- round off errors
- stability
- condition
- accuracy

PREFACE

Numerical linear algebra, no longer a subtopic of numerical analysis, has grown into an independent topic for research and teaching. Because it is crucial to scientific computing (a major component of modern applied and engineering research), numerical linear algebra is becoming integral to courses in mathematics, computer science, and engineering.

Numerical Linear Algebra and Applications can be used in both undergraduate and beginning graduate courses, as well as by working scientists and engineers for self-study and reference. I and other colleagues have class-tested the contents. Far more material than can be covered in one semester has been included, so teachers can tailor material to particular classes and easily develop course syllabi.

To promote undergraduate education in numerical linear algebra is my primary objective. I want new students who use this book to develop a firm grasp of the concepts of roundoff errors, stability, condition, and accuracy; they should also appreciate the core algorithms, their usefulness, and implementations. To this end I have developed the following special features.

Special Features

Applications. Whenever possible, computational problems are accompanied by real-life examples drawn from numerous disciplines, including heat transfer, fluid dynamics, signal processing, biomedical engineering, statistics, business, bioscience, and control theory, and vibration engineering. When a physical problem is posed, the physical and engineering significance of the solution is considered. The applications make the text appropriate for courses in applied linear algebra, as well.

Recent research. Advanced students in particular will benefit from the up-to-date discussions of the concepts of strong, weak, and mild stability of algorithms; counterexamples of a long-standing conjecture on growth factor; componentwise perturba-

tion analysis of linear system problems; Krylov subspace methods for linear systems and eigenvalue problems; equivalence of Householder and Gram-Schmidt processes for least-squares problems; the use of modified Gram-Schmidt process to solve the undetermined systems in a numerically stable way; the divide-and-conquer algorithm for symmetric eigenvalue problems; and the zero-shift algorithm for the SVD problem.

Clear explanations of basic concepts. Very early in the text I introduce the concepts of conditioning of the problem and the stability of an algorithm via backward roundoff error analysis. Simple motivating examples are also included. The influence of these concepts on specific results is demonstrated clearly.

Fundamental linear algebra problems. In Chapter 0 fundamental computational linear algebra problems are stated along with a brief mention of the importance of each. The difficulties of solving these problems in a computational setting using "obvious approaches" are then illustrated. This chapter serves as a motivating chapter for the entire book.

Extensive discussion of fundamental tools. Throughout the text I show how algorithms can be made cost-effective and efficient by using the rich structures of Elementary, Householder, and Givens matrices. Chapter 5 is devoted to them. The ability to introduce zeros in specific positions of a vector or matrix is a simple but very powerful property of these matrices. How it can be exploited to develop algorithms for useful matrix factorizations such as LU and QR and for reduction of a matrix to a simple form such as Hessenberg is clearly explained.

Step-by-step explanation of the algorithms. Before an algorithm is introduced I discuss its basic ideas and underlying principles. I describe the first few steps in detailed elementary terms and then show how the kth step follows the established pattern. Students may then be able to reproduce some of the well-known algorithms even before learning them in class.

Numerically-effective algorithms and software. In Chapter 4, I explain what makes algorithms and associated software numerically effective: efficiency, numerical stability, storage economy, and so on. Students often mistake an algorithm that is only efficient for a "good" algorithm, not understanding that it may also be highly unstable (for example, Gaussian elimination without pivoting). Here and in other chapters they will learn not just how to distinguish among algorithms but also how to transform a "bad" one into a "good" one whenever possible.

MATLAB® and MATCOM. I believe that teachers of numerical linear algebra should use MATLAB if they can. It is becoming increasingly popular in all areas of scientific and engineering computing. I have provided MATLAB codes for some of the basic algorithms so students can use them as templates for writing codes for more advanced algorithms. The MATLAB toolkit, MATCOM, will enable students to study different algorithms for the same problem, comparing efficiency, stability, and accuracy. (See the information sheet at the end of this book.)

Solutions and answers to selected problems. Especially for the benefit of undergraduates, I have included partial solutions and answers for important problems, emphasizing those that need proofs.

List of key terms. The common terms in numerical linear algebra are listed for quick reference.

Acknowledgments

This book would not exist in its present form without the help, guidance, suggestions, and encouragement of many friends and colleagues around the world.

My deepest gratitude is to the contributors: Professors James R. Bunch of the University of California, San Diego; M. Benbourenane, Y. P. Hong, Abhijit Gupta, and Pradip Majumdar of Northern Illinois University; Michael Berry of the University of Tennessee; Zhonggang Zeng of Northeastern Illinois University; and Dr. Daniel Pierce of Boeing Computing Services. Professors Bunch and Zeng and Dr. Pierce read and re-read the different versions of the manuscript, made constructive criticisms, gave helpful suggestions for improvements, and occasionally contributed to the writing. Furthermore, Professor Bunch taught an undergraduate numerical linear algebra course at the University of California at San Diego out of a semi-final verson of the book and his valuable feedback led to important improvements. I am deeply grateful to Professor Bunch for all his efforts. Professor Hong carefully and patiently checked many of the theoretical linear algebra theorems in the book, and my colleagues Professors Benbournane, Gupta, and Majumdar not only helped write the sections on engineering applications but also proofread the different versions.

The following people read certain chapters carefully and thoroughly, pointed out errors, and gave comments and suggestions for improvements; I am grateful to each of them. Professors Mark Arnold of the University of Arkansas, Z. Bai of the University of Kentucky; Michael Berry of the University of Tennessee; Ake Björck of Linkoping University, Sweden; K. B. Datta of Indian Institute of Technology, Khargpur, India; Karabi Datta of Northern Illinois University; Bart DeMoor of Katholieke Universiteit, Leuven, Belgium; Daniel Inman of Virginia Polytechnic and State University; Mauro Leoncini, of the University of Pisa, Italy; Beresford Parlett of the University of California, Berkeley; Yitshak Ram of the University of Adelaide, Australia; Axel Ruhe of Chalmers University of Technology, Sweden; Youcef Saad of the University of Minnesota; Gilbert Strang of Massachusetts Institute of Technology; and Paul Van Dooren of Center for Systems Engineering and Applied Mechanics in Belgium. Professor Faisal Saied of the University of Illinois, in his lecture on numerical linear algebra education at the 1994 SIAM Applied Linear Algebra meeting in Snowbird, Utah, made comments about this book that were highly encouraging and stimulating for me.

In addition to Professors Bunch and Zeng and Dr. Pierce, the following reviewers deserve my special thanks and gratitude for providing me with constructive criticism and very helpful guidance: Professors Daniel Boley of the University of Minnesota; Alan Edelman of the Massachusetts Institute of Technology; Floyd Han-

son of the University of Illinois at Chicago; Rajnikant Patel of Concordia University, Montreal; and Simon Tavener of Pennsylvania State University.

My present and former graduate students Mark Arnold, Samar Choudhary, Conrad Fernandes, Weiyao He, C. Hetti, Ishita Majumdar, and Avijit Purkayastha all played a role in the project by proofreading, problem solving, and typing. My special appreciation goes to Conrad Fernandes and Weiyao He. Conrad wrote the M-files for the MATLAB-based software MATCOM, and Weiyao proofread each chapter carefully, occasionally fixed up some "bugs" in the algorithms, and helped prepare the partial solutions for selected problems.

Great appreciation is also due to the undergraduate and graduate students at Northern Illinois University and at the University of California at San Diego, who used unpublished versions of the book in their first course in numerical linear algebra. Their constructive criticism helped me revise certain material.

My colleagues Professors William Blair, John Beachy, Linda Sons, and Jindřich Nečas, who are textbook authors themselves, occasionally listened to my frustrations with great patience, shared their experience, and gave me moral support and encouragement.

The present and the past office crew of the Mathematical Sciences Department at Northern Illinois University contributed a great deal by typing the many different versions for the book and providing timely assistance with other aspects of the manuscript preparation. Thank you, Lisa Allison, Elizabeth Buck, Sara Clayton, Mari-Anne Hartig, Jennifer Jacobson, Paula Propst, and Michael Smith. Sara Clayton and Mari-Anne Hartig did most of the typing. Mari-Anne's ingenious LaTeX skill can be seen everywhere in the book; she was the architect of the manuscript. Thanks are also due to Mr. Eric Behr, our computer systems manager, for his frequent help.

This book would probably never have been completed without the moral support, understanding, and continuous encouragement from the members of my family: Karabi, my wife, who is also a mathematician; Rajarshi, my son, who is now a college freshman at the University of Illinois at Urbana-Champaign; Rakhi, my daughter, who is a high school student in DeKalb; my nephew Sudeep, who is now a junior at the University of Texas at Austin; and my father-in-law and mother-in-law, Mr. R. L. Sarkar and Mrs. Belarani Sarkar, who live in India but weekly discussed my progress and gave constant encouragement and blessing over the telephone. Obviously, my duties as a husband and as a parent were very much neglected; furthermore, my wife and the children had to put up with my occasional frustrations over the past few years. I am lucky to have such a supportive family.

I am also thankful to the staff at Brooks/Cole Publishing Company. Gary Ostedt, the acquisition editor; Carol Benedict, the editorial assistant; Nancy Shammas, the production editor; John Kimmel (now at Chapman & Hall) who originally acquired the book; Linda Thompson and Barbara Kimmel, copy editors; and Adrienne Carter, the advertising coordinator. These people deserve special mention for their interest, unfailing assistance, and enormous patience. It was a joy to work with them.

Finally, I am indebted to the authors of several valuable books in the field, in particular to Professors Ake Björck, George Forsythe, Gene H. Golub, R. J. Hanson, C. L. Lawson, M. A. Malcolm, Cleve Moler, James Ortega, Beresford Parlett, G. W. Stewart, Charles Van Loan, Richard Varga, James H. Wilkinson, and David Young.

Their influence is clearly visible in many parts of my book. I never would have thought of starting the project had MATLAB not existed; a very special thanks to Dr. Cleve Moler.

I shall remain grateful to any reader kind enough to send his or her comments and suggestions or to point out errors in the book. I can be reached by e-mail: dattab@math.niu.edu. Any corrections or suggestions on the educational software package MATCOM may also be sent to me.

Biswa Nath Datta

Some Suggestions for Using the Book

1. A first course in Numerical Linear Algebra (undergraduate/one semester): Chapters 0 through 4, 5.1–5.4, 5.6, 6.2, 6.3 (some selected applications), 6.4, 6.5.1, 6.5.3, 6.5.4, 6.6.1, 6.7, 6.9, 6.10.1–6.10.4, 7.2–7.6, 7.8.1, 7.8.2, 8.2, some selected applications from 8.3, 8.4, 8.5.3, 8.6, 8.7.1, 8.8, 8.9.1–8.9.3, 10.2, 10.3–10.6, 10.8. (For a one-quarter course, some adjustments have to be made, for example, the instructor might wish to omit sections 6.10.1–6.10.4).
2. A second course in Numerical Linear Algebra (Beginning Graduate Course): 5.5–5.8, 6.6, 6.8, 6.10.5, 6.10.6, 7.7, 7.8.3, 7.8.4, 7.9, 7.10, 7.11, 8.7–8.9, 8.11, 8.12, 9.2–9.5, chapter 10, and chapter 11, along with a review of material of the first course, as necessary.
3. A Course in Applied Linear Algebra: An advanced undergraduate/beginning graduate course in Applied Linear Algebra can be designed by focusing on the applications parts of the book (6.3, 7.2, 8.3, 9.6, 9.7, 10.4) and doing relevant theory, as needed.
4. A One-Semester Course in Numerical Linear Algebra for Engineers: Chapters 0 through 4, 5.1–5.4, 6.2–6.5, 6.6.3, 6.7.1–6.7.2, 6.9, 6.10, 7.1–7.6, 7.8.1, 7.8.2, 8.2–8.4, 8.5, 8.6, 8.7.1, 8.9.1–8.9.6, 9.4–9.7, 9.9, 10.2–10.8, 10.9.1–10.9.2.

