Experimental Designs*. Oxford, United Kingdom: Oxford University Press.

Attaway, Stormy. 2016. *Matlab: A Practical Introduction to Programming and Problem Solving*, 4th ed. Oxford, United Kingdom: Butterworth-Heinemann.

Bailey, David H. 1993. Algorithm 719: Multiprecision translation and execution of FORTRAN programs. *ACM Transactions on Mathematical Software* 19:288–319.

Bailey, David H. 1995. A Fortran 90-based multiprecision system. *ACM Transactions on Mathematical Software* 21:379–387.

Bailey, David H., King Lee, and Horst D. Simon. 1990. Using Strassen's algorithm to accelerate the solution of linear systems. *Journal of Supercomputing* 4:358–371.

Bapat, R. B., and T. E. S. Raghavan. 1997. *Nonnegative Matrices and Applications*. Cambridge, United Kingdom: Cambridge University Press.

Barker, V. A., L. S. Blackford, J. Dongarra, J. Du Croz, S. Hammarling, M. Marinova, J. Wasniewsk, and P. Yalamov. 2001. *LAPACK95 Users' Guide*. Philadelphia: Society for Industrial and Applied Mathematics.

Barrett, R., M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. 1994. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, 2nd ed. Philadelphia: Society for Industrial and Applied Mathematics.

Basilevsky, A. 1983. *Applied Matrix Algebra in the Statistical Sciences*. New York: North Holland

Beaton, Albert E., Donald B. Rubin, and John L. Barone. 1976. The acceptability of regression solutions: Another look at computational accuracy. *Journal of the American Statistical Association* 71:158–168.

Benzi, Michele. 2002. Preconditioning techniques for large linear systems: A survey. *Journal of Computational Physics* 182:418–477.

Bickel, Peter J., and Joseph A. Yahav. 1988. Richardson extrapolation and the bootstrap. *Journal of the American Statistical Association* 83:387–393.

Bindel, David, James Demmel, William Kahan, and Osni Marques. 2002. On computing Givens rotations reliably and efficiently. *ACM Transactions on Mathematical Software* 28:206–238.

Birkhoff, Garrett, and Surender Gulati. 1979. Isotropic distributions of test matrices. *Journal of Applied Mathematics and Physics (ZAMP)* 30:148–158.

Bischof, Christian H. 1990. Incremental condition estimation. *SIAM Journal of Matrix Analysis and Applications* 11:312–322.

Bischof, Christian H., and Gregorio Quintana-Ortí. 1998a. Computing rank-revealing QR factorizations. *ACM Transactions on Mathematical Software* 24:226–253.

Bischof, Christian H., and Gregorio Quintana-Ortí. 1998b. Algorithm 782: Codes for rank-revealing QR factorizations of dense matrices. *ACM Transactions on Mathematical Software* 24:254–257.

Björck, Åke. 1967. Solving least squares problems by Gram-Schmidt orthogonalization. *BIT* 7:1–21.

Björck, Åke. 1996. *Numerical Methods for Least Squares Problems*. Philadelphia: Society for Industrial and Applied Mathematics.

Blackford, L. S., J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. 1997a. *ScaLAPACK Users' Guide*. Philadelphia: Society for Industrial and Applied Mathematics.

Blackford, L. S., A. Cleary, A. Petitet, R. C. Whaley, J. Demmel, I. Dhillon, H. Ren, K. Stanley, J. Dongarra, and S. Hammarling. 1997b. Practical ex-

perience in the numerical dangers of heterogeneous computing. *ACM Transactions on Mathematical Software* 23:133–147.

Blackford, L. Susan, Antoine Petitet, Roldan Pozo, Karin Remington, R. Clint Whaley, James Demmel, Jack Dongarra, Iain Duff, Sven Hammarling, Greg Henry, Michael Heroux, Linda Kaufman, and Andrew Lumsdaine. 2002. An updated set of basic linear algebra subprograms (BLAS). *ACM Transactions on Mathematical Software* 28:135–151.

Bollobás, Béla. 2013. *Modern Graph Theory*. New York: Springer-Verlag.

Brown, Peter N., and Homer F. Walker. 1997. GMRES on (nearly) singular systems. *SIAM Journal of Matrix Analysis and Applications* 18: 37–51.

Bunch, James R., and Linda Kaufman. 1977. Some stable methods for calculating inertia and solving symmetric linear systems. *Mathematics of Computation* 31:163–179.