CONTENTS

CHAPTER ZERO

Numerical Linear Algebra Problems: Their Importance and Computational Difficulties 1

0.1 Introduction 1
0.2 Numerical Linear Algebra Problems and Their Importance 1
0.3 Solving Numerical Linear Algebra Problems Using Obvious Approaches 3

CHAPTER ONE

A Review of Required Linear Algebra Concepts 7

1.1 Introduction 7
1.2 Vectors 7
1.2.1 Orthogonality of Vectors and Subspaces 8
1.3 Matrices 9
1.3.1 Basic Concepts 9
1.3.2 Range and Null Space 12
1.3.3 Rank of a Matrix 13
1.3.4 The Inverse of a Matrix 13
1.3.5 Similar Matrices 14
1.3.6 Orthogonality and Projections 14
1.3.7 Projection of a Vector onto the Range and the Null Space of a Matrix 16
1.4 Some Special Matrices 17
1.4.1 Diagonal and Triangular Matrices 17
1.4.2 Orthogonal Matrix 17

1.4.3 Permutation Matrix 18
1.4.4 Hessenberg (Almost Triangular) Matrix 19
1.4.5 Companion Matrix 20
1.4.6 Nonderogatory Matrix 21
1.4.7 Diagonally Dominant Matrix 21
1.4.8 Positive Definite Matrix 21
1.5 The Cayley-Hamilton Theorem 23
1.6 Singular Values 23
1.7 Vector and Matrix Norms 24
1.7.1 Vector Norms 24
1.7.2 Matrix Norms 26
1.7.3 Convergence of a Matrix Sequence and Convergent Matrices 29
1.7.4 Norms and Inverses 32
1.8 Norm Invariant Properties of Orthogonal Matrices 34
1.9 Review and Summary 35
Suggestions for Further Reading 41

CHAPTER TWO

Floating-Point Numbers and Errors in Computations 43

2.1 Floating-Point Number Systems 43
2.2 Rounding Errors 46
2.3 Laws of Floating-Point Arithmetic 48
2.4 Addition of n Floating-Point Numbers 51
2.5 Multiplication of n Floating-Point Numbers 52
2.6 Inner Product Computation 53
2.7 Error Bounds for Floating-Point Matrix Computations 55
2.8 Roundoff Errors Due to Cancellation and Recursive Computations 57
2.9 Review and Summary 61
Suggestions for Further Reading 65

CHAPTER THREE

Stability of Algorithms and Conditioning of Problems 67

3.1 Some Basic Algorithms 67
3.1.1 Computing the Norm of a Vector 67
3.1.2 Computing the Inner Product of Two Vectors 68
3.1.3 Solution of an Upper Triangular System 68
3.1.4 Computing the Inverse of an Upper Triangular Matrix 69
3.1.5 Gaussian Elimination for Solving $Ax = b$ 71
3.2 Definitions and Concepts of Stability 75

3.4 Conditioning of the Problem, Stability of the Algorithm, and Accuracy of the Solution 79
3.5 The Wilkinson Polynomial 81
3.6 An Ill-Conditioned Linear System Problem 83
3.7 Examples of Ill-Conditioned Eigenvalue Problems 83
3.8 Strong, Weak, and Mild Stability 85
3.9 Review and Summary 87
Suggestions for Further Reading 90

CHAPTER FOUR

Numerically Effective Algorithms and Mathematical Software 91

4.1 Definitions and Examples 91
4.2 Flop Count and Storage Considerations for Some Basic Algorithms 93
4.3 Some Existing High-Quality Mathematical Software for Linear Algebra Problems 100
4.3.1 LINPACK 100
4.3.2 EISPACK 100
4.3.3 LAPACK 101
4.3.4 NETLIB 102
4.3.5 NAG 102
4.3.6 IMSL 103
4.3.7 MATLAB 103
4.3.8 MATLAB Codes and MATCOM 103
4.3.9 The ACM Library 103
4.3.10 ITPACK (Iterative Software Package) 103
4.3.11 The Software Package TEMPLATES (for the Solution of Linear Systems: Building Blocks for Iterative Methods) 104
4.3.12 SPARSKIT: A Basic Toolkit for Sparse Matrix Computations 104
4.4 Review and Summary 104
Suggestions for Further Reading 108

CHAPTER FIVE

Some Useful Transformations in Numerical Linear Algebra and Their Applications 110

5.1 A Computational Methodology in Numerical Linear Algebra 110
5.2 Elementary Matrices and LU Factorization 111
5.2.1 Gaussian Elimination without Pivoting 112
5.2.2 Gaussian Elimination with Partial Pivoting 120

5.3 Stability of Gaussian Elimination 131
5.4 Householder Transformations and Applications to QR Factorization and Hessenberg Reduction 133
5.4.1 Householder Matrices and QR Factorization 137
5.4.2 Householder QR Factorization of a Nonsquare Matrix 145
5.4.3 Householder Matrices and Reduction to Hessenberg Form 146
5.5 Givens Matrices and Applications to QR Factorization and Hessenberg Reduction 153
5.5.1 Givens Rotations and QR Factorization 159
5.5.2 Uniqueness in QR Factorization 161
5.5.3 Givens Rotations and Reduction to Hessenberg Form 163
5.5.4 Uniqueness in Hessenberg Reduction 165
5.6 Orthonormal Bases and Orthogonal Projections 166
5.7 QR Factorization with Column Pivoting 169
5.8 Modifying a QR Factorization 172
5.9 Summary and Table of Comparisons 174
Suggestions for Further Reading 185

CHAPTER SIX

Numerical Solutions of Linear Systems 187

6.1 Introduction 187
6.2 Basic Results on the Existence, Uniqueness, and Invariance of Solutions 189
6.3 Some Applications Giving Rise to Linear Systems Problems 190
6.3.1 An Electric Circuit Problem 190
6.3.2 Analysis of a Processing Plant Consisting of Interconnected Reactors 191
6.3.3 Linear Systems Arising from Ordinary Differential Equations (Finite Difference Scheme) 193
6.3.4 Linear Systems Arising from Partial Differential Equations: A Case Study on Temperature Distribution 195
6.3.5 Special Linear Systems Arising in Applications 199
6.3.6 Linear Systems Arising from Finite Element Methods 203
6.3.7 Approximation of a Function by a Polynomial: Hilbert System 206
6.4 Direct Methods 207
6.4.1 Solution of a Lower Triangular System 208
6.4.2 Solution of the System $Ax = b$ Using Gaussian Elimination without Pivoting 208
6.4.3 Solution of $Ax = b$ Using Pivoting Triangularization 209
6.4.4 Solution of $Ax = b$ without Explicit Factorization 213
6.4.5 Solution of $Ax = b$ Using QR Factorization 215
6.4.6 Solving a Linear System with Multiple Right-hand Sides 217

6.4.7 Special Systems 218
6.4.8 Scaling 235
6.4.9 LU Versus QR and a Table of Comparisons 236
6.5 Inverses and Determinants 238
6.5.1 Avoiding Explicit Computation of the Inverses 238
6.5.2 The Sherman–Morrison and Woodbury Formulas 239
6.5.3 Computing the Inverse of a Matrix 240
6.5.4 Computing the Determinant of a Matrix 241
6.6 Perturbation Analysis of the Linear System Problem 243
6.6.1 Effect of Perturbation in the Right-Hand-Side Vector b 244
6.6.2 Effect of Perturbation in Matrix A 247
6.6.3 Effect of Perturbations in Both Matrix A and Vector b 248
6.7 The Condition Number and Accuracy of Solution 250
6.7.1 Some Well-known Ill-conditioned Matrices 252
6.7.2 Effect of the Condition Number on Accuracy of the Computed Solution 252
6.7.3 Size of the Condition Number Required for Ill-Conditioning 253
6.7.4 The Condition Number and Nearness to Singularity 254
6.7.5 Conditioning and Pivoting 254
6.7.6 Conditioning and the Eigenvalue Problem 255
6.7.7 Conditioning and Scaling 256
6.7.8 Computing and Estimating the Condition Number 256
6.8 Componentwise Perturbations and the Errors 260
6.9 Iterative Refinement 262
6.10 Iterative Methods 264
6.10.1 The Jacobi Method 266
6.10.2 The Gauss–Seidel Method 269
6.10.3 Convergence of Iterative Methods 272
6.10.4 The Successive Overrelaxation (SOR) Method 277
6.10.5 The Conjugate Gradient Method 283
6.10.6 The Arnoldi Process and GMRES 288
6.11 Review and Summary 292
Suggestions for Further Reading 312

CHAPTER SEVEN

Least-Squares Solutions to Linear Systems 315

7.1 Introduction 315
7.2 A Simple Application Leading to an Overdetermined System 316
7.3 Existence and Uniqueness 317
7.4 Geometric Interpretation of the Least-Squares Problem 319
7.5 Normal Equations and Polynomial Fitting 319
7.6 Pseudoinverse and the Least-Squares Problem 323

Gram-Schmidt 338

7.7 Sensitivity of the Least-Squares Problem 324
7.8 Computational Methods for Overdetermined Least-Squares Problems 330
7.8.1 The Normal Equations Method 330
7.8.2 QR Factorization Methods for the Full-Rank Problem 334
7.8.3 The QR Factorization Method for the Rank-Deficient Case 343
7.8.4 Least-Squares Solution Using SVD 347
7.9 Underdetermined Systems 348
7.9.1 The Minimum-Norm Solution of the Full-Rank Underdetermined Problem Using Normal Equations 349
7.9.2 The QR Approach for the Full-Rank Underdetermined Problem 351
7.9.3 Minimum-Norm Solution by MGS 353
7.10 Iterative Refinement 353
7.11 Computing the Variance-Covariance Matrix $(A^TA)^{-1}$ 357
7.12 Review and Summary 360
Suggestions for Further Reading 370

CHAPTER EIGHT

Numerical Matrix Eigenvalue Problems 372

8.1 Introduction 372
8.2 Some Basic Results Involving Eigenvalues and Eigenvectors 373
8.2.1 Eigenvalues and Eigenvectors 374
8.2.2 The Schur Triangularization Theorem and Its Applications 377
8.2.3 The Jordan Canonical Form 380
8.3 The Eigenvalue Problems Arising in Practical Applications 381
8.3.1 Stability Problems for Differential and Difference Equations 381
8.3.2 Vibration Problem, Buckling Problem, and Simulating Transient Current of an Electrical Circuit 387
8.3.3 An Example of the Eigenvalue Problem Arising in Statistics: Principal Components Analysis 398
8.4 Localization of Eigenvalues 400
8.4.1 The Geršgorin Disk Theorems 400
8.4.2 Eigenvalue Bounds and Matrix Norms 403
8.4.3 Inertia of a Matrix 404
8.5 Computing Selected Eigenvalues and Eigenvectors 406
8.5.1 Discussions on the Importance of the Largest and Smallest Eigenvalues 406
8.5.2 The Role of Dominant Eigenvalues and Eigenvectors in Dynamic Systems 406
8.5.3 The Power Method, the Inverse Iteration, and the Rayleigh Quotient Iteration 407
8.5.4 Computing the Subdominant Eigenvalues and Eigenvectors: Deflation 418