Buttari, Alfredo, Julien Langou, Jakub Kurzak, and Jack Dongarra. 2009. A class of parallel tiled linear algebra algorithms for multicore architectures. *Parallel Computing* 35:38–53.

Calvetti, Daniela. 1991. Roundoff error for floating point representation of real data. *Communications in Statistics* 20:2687–2695.

Campbell, S. L., and C. D. Meyer, Jr. 1991. *Generalized Inverses of Linear Transformations*. New York: Dover Publications, Inc.

Carmeli, Moshe. 1983. *Statistical Theory and Random Matrices*. New York: Marcel Dekker, Inc.

Chaitin-Chatelin, Françoise, and Valérie Frayssé. 1996. *Lectures on Finite Precision Computations*. Philadelphia: Society for Industrial and Applied Mathematics.

Chambers, John M. 2016. *Extending R*. Boca Raton: Chapman and Hall/CRC Press.

Chan, T. F. 1982a. An improved algorithm for computing the singular value decomposition. *ACM Transactions on Mathematical Software* 8:72–83.

Chan, T. F. 1982b. Algorithm 581: An improved algorithm for computing the singular value decomposition. *ACM Transactions on Mathematical Software* 8:84–88.

Chan, T. F., G. H. Golub, and R. J. LeVeque. 1982. Updating formulae and a pairwise algorithm for computing sample variances. In *Compstat 1982: Proceedings in Computational Statistics*, ed. H. Caussinus, P. Ettinger, and R. Tomassone, 30–41. Vienna: Physica-Verlag.

Chan, Tony F., Gene H. Golub, and Randall J. LeVeque. 1983. Algorithms for computing the sample variance: Analysis and recommendations. *The American Statistician* 37:242–247.

Chapman, Barbara, Gabriele Jost, and Ruud van der Pas. 2007. *Using OpenMP: Portable Shared Memory Parallel Programming*. Cambridge, Massachusetts: The MIT Press.

Cheng, John, Max Grossman, and Ty McKercher. 2014. *Professional CUDA C Programming*. New York: Wrox Press, an imprint of John Wiley and Sons.

Chu, Moody T. 1991. Least squares approximation by real normal matrices with specified spectrum. *SIAM Journal on Matrix Analysis and Applications* 12:115–127.

Čížková, Lenka, and Pavel Čížek. 2012. Numerical linear algebra. In *Handbook of Computational Statistics: Concepts and Methods*, 2nd revised and updated ed., ed. James E. Gentle, Wolfgang Härdle, and Yuichi Mori, 105–137. Berlin: Springer.

Clerman, Norman, and Walter Spector. 2012. *Modern Fortran*. Cambridge, United Kingdom: Cambridge University Press.

Cline, Alan K., Andrew R. Conn, and Charles F. Van Loan. 1982. Generalizing the LINPACK condition estimator. In *Numerical Analysis, Mexico, 1981*, ed. J. P. Hennart, 73–83. Berlin: Springer-Verlag.

Cline, A. K., C. B. Moler, G. W. Stewart, and J. H. Wilkinson. 1979. An estimate for the condition number of a matrix. *SIAM Journal of Numerical Analysis* 16:368–375.

Cline, A. K., and R. K. Rew. 1983. A set of counter-examples to three condition number estimators. *SIAM Journal on Scientific and Statistical Computing* 4:602–611.

Cody, W. J. 1988. Algorithm 665: MACHAR: A subroutine to dynamically determine machine parameters. *ACM Transactions on Mathematical Software* 14:303–329.

Cody, W. J., and Jerome T. Coonen. 1993. Algorithm 722: Functions to support the IEEE standard for binary floating-point arithmetic. *ACM Transactions on Mathematical Software* 19:443–451.

Coleman, Thomas F., and Charles Van Loan. 1988. *Handbook for Matrix Computations*. Philadelphia: Society for Industrial and Applied Mathematics.

Cragg, John G., and Stephen G. Donald. 1996. On the asymptotic properties of LDU-based tests of the rank of a matrix. *Journal of the American Statistical Association* 91:1301–1309.

Cullen, M. R. 1985. *Linear Models in Biology*. New York: Halsted Press.

Dauger, Dean E., and Viktor K. Decyk. 2005. Plug-and-play cluster computing: High-performance computing for the mainstream. *Computing in Science and Engineering* 07(2):27–33.