8.6 Similarity Transformations and Eigenvalue Computations 422
8.6.1 Difficulties with Using the Characteristic Polynomial 422
8.6.2 Eigenvalue Computations via the Jordan Canonical Form 425
8.7 Eigenvalue Sensitivity 425
8.7.1 The Bauer–Fike Theorem 425
8.7.2 Sensitivity of the Individual Eigenvalues 427
8.8 Eigenvector Sensitivity 431
8.9 The Real Schur Form and QR Iterations 432
8.9.1 The Basic QR Iteration 435
8.9.2 The Hessenberg QR Iteration 437
8.9.3 Convergence of the QR Iterations and the Shift of Origin 438
8.9.4 The Single-Shift QR Iteration 439
8.9.5 The Double-Shift QR Iteration 441
8.9.6 Implicit QR Iteration 444
8.9.7 Obtaining the Real Schur Form 447
8.9.8 The Real Schur Form and Invariant Subspaces 450
8.10 Computing the Eigenvectors 451
8.10.1 The Hessenberg Inverse Iteration 451
8.10.2 Calculating the Eigenvectors from the Real Schur Form 452
8.11 The Symmetric Eigenvalue Problem 455
8.11.1 The Sturm Sequence and the Bisection Method 458
8.11.2 The Symmetric QR Iteration Method 462
8.11.3 The Divide-and-Conquer Method 464
8.12 The Lanczos Algorithm for Symmetric Matrices 468
8.13 Review and Summary 474
Suggestions for Further Reading 491

CHAPTER NINE

The Generalized Eigenvalue Problem 494

9.1 Introduction 494
9.2 Generalized Schur Decomposition 496
9.3 The QZ Algorithm 497
9.3.1 Reduction to Hessenberg-Triangular Form 498
9.3.2 Reduction to the Generalized Schur Form 500
9.4 The Inverse Iteration for Eigenvectors of the Pencil (A, B) 505
9.5 The Symmetric Definite Generalized Eigenvalue Problem 506
Computational Algorithms 507
9.5.1 The QZ Method for the Symmetric Definite Pencil 507
9.5.2 The Cholesky QR Algorithm 507
9.5.3 Diagonalization of the Symmetric Definite Pencil 509
9.6 Symmetric Definite Generalized Eigenvalue Problems Arising

9.6.1 A Case Study: The Vibration of a Spring-Mass System 512
9.6.2 A Case Study: The Vibration of a Building 514
9.6.3 Forced Harmonic Vibration 517
9.7 Applications to Decoupling and Model Reduction 520
9.7.1 Decoupling of a Second-Order System 520
9.7.2 The Reduction of a Large Model 527
9.7.3 A Case Study: The Potential Damage to a Building Due to an Earthquake 528
9.8 The Quadratic Eigenvalue Problem 531
9.9 Symmetric Generalized Eigenvalue Problems for Large and Structured Matrices 534
9.9.1 The Sturm Sequence Method for Tridiagonal A and B 534
9.9.2 The Lanczos Algorithm for the Pencil $A - \lambda B$ 535
9.9.3 Estimating the Eigenvalues 537
9.10 The Lanczos Method for Eigenvalues of the Pencil $A - \lambda B$ in an Interval 538
9.11 Review and Summary 539
Suggestions for Further Reading 549

CHAPTER TEN

The Singular Value Decomposition (SVD) 551

10.1 Introduction 551
10.2 The Singular Value Decomposition Theorem 552
10.3 A Relationship between the Singular Value Decomposition and Eigendecompositions 555
10.4 Applications of the SVD 558
10.5 Sensitivity of the Singular Values 560
10.6 The Singular Value Decomposition and the Structure of a Matrix 561
10.6.1 Norms and the Condition Number 562
10.6.2 Orthonormal Bases and the Projections 562
10.6.3 The Rank and the Rank-Deficiency of a Matrix 564
10.6.4 An Outer-Product Expansion of A and Its Consequences 568
10.6.5 Numerical Rank 569
10.7 Computing the Variance-Covariance Matrix 570
10.8 The Singular Value Decomposition, the Least-Squares Problem, and the Pseudoinverse 570
10.8.1 The SVD and Least-Squares Problem 571
10.8.2 Solving a Linear System Using Singular Value Decomposition 573
10.8.3 The SVD and the Pseudoinverse 574
10.9 Computing the Singular Value Decomposition 576

10.9.1 Computing the SVD from the Eigenvalue Decomposition of $A^T A$ 576
10.9.2 The Golub–Kahan–Reinsch Algorithm 577
10.9.3 The Chan SVD 583
10.9.4 The Demmel–Kahan Zero-Shift QR Algorithm 585
10.9.5 The Fernando-Parlett dqds Algorithm 588
10.9.6 The Bisection Method 588
10.10 Generalized SVD 589
10.11 Review and Summary 590
Suggestions for Further Reading 598

CHAPTER ELEVEN

A Taste of Roundoff Error Analysis 600

11.1 Basic Laws of Floating-Point Arithmetic 600
11.2 Backward Error Analysis for Forward Elimination and Back Substitutions 601
11.3 Backward Error Analysis for Triangularization by Gaussian Elimination 603
11.4 Backward Error Analysis for Solving $Ax = b$ 608
11.5 Review and Summary 611
Suggestions for Further Reading 612

APPENDIX A

A Brief Introduction to MATLAB 613

A.1 Some Basic Information on MATLAB 613
A.1.1 What is MATLAB? 613
A.1.2 Entering and Exiting MATLAB 613
A.1.3 Two Important Commands: help and demo 613
A.1.4 Most Frequently Used MATLAB Operations and Functions 614
A.1.5 Numerical Examples 615
A.2 Writing Your Own Programs Using MATLAB Commands 618
A.2.1 Some Relational Operators in MATLAB 618
A.2.2 Some Matrix-Building Functions 618
A.2.3 Colon Notation 619
A.2.4 for, while, if Commands 619
A.2.5 Computing Flop Count and Elapsed Time of an Algorithm 619
A.2.6 Saving a MATLAB Program 619
A.2.7 Getting a Hard Copy 619
A.2.8 Examples of Some Simple MATLAB Programs 620

A.2.9 Executing an M-File 621
A.2.10 Use of *diary* Command and Printing the Output 621

APPENDIX B

MATCOM and Selected MATLAB Programs from MATCOM 623

B.1 What is MATCOM? 623
B.2 Using MATCOM 623
B.3 Chapter-by-Chapter Listing of MATCOM Programs 624
B.4 Some Selected MATCOM Programs 628

APPENDIX C

A List of Commonly Used LAPACK Routines 633

Key Terms in Numerical Linear Algebra 635

Partial Solutions and Answers to Some Selected Problems 643

References 655

Table of Symbols 671

List of Algorithms 673

Index 675

CHAPTER ZERO

Numerical Linear Algebra Problems: Their Importance and Computational Difficulties

0.1 Introduction

The main objectives of this chapter are to state the fundamental numerical linear algebra problems at the outset, make a brief mention of their importance, and point out the difficulties that one faces in computational setting when trying to solve these problems using obvious approaches.

0.2 Numerical Linear Algebra Problems and Their Importance

The numerical linear algebra problems are shown in the following boxes.

The Linear System Problem
Given an $n \times n$ nonsingular matrix A and an n-vector b, the problem is to find an n-vector x such that $Ax = b$.

A practical variation of the problem requires solutions of several linear systems with the same matrix A on the left-hand side. That is, the problem is to find a matrix $X = (x_1, x_2, \ldots, x_m)$ such that

$$AX = B$$

where $B = (b_1, b_2, \ldots, b_m)$ is an $n \times m$ matrix.

Associated with linear system problems are the problems of finding the inverse of a matrix, the rank, the determinant, the leading principal minors, an orthonormal basis for the range and the null-space of A, and various projection matrices associated with A. Solutions of some of these later problems require matrix factorizations and the problem of matrix factorizations and linear system problems are intimately related.

It is perhaps not an exaggeration to say that the linear system problem arises in almost all branches of science and engineering: applied mathematics, biology, chemistry, physics, electrical, mechanical, civil, and vibration engineering, and so on.

The most common source of the above problems is the numerical solution of differential equations. Many mathematical models of physical and engineering systems are systems of differential equations, both ordinary and partial. A system of differential equations is normally solved numerically by discretizing the system by means of finite differences or finite element methods. The process of discretization, in general, leads to a linear system, the solution of which is an approximate solution to the differential equations. (See Chapter 6 for more details.)

The Least-Squares Problem
Given an $m \times n$ matrix A and an m-vector b, the least-squares problem is to find an n-vector x such that the norm of the residual vector, $\|Ax - b\|_2$, is as small as possible.

Least-squares problems arise in statistical and geometric applications that require fitting a polynomial or curve to experimental data and engineering applications such as signal and image processing. See Chapter 7 for some specific applications of least-squares problems. It is worth mentioning here that methods for numerically solving least-squares problems invariably lead to solutions of linear systems problems (again, see Chapter 7 for details).

The Eigenvalue Problem
Given an $n \times n$ matrix A, the problem is to find n scalars λ and nonzero vectors x such that

$$Ax = \lambda x$$

The eigenvalue problem typically arises in the explicit solution and stability analysis of a homogeneous system of first-order differential equations. The stability analysis requires only implicit knowledge of eigenvalues, whereas the explicit solution requires eigenvalues and eigenvectors explicitly.

Applications such as buckling problems, stock market analysis, and study of behavior of dynamic systems require computations of only a few eigenvalues and eigenvectors—usually the few largest or smallest ones.

In many practical instances, the matrix A is symmetric, and thus the eigenvalue problem becomes a symmetric eigenvalue problem. For details of some specific applications, see Chapter 8. A great number of eigenvalue problems arising in engineering applications are, however, generalized eigenvalue problems, as stated next.

The Generalized Eigenvalue Problem
Given square matrices A and B, the problem is to find scalars λ and nonzero vectors x such that

$$Ax = \lambda Bx$$

The generalized eigenvalue problem typically arises in vibration analysis of undamped structures modeled by a second-order system of differential equations

$$B\frac{d^2z}{dt^2} + Az = 0$$

Similarly the damped system

$$A\frac{d^2z}{dt^2} + B\frac{dz}{dt} + C = 0$$

leads to the *Quadratic Eigenvalue Problem.*

The Quadratic Eigenvalue Problem
Given square matrices A, B, and C, find scalars λ and nonzero vectors x such that

$$(\lambda^2 A + \lambda B + C)x = 0$$

For details of how the generalized eigenvalue problem and the quadratic eigenvalue problem arise in specific applications, see Chapters 8 and 9.

Singular Value Decomposition Problem
Given an $m \times n$ matrix A, the problem is to find orthogonal matrices U and V and a "diagonal" matrix Σ such that

$$A = U(\Sigma)V^T$$

This decomposition is known as the *singular value decomposition of A*. The entries of Σ are singular values. The column vectors of U and V are called the singular vectors.