Davies, Philip I., and Nicholas J. Higham. 2000. Numerically stable generation of correlation matrices and their factors. *BIT* 40:640–651.

Dempster, Arthur P., and Donald B. Rubin. 1983. Rounding error in regression: The appropriateness of Sheppard's corrections. *Journal of the Royal Statistical Society, Series B* 39:1–38.

Devlin, Susan J., R. Gnanadesikan, and J. R. Kettenring. 1975. Robust estimation and outlier detection with correlation coefficients. *Biometrika* 62:531–546.

Dey, Aloke, and Rahul Mukerjee. 1999. *Fractional Factorial Plans*. New York: John Wiley and Sons.

Dongarra, J. J., J. R. Bunch, C. B. Moler, and G. W. Stewart. 1979. *LINPACK Users' Guide*. Philadelphia: Society for Industrial and Applied Mathematics.

Dongarra, J. J., J. DuCroz, S. Hammarling, and I. Duff. 1990. A set of level 3 basic linear algebra subprograms. *ACM Transactions on Mathematical Software* 16:1–17.

Dongarra, J. J., J. DuCroz, S. Hammarling, and R. J. Hanson. 1988. An extended set of Fortran basic linear algebra subprograms. *ACM Transactions on Mathematical Software* 14:1–17.

Dongarra, Jack J., and Victor Eijkhout. 2000. Numerical linear algebra algorithms and software. *Journal of Computational and Applied Mathematics* 123:489–514.

Draper, Norman R., and Harry Smith. 1998. *Applied Regression Analysis*, 3rd ed. New York: John Wiley and Sons.

Duff, Iain S., Michael A. Heroux, and Roldan Pozo. 2002. An overview of the sparse basic linear algebra subprograms: the new standard from the BLAS technical forum. *ACM Transactions on Mathematical Software* 28:239–267.

Duff, Iain S., Michele Marrone, Guideppe Radicati, and Carlo Vittoli. 1997. Level 3 basic linear algebra subprograms for sparse matrices: A user-level interface. *ACM Transactions on Mathematical Software* 23:379–401.

Duff, Iain S., and Christof Vömel. 2002. Algorithm 818: A reference model implementation of the sparse BLAS in Fortran 95. *ACM Transactions on Mathematical Software* 28:268–283.

Eckart, Carl, and Gale Young. 1936. The approximation of one matrix by another of lower rank. *Psychometrika* 1:211–218.

Eddelbuettel, Dirk. 2013. *Seamless R and C++ Integration with Rcpp*. New York: Springer-Verlag.

Ericksen, Wilhelm S. 1985. Inverse pairs of test matrices. *ACM Transactions on Mathematical Software* 11:302–304.

Efron, Bradley, Trevor Hastie, Iain Johnstone, and Robert Tibshirani. 2004. Least angle regression. *The Annals of Statistics* 32:407–499.

Escobar, Luis A., and E. Barry Moser. 1993. A note on the updating of regression estimates. *The American Statistician* 47:192–194.

Eskow, Elizabeth, and Robert B. Schnabel. 1991. Algorithm 695: Software for a new modified Cholesky factorization. *ACM Transactions on Mathematical Software* 17:306–312.

Eubank, Randall L., and Ana Kupresanin. 2012. *Statistical Computing in C++ and R*. Boca Raton: Chapman and Hall/CRC Press.

Fasino, Dario, and Luca Gemignani. 2003. A Lanczos-type algorithm for the QR factorization of Cauchy-like matrices. In *Fast Algorithms for Structured Matrices: Theory and Applications*, ed. Vadim Olshevsky, 91–104. Providence, Rhode Island: American Mathematical Society.

Filippone, Salvatore, and Michele Colajanni. 2000. PSBLAS: A library for parallel linear algebra computation on sparse matrices. *ACM Transactions on Mathematical Software* 26:527–550.

Fuller, Wayne A. 1995. *Introduction to Statistical Time Series*, 2nd ed. New York: John Wiley and Sons.

Galassi, Mark, Jim Davies, James Theiler, Brian Gough, Gerard Jungman, Michael Booth, and Fabrice Rossi. 2002. *GNU Scientific Library Reference Manual*, 2nd ed. Bristol, United Kingdom: Network Theory Limited.

Gandrud, Christopher. 2015. *Reproducible Research with R and R Studio*, 2nd ed. Boca Raton: Chapman and Hall/CRC Press.

Gantmacher, F. R. 1959. *The Theory of Matrices*, Volumes I and II, translated by K. A. Hirsch, Chelsea, New York.

Geist, Al, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, and Vaidy Sunderam. 1994. *PVM. Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing*. Cambridge, Massachusetts: The MIT Press.

Gentle, James E. 2003. *Random Number Generation and Monte Carlo Methods*, 2nd ed. New York: Springer-Verlag.

Gentle, James E. 2009. *Computational Statistics*. New York: Springer-Verlag.

Gentleman, W. M. 1974. Algorithm AS 75: Basic procedures for large, sparse or weighted linear least squares problems. *Applied Statistics* 23:448–454.

Gill, Len, and Arthur Lewbel. 1992. Testing the rank and definiteness of estimated matrices with applications to factor, state-space and ARMA models. *Journal of the American Statistical Association* 87:766–776.

Golub, G., and W. Kahan. 1965. Calculating the singular values and pseudo-inverse of a matrix. *SIAM Journal of Numerical Analysis, Series B* 2:205–224.

Golub, G. H., and C. Reinsch. 1970. Singular value decomposition and least squares solutions. *Numerische Mathematik* 14:403–420.

Golub, G. H., and C. F. Van Loan. 1980. An analysis of the total least squares problem. *SIAM Journal of Numerical Analysis* 17:883–893.

Golub, Gene H., and Charles F. Van Loan. 1996. *Matrix Computations*, 3rd ed. Baltimore: The Johns Hopkins Press.

Graybill, Franklin A. 1983. *Introduction to Matrices with Applications in Statistics*, 2nd ed. Belmont, California: Wadsworth Publishing Company.

Greenbaum, Anne, and Zdeněk Strakoš. 1992. Predicting the behavior of finite precision Lanczos and conjugate gradient computations. *SIAM Journal for Matrix Analysis and Applications* 13:121–137.

Gregory, Robert T., and David L. Karney. 1969. *A Collection of Matrices for Testing Computational Algorithms*. New York: John Wiley and Sons.

Gregory, R. T., and E. V. Krishnamurthy. 1984. *Methods and Applications of Error-Free Computation*. New York: Springer-Verlag.

Grewal, Mohinder S., and Angus P. Andrews. 1993. *Kalman Filtering Theory and Practice*. Englewood Cliffs, New Jersey: Prentice-Hall.

Griva, Igor, Stephen G. Nash, and Ariela Sofer. 2009. *Linear and Nonlinear Optimization*, 2nd ed. Philadelphia: Society for Industrial and Applied Mathematics.

Gropp, William D. 2005. Issues in accurate and reliable use of parallel computing in numerical programs. In *Accuracy and Reliability in Scientific Computing*, ed. Bo Einarsson, 253–263. Philadelphia: Society for Industrial and Applied Mathematics.

Gropp, William, Ewing Lusk, and Anthony Skjellum. 2014. *Using MPI: Portable Parallel Programming with the Message-Passing Interface*, 3rd ed. Cambridge, Massachusetts: The MIT Press.

Gropp, William, Ewing Lusk, and Thomas Sterling (Editors). 2003. *Beowulf Cluster Computing with Linux*, 2nd ed. Cambridge, Massachusetts: The MIT Press.

Haag, J. B., and D. S. Watkins. 1993. QR-like algorithms for the nonsymmetric eigenvalue problem. *ACM Transactions on Mathematical Software* 19:407–418.

Hager, W. W. 1984. Condition estimates. *SIAM Journal on Scientific and Statistical Computing* 5:311–316.

Hanson, Richard J., and Tim Hopkins. 2013. *Numerical Computing with Modern Fortran*. Philadelphia: Society for Industrial and Applied Mathematics.

Harville, David A. 1997. *Matrix Algebra from a Statistician's Point of View*. New York: Springer-Verlag.

Heath, M. T., E. Ng, and B. W. Peyton. 1991. Parallel algorithms for sparse linear systems. *SIAM Review* 33:420–460.

Hedayat, A. S., N. J. A. Sloane, and John Stufken. 1999. *Orthogonal Arrays: Theory and Applications*. New York: Springer-Verlag.

Heiberger, Richard M. 1978. Algorithm AS127: Generation of random orthogonal matrices. *Applied Statistics* 27:199–205.