Many areas of engineering, such as control and systems theory, biomedical engineering, signal and image processing, and statistical applications, give rise to the singular value decomposition problem. These applications typically require the rank of A, an orthonormal basis, projections, the distance of a matrix from another matrix of lower rank, and so on, in the presence of certain impurities (known as noises) in the data. Singular values and singular vectors are the most numerically reliable tools for finding these entities. The singular value decomposition is also the most numerically effective approach to solve the least-squares problem, especially in the rank-deficient case.

0.3 Solving Numerical Linear Algebra Problems Using Obvious Approaches

In this section we would like to point out how some of the "obvious" approaches to solving some of the previously mentioned problems can be impractical in a computational setting and can produce inaccurate results.

• *Solving a linear system by Cramer's rule:* Cramer's rule, as taught at an undergraduate linear algebra course, is of significant theoretical and historical importance (for a statement of this rule, see Chapter 6). Unfortunately, it cannot be recommended as a practical computational procedure.

Solving a 20×20 linear system, even on a fast, modern-day computer, might take more than a million years with this rule, using the usual definition of the determinant of a matrix.

• *Computing the unique solution of a linear system by matrix inversion:* The unique solution of a nonsingular linear system can be written explicitly as $x = A^{-1}b$.

Unfortunately, computing a solution to a linear system by first explicitly computing the matrix inverse is not practical.

The computation of the matrix inverse is about three times as expensive as solving the linear system problem itself using a standard elimination procedure (see Chapter 6) and often leads to more inaccuracies. Consider a trivial example: Solve $3x = 27$. An elimination procedure will give $x = 9$ and require only one division. On the other hand, solving the equation using matrix inversion will be cast as $x = \frac{1}{3} \cdot 27$, giving $x = 0.3333 \cdot 27 = 8.9991$ (in four-digit arithmetic), and will require one division and one multiplication.

Note that computer time consumed by an algorithm is theoretically measured by the number of arithmetic operations needed to execute the algorithm.

• *Solving a least-squares problem by normal equations:* If the $m \times n$ matrix A has full rank and m is greater than or equal to n, then the least-squares problem has a unique solution, and this solution is theoretically given by the solution x to the linear system

$$A^T Ax = A^T b$$

These equations are known as the *normal equations*. Unfortunately, this procedure has some severe numerical limitations. First, in finite-precision arithmetic, during an explicit formation of $A^T A$, some vital information might be lost. Second, the normal equations are more sensitive to perturbations than the ordinary linear system $Ax = b$, and this sensitivity, in certain instances, corrupts the accuracy of the computed least squares solution to an extent not warranted by the data. (See Chapter 7 for more details.)

• *Computing the eigenvalues of a matrix by finding the roots of its characteristic polynomial:* The eigenvalues of a matrix A are the zeros of its characteristic polynomial. Thus an obvious procedure for finding the eigenvalues would be to compute the characteristic polynomial of A and then find its zeros by a standard, well-established root-finding procedure. Unfortunately, this is not a numerically viable approach. The rounding errors produced during a process for computing the characteristic polynomial will very likely produce some small perturbations in the computed coefficients. These small errors in the coefficients can produce large errors in the eigenvalues in certain cases. The zeros of certain polynomials are known to be extremely sensitive to small perturbations in the coefficients. A classic example of this is the Wilkinson polynomial (see Chapter 3). Wilkinson took a polynomial of degree 20 with the distinct roots 1 through 20 and perturbed the coefficient of x^{19} by

a significantly small amount. The zeros of this slightly perturbed polynomial were then computed by a well-established root-finding procedure, with the result that some zeros became totally different. Some even became complex.

• *Solving the generalized eigenvalue problem and the quadratic eigenvalue problems by matrix inversion:* The generalized eigenvalue problem in the case where B is nonsingular,

$$Ax = \lambda Bx$$

is theoretically equivalent to the ordinary eigenvalue problem:

$$B^{-1}Ax = \lambda x$$

However, if the matrix B is nearly singular, then forming the matrix on the left-hand side by explicitly computing the inverse of B will lead to inaccuracies that, in turn, will lead to computations of inaccurate eigenvalues.

Similar results hold for the quadratic eigenvalue problem. In major engineering applications, such as in vibration engineering, the matrix A is symmetric positive definite and is thus nonsingular. In that case the quadratic eigenvalue problem is equivalent to the eigenvalue problem $Eu = \lambda u$, where

$$E = \begin{pmatrix} 0 & I \\ -A^{-1}C & -A^{-1}B \end{pmatrix}$$

But numerically it is not advisable to solve the quadratic eigenvalue problem by actually computing the matrix E explicitly. If A is nearly singular, the matrix E cannot be formed accurately, and the computed eigenvalues will then be inaccurate.

• *Finding singular values by computing the eigenvalues of $A^T A$:* When $m \geq n$, the singular values of A are the nonnegative square roots of the eigenvalues of $A^T A$. However, finding the singular values this way is not advisable. Again, explicit formation of the matrix product might lead to the loss of significant relevant information. Consider a rather trivial example:

$$A = \begin{pmatrix} 1 & 1 \\ \epsilon & 0 \\ 0 & 0 \end{pmatrix}$$

where ϵ is such that in finite precision computation $1 + \epsilon^2 = 1$. Then, computationally, we have

$$A^T A = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

The eigenvalues are 2 and 0. Thus, the computed singular values are given by $\sqrt{2}$ and 0. The exact singular values, however are $\sqrt{2}$ and $\epsilon/\sqrt{2}$. (See Chapter 10 for details.)

Conclusion

We have pointed out how certain obvious theoretical approaches to linear algebra problems might lead to computational difficulties and inaccuracies in computed results. Numerical linear algebra deals with in-depth analysis of such difficulties, with investigations of how these difficulties can be overcome in certain instances, and with formulation and implementations of viable numerical algorithms for scientific and engineering use.

CHAPTER ONE

A Review of Required Linear Algebra Concepts

1.1 Introduction

Although a first course in linear algebra is a prerequisite for this book, for the sake of completeness we establish some notation and quickly review the basic definitions and concepts of matrices and vectors in this chapter. We then discuss in somewhat greater detail the concepts and fundamental results of vector and matrix norms and their applications to the study of convergent matrices. These results are used frequently in the later chapters of the book.

1.2 Vectors

An ordered set of real numbers is called a **vector**; the numbers themselves are called the **components** of the vector. A lowercase italic letter is usually used to denote a vector. A vector v having n components has the form

$$v = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix}$$

A vector in this form is referred to as a **column vector**, and its transpose is a **row vector**. The set of all n-vectors (that is, each vector having n components) denoted by $\mathbb{R}^n$ is called the **vector space** $\mathbb{R}^n$. The transpose of a vector v is denoted by v^T. Unless otherwise stated, a column vector will simply be called a vector.

If u and v are two vectors in $\mathbb{R}^n$, then their **sum** $u + v$ is defined by

$$u + v = (u_1 + v_1, u_2 + v_2, \ldots, u_n + v_n)^T$$

If c is a scalar, then $cu = (cu_1, cu_2, \ldots, cu_n)^T$. The **inner product** of two vectors u and v is the scalar given by

The **length** of a vector v, denoted by $\|v\|$, is $\sqrt{v^T v}$; that is, the length of v (or Euclidean length of v) is $\sqrt{v_1^2 + v_2^2 + \cdots + v_n^2}$.

A set of vectors $\{m_1, \ldots, m_k\}$ in $\mathbb{R}^n$ is said to be **linearly dependent** if there exist scalars $c_1, \ldots, c_k$, not all zero, such that $c_1 m_1 + \cdots + c_k m_k = 0$ (zero vector, that is, a vector having all its components zero). Otherwise, the set is called **linearly independent**. The set of all linear combinations of $m_1, \ldots, m_k$ will be denoted by $\text{span}\{m_1, \ldots, m_k\}$.

Example 1.2.1 The set of vectors

$$e_i = (0, 0, \ldots, 0, \underset{\substack{\uparrow \\ i\text{th component}}}{1}, 0 \ldots, 0)^T, \qquad i = 1, \ldots, n$$

is linearly independent, because

$$c_1 e_1 + c_2 e_2 + \cdots + c_n e_n = \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{pmatrix} = 0$$

is true if and only if

$$c_1 = c_2 = \cdots = c_n = 0$$

Example 1.2.2 The vectors $\begin{pmatrix} 1 \\ -2 \end{pmatrix}$ and $\begin{pmatrix} -3 \\ 6 \end{pmatrix}$ are linearly dependent, because

$$3\begin{pmatrix} 1 \\ -2 \end{pmatrix} + \begin{pmatrix} -3 \\ 6 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Here $c_1 = 3$ and $c_2 = 1$.

1.2.1 Orthogonality of Vectors and Subspaces

The angle θ between two vectors u and v is given by

$$\cos(\theta) = \frac{u^T v}{\|u\|\|v\|}$$

Two vectors u and v are **orthogonal** if $\theta = 90°$—that is, $u^T v = 0$. The symbol $\perp$ is used to denote orthogonality.

Let S be a set of vectors in $\mathbb{R}^n$. Then S is called a **subspace** of $\mathbb{R}^n$ if $s_1, s_2 \in S$ implies $c_1 s_1 + c_2 s_2 \in S$, where c_1 and c_2 are any scalars. That is, S is a subspace if any linear combination of two vectors in S is also in S. Note that the space $\mathbb{R}^n$ is itself a subspace of $\mathbb{R}^n$. For every subspace there is a unique smallest positive integer r such that every vector in the subspace can be expressed as a linear combination of at most r vectors in the subspace; r is called the **dimension** of the subspace and is denoted by $\dim[S]$. Any set of r linearly independent vectors from S of $\dim[S] = r$ is said to form a **basis** of the subspace.

Two subspaces S_1 and S_2 of $\mathbb{R}^n$ are said to be orthogonal if $s_1^T s_2 = 0$ for every $s_1 \in S_1$ and every $s_2 \in S_2$. Two orthogonal subspaces S_1 and S_2 are denoted by $S_1 \perp S_2$.

1.3 Matrices

1.3.1 Basic Concepts

A collection of n vectors in $\mathbb{R}^n$ arranged in a rectangular array of m rows and n columns is called a **matrix**. A matrix A, therefore, has the form

$$A = \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & & & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{pmatrix}$$

It is denoted by $A = (a_{ij})_{m \times n}$ or, simply, by $A = (a_{ij})$, where it is understood that $i = 1, \ldots, m$ and $j = 1, \ldots, n$. A is said to be of order $m \times n$. The set of all $m \times n$ matrices is denoted by $\mathbb{R}^{m \times n}$.

A matrix A having the same number of rows as columns is called a **square matrix**. The square matrix with 1s along the main diagonal and zeros everywhere else is called the **identity matrix** and is denoted by I. Sometimes a $p \times p$ identity matrix is denoted by $Ip \times p$ in this text.