Heroux, Michael A. 2015. Editorial: ACM TOMS replicated computational results initiative. *ACM Transactions on Mathematical Software* 41:Article No. 13.

Higham, Nicholas J. 1987. A survey of condition number estimation for triangular matrices. *SIAM Review* 29:575–596.

Higham, Nicholas J. 1988. FORTRAN codes for estimating the one-norm of a real or complex matrix, with applications to condition estimation. *ACM Transactions on Mathematical Software* 14:381–386.

Higham, Nicholas J. 1990. Experience with a matrix norm estimator. *SIAM Journal on Scientific and Statistical Computing* 11:804–809.

Higham, Nicholas J. 1991. Algorithm 694: A collection of test matrices in Matlab. *ACM Transactions on Mathematical Software* 17:289–305.

Higham, Nicholas J. 1997. Stability of the diagonal pivoting method with partial pivoting. *SIAM Journal of Matrix Analysis and Applications* 18:52–65.

Higham, Nicholas J. 2002. *Accuracy and Stability of Numerical Algorithms*, 2nd ed. Philadelphia: Society for Industrial and Applied Mathematics.

Higham, Nicholas J. 2008. *Functions of Matrices. Theory and Computation*. Philadelphia: Society for Industrial and Applied Mathematics.

Hill, Francis S., Jr., and Stephen M Kelley. 2006. *Computer Graphics Using OpenGL*, 3rd ed. New York: Pearson Education.

Hoffman, A. J., and H. W. Wielandt. 1953. The variation of the spectrum of a normal matrix. *Duke Mathematical Journal* 20:37–39.

Hong, H. P., and C. T. Pan. 1992. Rank-revealing QR factorization and SVD. *Mathematics of Computation* 58:213–232.

Horn, Roger A., and Charles R. Johnson. 1991. *Topics in Matrix Analysis*. Cambridge, United Kingdom: Cambridge University Press.

IEEE. 2008. *IEEE Standard for Floating-Point Arithmetic, Std 754-2008*. New York: IEEE, Inc.

Jansen, Paul, and Peter Weidner. 1986. High-accuracy arithmetic software — some tests of the ACRITH problem-solving routines. *ACM Transactions on Mathematical Software* 12:62–70.

Jaulin, Luc, Michel Kieffer, Olivier Didrit, and Eric Walter. (2001). *Applied Interval Analysis*. New York: Springer.

Jolliffe, I. T. 2002. *Principal Component Analysis*, 2nd ed. New York: Springer-Verlag.

Karau, Holden, Andy Konwinski, Patrick Wendell, and Matei Zaharia. 2015. *Learning Spark*. Sabastopol, California: O'Reilly Media, Inc.

Kearfott, R. Baker. 1996. INTERVAL_ARITHMETIC: A Fortran 90 module for an interval data type. *ACM Transactions on Mathematical Software* 22:385–392.

Kearfott, R. Baker, and Vladik Kreinovich (Editors). 1996. *Applications of Interval Computations*. Netherlands: Kluwer, Dordrecht.

Kearfott, R. B., M. Dawande, K. Du, and C. Hu. 1994. Algorithm 737: INTLIB: A portable Fortran 77 interval standard-function library. *ACM Transactions on Mathematical Software* 20:447–459.

Keller-McNulty, Sallie, and W. J. Kennedy. 1986. An error-free generalized matrix inversion and linear least squares method based on bordering. *Communications in Statistics — Simulation and Computation* 15:769–785.

Kennedy, William J., and James E. Gentle. 1980. *Statistical Computing*. New York: Marcel Dekker, Inc.

Kenney, C. S., and A. J. Laub. 1994. Small-sample statistical condition estimates for general matrix functions. *SIAM Journal on Scientific Computing* 15:191–209.

Kenney, C. S., A. J. Laub, and M. S. Reese. 1998. Statistical condition estimation for linear systems. *SIAM Journal on Scientific Computing* 19:566–583.

Kim, Hyunsoo, and Haesun Park. 2008. Nonnegative matrix factorization based on alternating non-negativity-constrained least squares and the active set method. *SIAM Journal on Matrix Analysis and Applications* 30:713–730.

Kleibergen, Frank, and Richard Paap. 2006. Generalized reduced rank tests using the singular value decomposition. *Journal of Econometrics* 133:97–126.

Kollo, Tõnu, and Dietrich von Rosen. 2005. *Advanced Multivariate Statistics with Matrices*. Amsterdam: Springer.