The sum of two matrices $A = (a_{ij})$ and $B = (b_{ij})$ in $\mathbb{R}^{m \times n}$ is a matrix of the same order as A and B and is given by

$$A + B = (a_{ij} + b_{ij})$$

If c is a scalar, then cA is a matrix given by

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

Let A be $m \times n$ and B be $n \times p$. Then their product AB is an $m \times p$ matrix given by

$$AB = \left(\sum_{k=1}^{n} a_{ik} b_{kj}\right); \qquad i = 1, \ldots, m, \quad j = 1, \ldots, p$$

Note that if b is a column vector, then Ab is a column vector. On the other hand, if a is a column vector and b^T is a row vector, then ab^T is a matrix, known as the outer product of the two vectors a and b. Thus,

$$ab^T = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} (b_1 \quad b_2 \quad \ldots \quad b_m) = \begin{pmatrix} a_1 b_1 & \ldots & a_1 b_m \\ a_2 b_1 & \ldots & a_2 b_m \\ \vdots & & \vdots \\ a_n b_1 & \ldots & a_n b_m \end{pmatrix}$$

Example 1.3.1 Let

$$a = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}, \qquad b^T = (2 \quad 3 \quad 4)$$

The **outer product** is $ab^T = \begin{pmatrix} 2 & 3 & 4 \\ 4 & 6 & 8 \\ 6 & 9 & 12 \end{pmatrix}$ (a matrix).

The **inner product** is $a^T b = (1 \quad 2 \quad 3)\begin{pmatrix} 2 \\ 3 \\ 4 \end{pmatrix} = 20$ (a scalar).

The **transpose** of a matrix A of order $m \times n$, denoted by A^T, is a matrix of order $n \times m$ with rows and columns interchanged:

$$A^T = (a_{ji}), \qquad i = 1, \ldots n, \qquad j = 1, \ldots m$$

Note that *the matrix product is not commutative*; that is, in general,

$$AB \neq BA$$

Also,

$$\boxed{(AB)^T = B^T A^T}$$

DEFINITION 1.3.1 A matrix A is called **symmetric** if $A^T = A$.

$b_i \in \mathbb{R}^m$

An alternative way of writing the matrix product is to write $B = (b_1, \ldots, b_p)$, where b_i is the ith column of B. Then the matrix product AB can be written as

$$AB = (Ab_1, \ldots, Ab_p)_{n \times p}$$

$a_i^T \in \mathbb{R}^m$

Similarly, if a_i^T is the ith row of A,

$$A = \begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix}_{n \times m} \qquad AB = \begin{pmatrix} a_1^T B \\ a_2^T B \\ \vdots \\ a_m^T B \end{pmatrix}_{n \times p} ? = \begin{pmatrix} a_1 B \\ \vdots \\ a_n B \end{pmatrix}_{n \times p}$$

$A_{n \times m} \cdot B_{m \times p} = AB_{n \times p}$

$a_i^T{}_{1 \times m} B_{m \times p} = a_i^T B_{1 \times p}$

$A_{n \times m} b_i{}_{m \times 1} = Ab_i{}_{n \times 1}$

Block Matrices

If two matrices A and B can be partitioned as

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \qquad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

then considering each block as an element of the matrix, we can perform addition, scalar multiplication, and matrix multiplication in the usual way. Thus,

$$A + B = \begin{pmatrix} A_{11} + B_{11} & A_{12} + B_{12} \\ A_{21} + B_{21} & A_{22} + B_{22} \end{pmatrix}$$

and

$$AB = \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix}$$

assuming that the partitioning has been done **conformably** so that the corresponding matrix multiplications are possible. The concept of two block partitioning can be easily generalized.

Thus, if $A = (A_{ij})$ and $B = (B_{ij})$ are two block matrices, $C = AB$ is given by

$$C = (C_{ij}) = \left(\sum_{k=1}^{n} A_{ik} B_{kj} \right)$$

where each A_{ik}, B_{kj}, and C_{ij} is a block matrix.

A **block diagonal matrix** is an $n \times n$ matrix where each diagonal element is a square matrix and other elements are all zero. It is conveniently written as

$$A = \operatorname{diag}(A_{11}, \ldots, A_{kk})$$

where A_{ii} are square matrices. The sum of the orders of these matrices is equal to n.

The Determinant of a Matrix

For every square matrix A, there is a unique scalar, called the **determinant** of A, which is denoted by $\det(A)$, associated with the matrix. For a 2×2 matrix A, $\det(A) = a_{11}a_{22} - a_{12}a_{21}$; for a 3×3 matrix $A = (a_{ij})$, $\det(A) = a_{11} \cdot \det(A_{11}) - a_{12} \cdot \det(A_{12}) + a_{13} \cdot \det(A_{13})$, where A_{1i} is a 2×2 submatrix obtained by eliminating the first row and the ith column. This can be easily generalized. For an $n \times n$ matrix $A = (a_{ij})$ we have

$$\det(A) = (-1)^{i+1} a_{i1} \det(A_{i1}) + (-1)^{i+2} a_{i2} \det(A_{i2}) + \cdots + (-1)^{i+n} a_{in} \det(A_{in})$$

where A_{ij} is the submatrix of A of order $(n-1)$ obtained by eliminating the ith row and jth column.

Example 1.3.2

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

Set $i = 1$. Then

$$\det(A) = 1 \cdot \det\begin{pmatrix} 5 & 6 \\ 8 & 9 \end{pmatrix} - 2 \cdot \det\begin{pmatrix} 4 & 6 \\ 7 & 9 \end{pmatrix} + 3 \cdot \det\begin{pmatrix} 4 & 5 \\ 7 & 8 \end{pmatrix}$$
$$= 1(-3) - 2(-6) + 3(-3) = 0$$

THEOREM 1.3.1 The following simple properties of $\det(A)$ hold:

1. $\det(A) = \det(A^T)$.
2. $\det(\alpha A) = \alpha^n \det(A)$, where α is a scalar.
3. $\det(AB) = \det(A) \cdot \det(B)$.
4. If two rows or two columns of A are identical, then $\det(A) = 0$.
5. If B is a matrix obtained from A by interchanging two rows or two columns, then $\det(B) = -\det(A)$.
6. The determinant of a triangular matrix is the product of its diagonal entries.

(A square matrix A is **triangular** if all elements below or above the diagonal are all zero.)

The Characteristic Polynomial, Eigenvalues, and Eigenvectors of a Matrix

Let A be an $n \times n$ matrix. Then the polynomial $p_n(\lambda) = \det(A - \lambda I)$ is called the **characteristic polynomial**. The zeros of the characteristic polynomial are called the **eigenvalues** of A. Note that this is equivalent to the following: λ is an eigenvalue of A if and only if (iff) there exists a nonzero vector x such that $Ax = \lambda x$. The vector x is called a **right eigenvector** (or just an eigenvector).

DEFINITION 1.3.2 An $n \times n$ matrix A having fewer than n linearly independent eigenvectors is called a **defective matrix**.

Example 1.3.3 The matrix $A = \begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}$ is defective. It has only one eigenvector $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$.

The Determinant of a Block Triangular Matrix

Let

$$A = \begin{pmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{pmatrix}$$

where A_{11} and A_{22} are square matrices. Then $\det(A) = \det(A_{11}) \cdot \det(A_{22})$.

1.3.2 Range and Null Space

For every $m \times n$ matrix A, there are two important associated subspaces: the **range** of A, denoted by $R(A)$, and the **null space** of A, denoted by $N(A)$:

$$R(A) = \{b \in \mathbb{R}^m \mid b = Ax \text{ for some } x \in \mathbb{R}^n\}$$

$$N(A) = \{x \in \mathbb{R}^n \mid Ax = 0\}$$

Let S be a subspace of $\mathbb{R}^m$. Then the subspace $S^\perp$ defined by

$$S^\perp = \{y \in \mathbb{R}^m \mid y^T x = 0 \text{ for all } x \in S\}$$

is called the **orthogonal complement** of S. It can be shown (see the exercises) that

1. $N(A) = R(A^T)^\perp$
2. $R(A)^\perp = N(A^T)$

The dimension of $N(A)$ is called the **nullity** of A and is denoted by $\text{null}(A)$.

1.3.3 Rank of a Matrix

Let A be an $m \times n$ matrix. Then the subspace spanned by the row vectors of A is called the **row space** of A. The subspace spanned by the columns of A is called the **column space** of A. The range of A is the same as the column space of A.

The **rank** of a matrix A is the dimension of the column space of A. It is denoted by $\text{rank}(A)$. A square matrix $A \in \mathbb{R}^{n \times n}$ is called **nonsingular** if $\text{rank}(A) = n$. Otherwise it is **singular**.

An $m \times n$ matrix $A \in \mathbb{R}^{m \times n}$ is said to have **full column rank** if its columns are linearly independent. **Full row rank** is similarly defined. A matrix A is said to have **full rank** if it has either full row rank or full column rank. If A does not have full rank, it is **rank deficient**.

Example 1.3.4

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$$

has full rank; $\text{rank}(A) = 2$ (it has full column rank); $\text{null}(A) = 0$.

Example 1.3.5

$$A = \begin{pmatrix} 1 & 2 \\ 2 & 4 \\ 0 & 0 \end{pmatrix}$$

is rank deficient; $\text{rank}(A) = 1$; $\text{null}(A) = 1$.

Some Rank Properties

Let A be an $m \times n$ matrix. Then

1. $\text{rank}(A) = \text{rank}(A^T)$;
2. $\text{rank}(A) + \text{null}(A) = n$;
3. $\text{rank}(AB) \geq \text{rank}(A) + \text{rank}(B) - n$, where B is $n \times p$;
4. $\text{rank}(BA) = \text{rank}(A) = \text{rank}(AC)$, where B and C are nonsingular matrices of appropriate order;
5. $\text{rank}(AB) \leq \min\{\text{rank}(A), \text{rank}(B)\}$.
6. $\text{rank}(A + B) \leq \text{rank}(A) + \text{rank}(B)$.

1.3.4 The Inverse of a Matrix

An $n \times n$ matrix A is said to be invertible if there exists an $n \times n$ matrix B such that

$$AB = BA = I$$

The inverse of A is denoted by A^{-1}. *The inverse is unique.* An invertible matrix is often called a nonsingular matrix.

An interesting property of the inverse of the product of two invertible matrices is

$$\boxed{(AB)^{-1} = B^{-1}A^{-1}}$$

THEOREM 1.3.2 For an $n \times n$ matrix A, the following are equivalent:

1. A is nonsingular.
2. $\det(A)$ is nonzero.
3. $\text{rank}(A) = \text{rank}(A^T) = n$.
4. $N(A) = \{0\}$.
5. A^{-1} exists.
6. A has linearly independent rows and columns.
7. The eigenvalues of A are nonzero.

1.3.5 Similar Matrices

Two matrices A and B are called **similar** if there exists a nonsingular matrix T such that

$$T^{-1}AT = B$$

An important property of similar matrices is that two similar matrices have the same eigenvalues. (For a proof, see Section 8.2.)

1.3.6 Orthogonality and Projections

A set of vectors $\{v_1, \ldots, v_m\}$ in $\mathbb{R}^n$ is **orthogonal** if

$$v_i^T v_j = 0, \qquad i \neq j$$

If, in addition, $v_i^T v_i = 1$, for each i, then they are called **orthonormal**.

A basis for a subspace that is also orthonormal is called an **orthonormal basis** for the subspace.

Example 1.3.6

$$A = \begin{pmatrix} 0 & 0 \\ \frac{1}{2} & 1 \\ \frac{1}{2} & 1 \end{pmatrix}$$

The vector

$$\begin{pmatrix} 0 \\ -\frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{pmatrix}$$

forms an orthonormal basis for $R(A)$. (See Section 5.6.)

Example 1.3.7

$$A = \begin{pmatrix} 1 & 2 \\ 0 & 1 \\ 1 & 0 \end{pmatrix}$$

The vectors

$$\begin{pmatrix} -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{3}} \\ 0 & \frac{-1}{\sqrt{3}} \\ \frac{-1}{\sqrt{2}} & \frac{1}{\sqrt{3}} \end{pmatrix}$$

form an orthonormal basis for $R(A)$. (See Section 5.6.1.)

Orthogonal Projection

Let S be a subspace of $\mathbb{R}^n$. Then an $n \times n$ matrix P having the properties

1. $R(P) = S$,
2. $P^T = P$ (P is **symmetric**),
3. $P^2 = P$ (P is **idempotent**)

is called the **orthogonal projection** onto S or, simply, the **projection matrix**. We denote the orthogonal projection P onto S by P_S. *The orthogonal projection onto a subspace is unique.* Note that an orthogonal projection is not necessarily orthogonal.

> Let $V = (v_1, \ldots, v_k)$ where $\{v_1, \ldots, v_k\}$ is an orthonormal basis for a subspace S. Then
>
> $$P_S = VV^T$$
>
> is the unique orthogonal projection onto S. *Note that V is not unique, but P_S is.*

$P_{S^\perp} = I - P_S$

A Relationship between P_S and $P_{S^\perp}$

If P_S is the orthogonal projection onto S, then $I - P_S$, where I is the identity matrix of the same order as P_S, is the orthogonal projection onto $S^\perp$. [See exercise 14(a).]

The Orthogonal Projections onto $R(A)$ and $N(A^T)$

When the subspace S is $R(A)$ or $N(A^T)$, associated with the matrix A, we denote the unique orthogonal projections onto $R(A)$ and $N(A^T)$ by P_A and P_N, respectively.

> It can be shown [Exercise 14(b)] that if A is $m \times n$ ($m \geq n$) and has **full rank**, then
>
> $$P_A = A(A^TA)^{-1}A^T$$
> $$P_N = I - A(A^TA)^{-1}A^T$$

Example 1.3.8

$$A = \begin{pmatrix} 1 & 2 \\ 0 & 1 \\ 1 & 0 \end{pmatrix}$$

$$A^T A = \begin{pmatrix} 2 & 2 \\ 2 & 5 \end{pmatrix}, \qquad (A^T A)^{-1} = \begin{pmatrix} \frac{5}{6} & -\frac{1}{3} \\ -\frac{1}{3} & \frac{1}{3} \end{pmatrix}$$

$$P_A = \begin{pmatrix} \frac{5}{6} & \frac{1}{3} & \frac{1}{6} \\ \frac{1}{3} & \frac{1}{3} & -\frac{1}{3} \\ \frac{1}{6} & -\frac{1}{3} & \frac{5}{6} \end{pmatrix}$$

1.3.7 Projection of a Vector onto the Range and the Null Space of a Matrix

Let S be a subspace of $\mathbb{R}^n$. Then any vector b can be written as

$$b = b_S + b_{S^\perp}$$

where $b_S \in S$ and $b_{S^\perp} \in S^\perp$. Let S be the $R(A)$ of a matrix A. Then $b_S \in R(A)$ and $b_{S^\perp} \in N(A^T)$. We will, therefore, denote b_S by b_R and $b_{S^\perp}$ by b_N, meaning that b_R is in the range of A and b_N is in the null space of A^T.

> It can be shown [Exercise 14(c)] that
>
> $$b_R = P_A b \quad \text{and} \quad b_N = P_N b$$

Here, b_R and b_N are called the **orthogonal projection of b onto $R(A)$** and the **orthogonal projection of b onto $N(A^T)$**, respectively.

From the preceding, we easily see that

$$b_R^T b_N = 0$$

Example 1.3.9

$$A = \begin{pmatrix} 0 & 0 \\ \frac{1}{2} & 1 \\ \frac{1}{2} & 1 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

$$V = \text{an orthonormal basis} = \begin{pmatrix} 0 \\ -\frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{pmatrix}$$

$$P_A = VV^T = \begin{pmatrix} 0 & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & \frac{1}{2} & \frac{1}{2} \end{pmatrix}$$

$$b_R = P_A b = \begin{pmatrix} 0 & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & \frac{1}{2} & \frac{1}{2} \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$$

$$P_N = (I - P_A) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{2} & -\frac{1}{2} \\ 0 & -\frac{1}{2} & \frac{1}{2} \end{pmatrix}$$

$$b_N = P_N b = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

Note that $b = b_R + b_N$.

1.4 Some Special Matrices

1.4.1 Diagonal and Triangular Matrices

An $m \times n$ matrix $A = (a_{ij})$ is a **diagonal matrix** if $a_{ij} = 0$ for $i \neq j$. We write $A = \text{diag}(a_{11}, \ldots, a_{ss})$, where $s = \min(m, n)$.

A square matrix $A = (a_{ij})$ is an **upper triangular matrix** if $a_{ij} = 0$ for $i > j$.

The transpose of an upper triangular matrix is **lower triangular**; that is, $A = (a_{ij})$ is lower triangular if $a_{ij} = 0$ for $i < j$.

$$\begin{pmatrix} \backslash & 0 \\ * & \backslash \end{pmatrix} \qquad \begin{pmatrix} \backslash & * \\ 0 & \backslash \end{pmatrix}$$

Lower triangular Upper triangular

Some Useful Properties of Triangular Matrices

The following properties of triangular matrices are useful.

1. The product of two upper (lower) triangular matrices is an upper (lower) triangular matrix. The diagonal entries of the product matrix are just the products of the diagonal entries of the individual matrices [Exercise 19(a)].
2. The inverse of a nonsingular upper (lower) triangular matrix is an upper (lower) triangular matrix. The diagonal entries of the inverse are the reciprocals of the diagonal entries of the original matrix [Exercise 19(b)].
3. The eigenvalues of a triangular matrix are its diagonal entries [Exercise 19(d)].
4. The determinant of a triangular matrix is the product of its diagonal entries [Exercise 19(c)].

Thus, a triangular matrix is nonsingular iff all its diagonal entries are nonzero.

1.4.2 Orthogonal Matrix

A square matrix U is **orthogonal** if

$$U^T U = UU^T = I$$

Orthogonal matrices play an important role in numerical matrix computations.

The following two properties of orthogonal matrices are attractive for numerical computations:

1. The inverse of an orthogonal matrix O is just its transpose: $O^{-1} = O^T$.
2. The product of two orthogonal matrices is an orthogonal matrix.

1.4.3 Permutation Matrix

A nonzero square matrix P is called a **permutation matrix** if there is exactly one nonzero entry in each row and column that is 1 and if the rest are all zero. Thus, if $(\alpha_1, \ldots, \alpha_n)$ is a permutation of $(1, 2, \ldots, n)$, then

$$P = \begin{pmatrix} e_{\alpha_1}^T \\ e_{\alpha_2}^T \\ \vdots \\ e_{\alpha_n}^T \end{pmatrix}$$

where e_i is the ith column of the $n \times n$ identity matrix I, is a permutation matrix. Similarly,

$$P = (e_{\alpha_1}, e_{\alpha_2}, \ldots, e_{\alpha_n})$$

where e_i is the ith column of I, is a permutation matrix.

Example 1.4.1

$$P_1 = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}, \quad P_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad P_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

are all permutation matrices.

Effects of Premultiplication and Postmultiplication by a Permutation Matrix

If

$$P_1 = \begin{pmatrix} e_{\alpha_1}^T \\ \vdots \\ e_{\alpha_n}^T \end{pmatrix}$$

then

$$P_1 A = \begin{pmatrix} \alpha_1 \text{ row of } A \\ \alpha_2 \text{ row of } A \\ \vdots \\ \alpha_n \text{ row of } A \end{pmatrix}$$

Similarly, if $P_2 = (e_{\alpha_1}, e_{\alpha_2}, \ldots, e_{\alpha_n})$, then

$$AP_2 = (\alpha_1 \text{ column of } A, \alpha_2 \text{ column of } A, \ldots, \alpha_n \text{ column of } A)$$

Example 1.4.2

1. $$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}, \quad P_1 = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} = \begin{pmatrix} e_2^T \\ e_3^T \\ e_1^T \end{pmatrix}$$

$$P_1 A = \begin{pmatrix} a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \\ a_{11} & a_{12} & a_{13} \end{pmatrix} = \begin{pmatrix} \text{2nd row of } A \\ \text{3rd row of } A \\ \text{1st row of } A \end{pmatrix}$$

2. $$P_1 = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} = (e_3, e_1, e_2)$$

$$AP_1 = \begin{pmatrix} a_{13} & a_{11} & a_{12} \\ a_{23} & a_{21} & a_{22} \\ a_{33} & a_{31} & a_{32} \end{pmatrix} = \begin{matrix} (\text{3rd column of } A, \text{ 1st column of } A, \\ \text{2nd column of } A) \end{matrix}$$

An important property of a permutation matrix is that a *permutation matrix is orthogonal.* Thus, the following are true.

1. The inverse of a permutation matrix P is its transpose, and the transpose is also a permutation matrix.
2. The product of two permutation matrices is a permutation matrix and is, therefore, orthogonal.

1.4.4 Hessenberg (Almost Triangular) Matrix

A square matrix A is **upper Hessenberg** if $a_{ij} = 0$ for $i > j + 1$. The transpose of an upper Hessenberg matrix is a lower Hessenberg matrix, that is, a square matrix $A = (a_{ij})$ is a **lower Hessenberg** matrix if $a_{ij} = 0$ for $j > i + 1$. A square matrix A that is both upper and lower Hessenberg is **tridiagonal**.

$$\begin{pmatrix} * & * & & 0 \\ \vdots & & \ddots & \\ * & & & * \\ * & * & \cdots & * \end{pmatrix} \qquad \begin{pmatrix} * & \cdots & * & * \\ * & \cdots & * & * \\ & \ddots & \vdots & \vdots \\ 0 & & * & * \end{pmatrix}$$

Lower Hessenberg Upper Hessenberg

An upper Hessenberg matrix $A = (a_{ij})$ is **unreduced** if

$$a_{i,i-1} \neq 0 \qquad \text{for } i = 2, 3, \ldots, n$$

Similarly, a lower Hessenberg matrix $A = (a_{ij})$ is unreduced if

$$a_{i,i+1} \neq 0 \qquad \text{for } i = 1, 2, \ldots, n-1$$

Example 1.4.3

$$A = \begin{pmatrix} 1 & 2 & 0 \\ 2 & 3 & 4 \\ 1 & 1 & 1 \end{pmatrix} \text{ is an unreduced lower Hessenberg matrix.}$$

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 2 & 3 \end{pmatrix} \text{ is an unreduced upper Hessenberg matrix.}$$

Some Useful Properties Hessenberg matrices

1. Every square matrix A can be transformed to an upper (or lower) Hessenberg matrix by means of an orthogonal similarity; that is, given a real matrix A, there exists an orthogonal matrix U such that

$$UAU^T = H$$

where H is a Hessenberg matrix. (A constructive proof is given in Chapter 5.)
2. If A is symmetric, then the transformed Hessenberg matrix as obtained in 1 is symmetric tridiagonal.
3. An arbitrary Hessenberg matrix can always be partitioned into blocks such that each diagonal block is an unreduced Hessenberg matrix.