Kshemkalyani, Ajay D., and Mukesh Singhal. 2011. *Distributed Computing: Principles, Algorithms, and Systems*. Cambridge, United Kingdom: Cambridge University Press.

Kulisch, Ulrich. 2011. Very fast and exact accumulation of products. *Computing* 91:397–405.

Lawson, C. L., R. J. Hanson, D. R. Kincaid, and F. T. Krogh. 1979. Basic linear algebra subprograms for Fortran usage. *ACM Transactions on Mathematical Software* 5:308–323.

Lee, Daniel D., and H. Sebastian Seung. 2001. Algorithms for non-negative matrix factorization. *Advances in Neural Information Processing Systems*, 556–562. Cambridge, Massachusetts: The MIT Press.

Lemmon, David R., and Joseph L. Schafer. 2005. *Developing Statistical Software in Fortran 95*. New York: Springer-Verlag.

Leskovec, Jure, Anand Rajaraman, and Jeffrey David Ullman. 2014. *Mining of Massive Datasets*, 2nd ed. Cambridge, United Kingdom: Cambridge University Press.

Levesque, John, and Gene Wagenbreth. 2010. *High Performance Computing: Programming and Applications*. Boca Raton: Chapman and Hall/CRC Press.

Liem, C. B., T. Lü, and T. M. Shih. 1995. *The Splitting Extrapolation Method*. Singapore: World Scientific.

Linnainmaa, Seppo. 1975. Towards accurate statistical estimation of rounding errors in floating-point computations. *BIT* 15:165–173.

Liu, Shuangzhe and Heinz Neudecker. 1996. Several matrix Kantorovich-type inequalities. *Journal of Mathematical Analysis and Applications* 197:23–26.

Loader, Catherine. 2012. Smoothing: Local regression techniques. In *Handbook of Computational Statistics: Concepts and Methods*, 2nd revised and updated ed., ed. James E. Gentle, Wolfgang Härdle, and Yuichi Mori, 571–596. Berlin: Springer.

Longley, James W. 1967. An appraisal of least squares problems for the electronic computer from the point of view of the user. *Journal of the American Statistical Association* 62:819–841.

Luk, F. T., and H. Park. 1989. On parallel Jacobi orderings. *SIAM Journal on Scientific and Statistical Computing* 10:18–26.

Magnus, Jan R., and Heinz Neudecker. 1999. *Matrix Differential Calculus with Applications in Statistics and Econometrics*, revised ed. New York: John Wiley and Sons.

Markus, Arjen. 2012. *Modern Fortran in Practice*. Cambridge, United Kingdom: Cambridge University Press.

Marshall, A. W., and I. Olkin. 1990. Matrix versions of the Cauchy and Kantorovich inequalities. *Aequationes Mathematicae* 40:89–93.

Metcalf, Michael, John Reid, and Malcolm Cohen. 2011. *Modern Fortran Explained*. Oxford, United Kingdom: Oxford University Press.

Meyn, Sean, and Richard L. Tweedie. 2009. *Markov Chains and Stochastic Stability*, 2nd ed. Cambridge, United Kingdom: Cambridge University Press.

Miller, Alan J. 1992. Algorithm AS 274: Least squares routines to supplement those of Gentleman. *Applied Statistics* 41:458–478 (Corrections, 1994, *ibid.* 43:678).

Miller, Alan. 2002. *Subset Selection in Regression*, 2nd ed. Boca Raton: Chapman and Hall/CRC Press.

Miller, Alan J., and Nam-Ky Nguyen. 1994. A Fedorov exchange algorithm for D-optimal design. *Applied Statistics* 43:669–678.

Mizuta, Masahiro. 2012. Dimension reduction methods. In *Handbook of Computational Statistics: Concepts and Methods*, 2nd revised and updated ed., ed. James E. Gentle, Wolfgang Härdle, and Yuichi Mori, 619–644. Berlin: Springer.

Moore, E. H. 1920. On the reciprocal of the general algebraic matrix. *Bulletin of the American Mathematical Society* 26:394–395.

Moore, Ramon E. (1979). *Methods and Applications of Interval Analysis*. Philadelphia: Society for Industrial and Applied Mathematics.

Mosteller, Frederick, and David L. Wallace. 1963. Inference in an authorship problem. *Journal of the American Statistical Association* 58:275–309.

Muirhead, Robb J. 1982. *Aspects of Multivariate Statistical Theory*. New York: John Wiley and Sons.

Mullet, Gary M., and Tracy W. Murray. 1971. A new method for examining rounding error in least-squares regression computer programs. *Journal of the American Statistical Association* 66:496–498.

Nachbin, Leopoldo. 1965. *The Haar Integral*, translated by Lulu Bechtolsheim. Princeton, New Jersey: D. Van Nostrand Co Inc.

Nakano, Junji. 2012. Parallel computing techniques. In *Handbook of Computational Statistics: Concepts and Methods*, 2nd revised and updated ed., ed. James E. Gentle, Wolfgang Härdle, and Yuichi Mori, 243–272. Berlin: Springer.

Nguyen, Nam-Ky, and Alan J. Miller. 1992. A review of some exchange algorithms for constructing D-optimal designs. *Computational Statistics and Data Analysis* 14:489–498.

Olshevsky, Vadim (Editor). 2003. *Fast Algorithms for Structured Matrices: Theory and Applications*. Providence, Rhode Island: American Mathematical Society.

Olver, Frank W. J., Daniel w. Lozier, Ronald F. Boisvert, and Charles W. Clark. 2010. *NIST Handbook of Mathematical Functions*. Cambridge: Cambridge University Press.

Overton, Michael L. 2001. *Numerical Computing with IEEE Floating Point Arithmetic*. Philadelphia: Society for Industrial and Applied Mathematics.

Parsian, Mahmoud. 2015. *Data Algorithms*. Sabastopol, California: O'Reilly Media, Inc.

Penrose, R. 1955. A generalized inverse for matrices. *Proceedings of the Cambridge Philosophical Society* 51:406–413.

Quinn, Michael J. 2003. *Parallel Programming in C with MPI and OpenMP*. New York: McGraw-Hill.

Rice, John R. 1966. Experiments on Gram-Schmidt orthogonalization. *Mathematics of Computation* 20:325–328.

Rice, John R. 1993. *Numerical Methods, Software, and Analysis*, 2nd ed. New York: McGraw-Hill Book Company.

Robin, J. M., and R. J. Smith. 2000. Tests of rank. *Econometric Theory* 16:151–175.

Roosta, Seyed H. 2000. *Parallel Processing and Parallel Algorithms: Theory and Computation*. New York: Springer-Verlag.

Rousseeuw, Peter J., and Geert Molenberghs. 1993. Transformation of nonpositive semidefinite correlation matrices. *Communications in Statistics — Theory and Methods* 22:965–984.

Rust, Bert W. 1994. Perturbation bounds for linear regression problems. *Computing Science and Statistics* 26:528–532.

Saad, Y., and M. H. Schultz. 1986. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing* 7:856–869.

Schott, James R. 2004. *Matrix Analysis for Statistics*, 2nd ed. New York: John Wiley and Sons.

Searle, S. R. 1971. *Linear Models*. New York: John Wiley and Sons.

Searle, Shayle R. 1982. *Matrix Algebra Useful for Statistics*. New York: John Wiley and Sons.

Shao, Jun. 2003. *Mathematical Statistics*, 2nd ed. New York: Springer-Verlag.

Sherman, J., and W. J. Morrison. 1950. Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. *Annals of Mathematical Statistics* 21:124–127.

Siek, Jeremy, and Andrew Lumsdaine. 2000. A modern framework for portable high-performance numerical linear algebra. In *Advances in Software Tools for Scientific Computing*, ed. Are Bruaset, H. Langtangen, and E. Quak, 1–56. New York: Springer-Verlag.

Skeel, R. D. 1980. Iterative refinement implies numerical stability for Gaussian elimination. *Mathematics of Computation* 35:817–832.

Smith, B. T., J. M. Boyle, J. J. Dongarra, B. S. Garbow, Y. Ikebe, V. C. Klema, and C. B. Moler. 1976. *Matrix Eigensystem Routines — EISPACK Guide*. Berlin: Springer-Verlag.

Stallings, W. T., and T. L. Boullion. 1972. Computation of pseudo-inverse using residue arithmetic. *SIAM Review* 14:152–163.

Stewart, G. W. 1980. The efficient generation of random orthogonal matrices with an application to condition estimators. *SIAM Journal of Numerical Analysis* 17:403–409.

Stewart, G. W. 1990. Stochastic perturbation theory. *SIAM Review* 32:579–610.