Example 1.4.4

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

$$A = \begin{pmatrix} A_1 & A_2 \\ 0 & A_3 \end{pmatrix}$$

Note that

$$A_1 = \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}$$

and

$$A_3 = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

are unreduced Hessenberg matrices.

1.4.5 Companion Matrix

An unreduced upper Hessenberg matrix of the form

$$C = \begin{pmatrix} 0 & 0 & \cdots & \cdots & a_1 \\ 1 & 0 & \cdots & \cdots & a_2 \\ 0 & 1 & \cdots & \cdots & a_3 \\ \vdots & \vdots & & & \vdots \\ 0 & 0 & 0 & 1 & a_n \end{pmatrix}$$

is called an **upper companion matrix**. The transpose of an upper companion matrix is a **lower companion matrix**.

The characteristic polynomial of a companion matrix is easy to determine:

$$\begin{aligned} \det(C - \lambda I) &= \det(C^T - \lambda I) \\ &= (-1)^n(\lambda^n - a_n\lambda^{n-1} - a_{n-1}\lambda^{n-2} - \cdots - a_2\lambda - a_1) \end{aligned}$$

1.4.6 Nonderogatory Matrix

A matrix A is **nonderogatory** if A is similar to a companion matrix; that is, A is nonderogatory if there exists a nonsingular T such that TAT^{-1} is a companion matrix. A is nonderog. $\Leftrightarrow$ TAT^{-1}

Remark: An unreduced Hessenberg matrix is nonderogatory, but the converse is not true.

A matrix that is not nonderogatory is called **derogatory**. A derogatory matrix A is similar to a direct sum of a number of companion matrices. That is, A is similar to

$$\begin{pmatrix} C_1 & & 0 \\ & \ddots & \\ 0 & & C_k \end{pmatrix}$$

where each C_i is a companion matrix, $k > 1$, and the characteristic polynomial of each C_i divides the characteristic polynomial of all the preceding C_i's. This form is also known as the **Frobenius canonical form**.

1.4.7 Diagonally Dominant Matrix

A matrix $A = (a_{ij})$ is **row diagonally dominant** if

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}| \qquad \text{for all } i$$ (za dato i !)

A **column diagonally dominant matrix** is similarly defined. The matrix

$$A = \begin{pmatrix} 10 & 1 & 1 \\ 1 & 10 & 1 \\ 1 & 1 & 10 \end{pmatrix}$$

is both row and column diagonally dominant.

Note: Sometimes in the literature of linear algebra, a matrix A having these properties is called a **strictly diagonally dominant matrix**.

1.4.8 Positive Definite Matrix

A symmetric matrix A is **positive definite** if for every nonzero vector x,

$$x^T Ax > 0$$

Let $x = (x_1, x_2, \ldots, x_n)^T$. Then $x^T A x = \sum_{i,j=1}^{n} a_{ij} x_i x_j$ is called the **quadratic form** associated with A.

Example 1.4.5

$$A = \begin{pmatrix} 2 & 1 \\ 1 & 5 \end{pmatrix}$$

$$x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

$$\begin{aligned} x^T A x &= (x_1 \quad x_2) \begin{pmatrix} 2 & 1 \\ 1 & 5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \\ &= (2x_1 + x_2, x_1 + 5x_2) \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \\ &= 2x_1^2 + 2x_1x_2 + 5x_2^2 \\ &= 2\left(x_1^2 + x_1x_2 + \frac{1}{4}x_2^2\right) + \frac{9}{2}x_2^2 \\ &= 2\left(x_1 + \frac{1}{2}x_2\right)^2 + \frac{9}{2}x_2^2 > 0 \end{aligned}$$

A positive semidefinite matrix is similarly defined. A symmetric matrix A is **positive semidefinite** if $x^T A x \geq 0$ for every x.

A commonly used notation for a symmetric positive definite (positive semidefinite) matrix is $A > 0$ (≥ 0).

Some Characterizations and Properties of Positive Definite Matrices

The following are some useful characterizations of positive definite matrices:

1. A symmetric matrix A is positive definite if and only if all its eigenvalues are positive. Note that in Example 1.4.5, the eigenvalues are 1.6972 and 5.3028.
2. A symmetric matrix A is positive definite iff all its leading principal minors are positive. There are n leading principal minors of an $n \times n$ matrix A. The ith leading principal minor is the determinant of the submatrix of A formed out of the first i rows and i columns.

   For example, if

$$A = \begin{pmatrix} 10 & 1 & 1 \\ 1 & 10 & 1 \\ 1 & 1 & 10 \end{pmatrix}$$

Then,

$$\text{First leading principal minor} = 10$$

$$\text{Second leading principal minor} = \det\begin{pmatrix} 10 & 1 \\ 1 & 10 \end{pmatrix} = 99$$

$$\text{Third leading principal minor} = \det A = 972$$

3. The sum of two positive definite matrices is positive definite.
4. If $A = (a_{ij})$ is positive definite, then $a_{ii} > 0$ for all i.
5. If $A = (a_{ij})$ is positive definite, then the largest element (in magnitude) of the whole matrix must lie on the diagonal.
6. A symmetric diagonally dominant matrix with positive diagonal entries is positive definite. Note that the matrix A, in the preceding example, is diagonally dominant.

Remarks: Note that (4) and (5) are only necessary conditions for a symmetric matrix to be positive definite. They can serve only as initial tests for positive definiteness. For example, the matrices

$$A = \begin{pmatrix} 4 & 1 & 1 & 1 \\ 1 & 0 & 1 & 2 \\ 1 & 1 & 2 & 3 \\ 1 & 2 & 3 & 4 \end{pmatrix}, \qquad B = \begin{pmatrix} 20 & 12 & 25 \\ 12 & 15 & 2 \\ 25 & 2 & 5 \end{pmatrix}$$

cannot be positive definite, because in matrix A, there is a zero entry on the diagonal and in B, the largest entry, 25, is not on the diagonal.

1.5 The Cayley-Hamilton Theorem

THEOREM 1.5.1 A square matrix A satisfies its own characteristic equation; that is, if $A = (a_{ij})$ is an $n \times n$ matrix and $P_n(\lambda)$ is the characteristic polynomial of A, then

$$P_n(A) \text{ is a zero matrix.}$$

Proof: See *Matrix Theory* by Franklin, pp. 113–114. ■

Example 1.5.1 Let

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 2 \end{pmatrix}$$

$$P_2(\lambda) = \lambda^2 - 2\lambda - 1$$

$$\begin{aligned} P_2(A) &= \begin{pmatrix} 1 & 2 \\ 2 & 5 \end{pmatrix} - 2\begin{pmatrix} 0 & 1 \\ 1 & 2 \end{pmatrix} - \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \\ &= \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \end{aligned}$$

1.6 Singular Values

Let A be $m \times n$ ($m \geq n$). Then the eigenvalues of the $n \times n$ symmetric matrix $A^T A$ are real and nonnegative. Let these eigenvalues be denoted by σ_i^2, where $\sigma_1^2 \geq \sigma_2^2 \geq$

$\cdots \geq \sigma_n^2$. Then $\sigma_1, \sigma_2, \ldots, \sigma_n$ are called the singular values of A. Every $m \times n$ matrix A can be decomposed into

$$A = U \Sigma V^T$$

where $U_{m\times m}$ and $V_{n\times n}$ are orthogonal and Σ is an $m \times n$ "diagonal" matrix. This decomposition is called the *singular value decomposition*, or SVD. The singular values $\sigma_i, i = 1, \ldots, n$, are the diagonal entries of Σ. The number of nonzero singular values is equal to the rank of the matrix A. (See Section 10.3.)

Example 1.6.1 Let

$$A = \begin{pmatrix} 0 & 1 \\ 2 & 2 \end{pmatrix}$$

$$A^T A = \begin{pmatrix} 4 & 4 \\ 4 & 5 \end{pmatrix}$$

The eigenvalues of $A^T A$ are $\left[\dfrac{9 \pm \sqrt{65}}{2}\right]$:

$$\sigma_1 = \sqrt{\left[\frac{9 + \sqrt{65}}{2}\right]}$$

$$\sigma_2 = \sqrt{\left[\frac{9 - \sqrt{65}}{2}\right]}$$

1.7 Vector and Matrix Norms

1.7.1 Vector Norms

Let

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

be an n-vector in $\mathbb{R}^n$. Then, a vector norm, denoted by the symbol $\|x\|$, is a real-valued *continuous* function of the components $x_1, x_2, \ldots, x_n$ of x, defined on $\mathbb{R}^n$; it has the following properties:

1. $\|x\| > 0$ for every nonzero vector x. $\|x\| = 0$ if x is a zero vector.
2. $\|\alpha x\| = |\alpha| \, \|x\|$ for all x on $\mathbb{R}^n$ and for all scalars α.
3. $\|x + y\| \leq \|x\| + \|y\|$ for all x and y in $\mathbb{R}^n$.

Property 3 is known as the *triangle inequality*.

Note:

$$\|-x\| = \|x\|$$
$$\big|\|x\| - \|y\|\big| \leq \|(x - y)\|$$

It is simple to verify that the following are vector norms.

1. $\|x\|_1 = |x_1| + |x_2| + \cdots + |x_n|$ (*sum norm* or *one norm*)
2. $\|x\|_2 = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$ (*Euclidean norm* or *two norm*)
3. $\|x\|_\infty = \max_i |x_i|$ (*infinity norm* or *maximum norm*)

In general, if p is a real number greater than or equal to 1, the **p-norm**, or **Hölder norm**, is defined by

$$\|x\|_p = (|x_1|^p + \cdots + |x_n|^p)^{1/p}$$

Unless otherwise stated, by $\|x\|$ we will mean $\|x\|_2$.

Example 1.7.1 Let $x = (1, 1, -2)^T$. Then

$$\|x\|_1 = 4$$
$$\|x\|_2 = \sqrt{1^2 + 1^2 + (-2)^2} = \sqrt{6}$$
$$\|x\|_\infty = 2$$

An important property of the Hölder-norm is the **Hölder inequality**

$$|x^T y| \leq \|x\|_p \, \|y\|_q$$

where

$$\frac{1}{p} + \frac{1}{q} = 1$$

A special case of the Hölder inequality is the **Cauchy-Schwarz inequality**,

$$|x^T y| \leq \|x\|_2 \, \|y\|_2$$

—that is,

$$\left|\sum_{j=1}^{n} x_j y_j\right| \leq \sqrt{\sum_{j=1}^{n} x_j^2} \sqrt{\sum_{j=1}^{n} y_j^2}$$

Equivalent Property of the Vector Norms

All vector norms are *equivalent* in the sense that there exist positive constants α and β such that

$$\alpha \|x\|_\mu \leq \|x\|_\nu \leq \beta \|x\|_\mu$$

Tako da ako se u nekom izrazu norme dele, izbor norme je proizvoljan!

for all x.

For the 2, 1, or ∞ norms, we can compute α and β easily:

$$\|x\|_2 \leq \|x\|_1 \leq \sqrt{n} \, \|x\|_2$$
$$\|x\|_\infty \leq \|x\|_2 \leq \sqrt{n} \, \|x\|_\infty$$
$$\|x\|_\infty \leq \|x\|_1 \leq n \, \|x\|_\infty$$

1.7.2 Matrix Norms

Let A be an $m \times n$ matrix. Then, analogous to the vector norm, we define a matrix norm $\|A\|$ with the following properties:

1. $\|A\| > 0$; $\|A\| = 0$ if A is a zero matrix.
2. $\|\alpha A\| = |\alpha| \, \|A\|$ for any scalar α.
3. $\|A + B\| \leq \|A\| + \|B\|$.