Stodden, Victoria, Friedrich Leisch, and Roger D. Peng. 2014. *Implementing Reproducible Research*. Boca Raton: Chapman and Hall/CRC Press.

Strang, Gilbert, and Tri Nguyen. 2004. The interplay of ranks of submatrices. *SIAM Review* 46:637–646.

Strassen, V. 1969, Gaussian elimination is not optimal. *Numerische Mathematik* 13:354–356.

Szabó, S., and R. Tanaka. 1967. *Residue Arithmetic and Its Application to Computer Technology*. New York: McGraw-Hill.

Tanner, M. A., and R. A. Thisted. 1982. A remark on AS127. Generation of random orthogonal matrices. *Applied Statistics* 31:190–192.

Titterington, D. M. 1975. Optimal design: Some geometrical aspects of *D*-optimality. *Biometrika* 62:313–320.

Trefethen, Lloyd N., and Mark Embree. 2005. *Spectra and Pseudospectra: The Behavior of Nonnormal Matrices and Operators*. Princeton: Princeton University Press.

Trefethen, Lloyd N., and David Bau III. 1997. *Numerical Linear Algebra*. Philadelphia: Society for Industrial and Applied Mathematics.

Trosset, Michael W. 2002. Extensions of classical multidimensional scaling via variable reduction. *Computational Statistics* 17:147–163.

Unicode Consortium. 1990. *The Unicode Standard, Worldwide Character Encoding, Version 1.0, Volume 1*. Reading, Massachusetts: Addison-Wesley Publishing Company.

Unicode Consortium. 1992. *The Unicode Standard, Worldwide Character Encoding, Version 1.0, Volume 2*. Reading, Massachusetts: Addison-Wesley Publishing Company.

Vandenberghe, Lieven, and Stephen Boyd. 1996. Semidefinite programming. *SIAM Review* 38:49–95.

Venables, W. N., and B. D. Ripley. 2003. *Modern Applied Statistics with S*, 4th ed. New York: Springer-Verlag.

Walker, Homer F. 1988. Implementation of the GMRES method using Householder transformations. *SIAM Journal on Scientific and Statistical Computing* 9:152–163.

Walker, Homer F., and Lu Zhou. 1994. A simpler GMRES. *Numerical Linear Algebra with Applications* 1:571–581.

Walster, G. William. 1996. Stimulating hardware and software support for interval arithmetic. In *Applications of Interval Computations*, ed. R. Baker Kearfott and Vladik Kreinovich, 405–416. Dordrecht, Netherlands: Kluwer.

Walster, G. William. 2005. The use and implementation of interval data types. In *Accuracy and Reliability in Scientific Computing*, ed. Bo Einarsson, 173–194. Philadelphia: Society for Industrial and Applied Mathematics.

Watkins, David S. 2002. *Fundamentals of Matrix Computations*, 2nd ed. New York: John Wiley and Sons.

White, Tom. 2015. *Hadoop: The Definitive Guide*, 4th ed. Sabastopol, California: O'Reilly Media, Inc.

Wickham, Hadley. 2015) *Advanced R*. Boca Raton: Chapman and Hall/CRC Press.

Wilkinson, J. H. 1959. The evaluation of the zeros of ill-conditioned polynomials. *Numerische Mathematik* 1:150–180.

Wilkinson, J. H. 1963. *Rounding Errors in Algebraic Processes*. Englewood Cliffs, New Jersey: Prentice-Hall. (Reprinted by Dover Publications, Inc., New York, 1994).

Wilkinson, J. H. 1965. *The Algebraic Eigenvalue Problem*. New York: Oxford University Press.

Woodbury, M. A. 1950. "Inverting Modified Matrices", Memorandum Report 42, Statistical Research Group, Princeton University.

Wynn, P. 1962. Acceleration techniques for iterated vector and matrix problems. *Mathematics of Computation* 16:301–322.

Xie, Yihui. 2015. *Dynamic Documents with R and knitr*, 2nd ed. Boca Raton: Chapman and Hall/CRC Press.

Zhou, Bing Bing, and Richard P. Brent. 2003. An efficient method for computing eigenvalues of a real normal matrix. *Journal of Parallel and Distributed Computing* 63:638–648.

Index

J.E. Gentle, *Matrix Algebra*, Springer Texts in Statistics,
DOI 10.1007/978-3-319-64867-5