Subordinate Matrix Norms

Given a matrix A and a vector norm $\|\cdot\|$, a nonnegative number defined by:

$$\|A\|_p = \max_{x \neq 0} \frac{\|Ax\|_p}{\|x\|_p}$$

satisfies all the properties of a matrix norm. This norm is called the matrix norm **subordinate** to the vector norm.

A very useful and frequently used property of a subordinate matrix norm (*we shall sometimes call it the p-norm of a matrix A*) is

$$\|Ax\|_p \leq \|A\|_p \, \|x\|_p$$

This property easily follows from the definition of p-norms. Note that

$$\|A\|_p \geq \frac{\|Ax\|_p}{\|x\|_p}$$

for any particular nonzero vector x. Multiplying both sides by $\|x\|_p$ gives the original inequality.

> The two most easily computable p-norms are
>
> $$\|A\|_1 = \max_{1 \leq j \leq n} \sum_{i=1}^{m} |a_{ij}|$$
>
> (maximum column-sum norm)
>
> $$\|A\|_\infty = \max_{1 \leq i \leq m} \sum_{j=1}^{n} |a_{ij}|$$
>
> (maximum row-sum norm)

Example 1.7.2

$$A = \begin{pmatrix} 1 & -2 \\ 3 & 4 \\ -5 & 6 \end{pmatrix}$$

$$\|A\|_1 = 12$$

$$\|A\|_\infty = 11$$

Another useful p-norm is the *spectral norm*.

DEFINITION 1.7.1 The **spectral norm** is

$$\|A\|_2 = \max_{x \neq 0} \frac{\|Ax\|_2}{\|x\|_2}$$

It can be shown that

$$\|A\|_2 = \sqrt{\text{maximum eigenvalue of } A^T A}$$

(Note that the eigenvalues of $A^T A$ are real and non-negative.)

simetrična m

Example 1.7.3

$$A = \begin{pmatrix} 2 & 5 \\ 1 & 3 \end{pmatrix}$$

$$A^T A = \begin{pmatrix} 5 & 13 \\ 13 & 34 \end{pmatrix}$$

The eigenvalues of $A^T A$ are 0.0257 and 38.9743.

$$\|A\|_2 = \sqrt{38.9743} = 6.2429$$

The Frobenius Norm

An important matrix norm consistent with the vector norm $\|x\|_2$ is the Frobenius norm:

$$\|A\|_F = \left[\sum_{j=1}^{n} \sum_{i=1}^{m} |a_{ij}|^2 \right]^{1/2}$$

A matrix norm $\|\cdot\|_M$ and a vector norm $\|\cdot\|_v$ are consistent if

$$\|Ax\|_v \leq \|A\|_M \, \|x\|_v$$

> For any subordinate matrix norm and the Frobenius norm we can prove that
>
> $$\|AB\| \leq \|A\| \, \|B\|$$

Notes:

1. For the identity matrix I,

   $$\|I\|_F = \sqrt{n}$$

   whereas $\|I\|_1 = \|I\|_2 = \|I\|_\infty = 1$.
2. $\|A\|_F^2 = \text{trace}(A^T A)$, where trace$(A)$ is defined as the *sum* of the diagonal entries of A; that is, if $A = (a_{ij})$, then trace$(A) = a_{11} + a_{22} + \cdots + a_{nn}$.

Equivalence Property of Matrix Norms

As in the case of vector norms, the matrix norms are also related. There exist scalars α and β such that

$$\alpha \, \|A\|_\mu \leq \|A\|_\mu \leq \beta \, \|A\|_\mu$$

In particular, the following inequalities relating various matrix norms are true and are used very frequently in practice.

THEOREM 1.7.1 Let A be $m \times n$.

1. $\frac{1}{\sqrt{n}} \|A\|_\infty \le \|A\|_2 \le \sqrt{m}\,\|A\|_\infty$.
2. $\|A\|_2 \le \|A\|_F \le \sqrt{n}\,\|A\|_2$.
3. $\frac{1}{\sqrt{m}} \|A\|_1 \le \|A\|_2 \le \sqrt{n}\,\|A\|_1$.
4. $\|A\|_2 \le \sqrt{\|A\|_1\,\|A\|_\infty}$.

We prove inequalities 1 and 2 and leave the rest as exercises.

Proof of (1): By definition:

$$\|A\|_\infty = \max_{x\neq 0} \frac{\|Ax\|_\infty}{\|x\|_\infty}$$

Again, from the equivalence property of the vector-norms, we have:

$$\|Ax\|_\infty \le \|Ax\|_2 \qquad \text{and} \qquad \|x\|_2 \le \sqrt{n}\,\|x\|_\infty$$

From the second inequality we get

$$\frac{1}{\|x\|_\infty} \le \frac{\sqrt{n}}{\|x\|_2}$$

It therefore follows that

$$\frac{\|Ax\|_\infty}{\|x\|_\infty} \le \sqrt{n}\,\frac{\|Ax\|_2}{\|x\|_2}$$

or

$$\max_{x\neq 0} \frac{\|Ax\|_\infty}{\|x\|_\infty} \le \sqrt{n} \max_{x\neq 0} \frac{\|Ax\|_2}{\|x\|_2} = \sqrt{n}\,\|A\|_2$$

that is,

$$\frac{1}{\sqrt{n}}\,\|A\|_\infty \le \|A\|_2$$

The first part is proved. To prove the second part, we again use the definition of $\|A\|_2$ and the appropriate equivalence property of the vector norms.

$$\|A\|_2 = \max_{x\neq 0} \frac{\|Ax\|_2}{\|x\|_2}$$

$$\|Ax\|_2 \le \sqrt{m}\,\|Ax\|_\infty$$

$$\|x\|_\infty \le \|x\|_2$$

Thus,

$$\frac{\|Ax\|_2}{\|x\|_2} \le \sqrt{m}\,\frac{\|Ax\|_\infty}{\|x\|_\infty}$$

So

$$\max_{x\neq 0} \frac{\|Ax\|_2}{\|x\|_2} \le \sqrt{m} \max_{x\neq 0} \frac{\|Ax\|_\infty}{\|x\|_\infty} \qquad \text{or} \qquad \|A\|_2 \le \sqrt{m}\|A\|_\infty$$

The proof of (1) is now complete.

Proof of (2): We prove (2) using a different technique. Recall that

$$\|A\|_F^2 = \operatorname{trace}(A^T A)$$

Since $A^T A$ is symmetric, there exists an orthogonal matrix O such that

$$O^T (A^T A) O = D = \operatorname{diag}(d_1, \ldots, d_n)$$

(See Theorem 8.2.8.)

The trace is invariant under a similarity transformation (see Exercise 35d). We then have

$$\operatorname{trace}(A^T A) = \operatorname{trace}(D) = d_1 + \cdots + d_n$$

Let $d_k = \max_i(d_i)$. Then, since $d_1, \ldots, d_n$ are also the eigenvalues of $A^T A$, we have

$$\|A\|_2^2 = d_k$$

Thus,

$$\|A\|_F^2 = \operatorname{trace}(A^T A) = d_1 + \cdots + d_n \ge d_k = \|A\|_2^2$$

To prove the other part, we note that

$$\|A\|_F^2 = d_1 + \cdots + d_n \le d_k + d_k + \cdots + d_k = nd_k$$

that is, $\|A\|_F^2 \le nd_k = n\|A\|_2^2$. So, $\|A\|_F \le \sqrt{n}\,\|A\|_2$. ■

1.7.3 Convergence of a Matrix Sequence and Convergent Matrices

A sequence of vectors $v^{(1)}, v^{(2)}, \ldots$ is said to converge to the vector v if

$$\lim_{k\to\infty} v_i^{(k)} = v_i, \qquad i = 1, \ldots, n$$

A sequence of matrices $A^{(1)}, A^{(2)}, \ldots$ is said to converge to the matrix $A = (a_{ij})$ if

$$\lim_{k\to\infty} a_{ij}^{(k)} = a_{ij}, \quad i, j = 1, 2, \ldots, n$$

If the sequence $\{A^{(k)}\}$ converges to A, then we write

$$\lim_{k\to\infty} A^{(k)} = A$$

We now state, without proof, necessary and sufficient conditions for the convergence of vector and matrix sequences. The proofs can easily be worked out.

THEOREM 1.7.2 The sequence $v^{(1)}, v^{(2)}, \ldots$ converges to v if and only if

$$\lim_{k\to\infty} \|v^{(k)} - v\| = 0$$

A similar theorem holds for a matrix sequence.

THEOREM 1.7.3 The sequence of matrices $A^{(1)}, A^{(2)}, \ldots$ converges to the matrix A if and only if

$$\|A - A^{(r)}\| \to 0$$

where $\| \ \|$ is any matrix norm.

We now state and prove a result on the convergence of the sequence of powers of a matrix to the zero matrix.

THEOREM 1.7.4 The sequence $A, A^2, \ldots$ of the powers of the matrix A converges to the zero matrix iff $|\lambda_i| < 1$ for each eigenvalue λ_i of A.

Proof: For every $n \times n$ matrix A, there exists a nonsingular matrix T such that

$$T^{-1}AT = J = \begin{pmatrix} J_1 & & 0 \\ & \ddots & \\ 0 & & J_r \end{pmatrix}$$

where each J_i has the form

$$J_i = \begin{pmatrix} \lambda_i & 1 & & & \\ & \lambda_i & 1 & 0 & \\ & & \ddots & \ddots & \\ & 0 & & \ddots & 1 \\ & & & & \lambda_i \end{pmatrix}$$

Jordan matrice

It is an easy computation to show that

$$J_i^k = \begin{pmatrix} \lambda_i^k & k\lambda_i^{k-1} & \binom{k}{2}\lambda_i^{k-2} & \cdots & \binom{k}{n-1}\lambda_i^{k-n+1} \\ & \lambda_i^k & k\lambda_i^{k-1} & \cdots & \vdots \\ & & \ddots & \ddots & \vdots \\ & & & \ddots & k\lambda_i^{k-1} \\ & & & & \lambda_i^k \end{pmatrix}$$

Jordan canonical form.

from which we see that $J_i^k \to 0$ if and only if $|\lambda_i| < 1$. This means that $\lim_{k\to\infty} A^k = 0$ iff $|\lambda_i| < 1$ for each i. ■

The above form is called the **Jordan canonical form** of A, and the diagonal block matrices are called **Jordan matrices**.

DEFINITION 1.7.2 A matrix A is called a **convergent matrix** if $A^k \to 0$ as $k \to \infty$.

We now prove a **sufficient** condition for a matrix A to be a convergent matrix in terms of a norm of the matrix A. We first prove the following result.

A Relationship between Norms and Eigenvalues

THEOREM 1.7.5 Let λ be an eigenvalue of a matrix A. Then for any subordinate matrix norm,

$$|\lambda| \le \|A\|$$

Proof: By definition, there exists a nonzero vector x such that

$$Ax = \lambda x$$

Taking the norm of each side, we have

$$\|Ax\| = \|\lambda x\| = |\lambda|\,\|x\|$$

However, $\|Ax\| \le \|A\|\,\|x\|$, so $|\lambda|\,\|x\| = \|Ax\| \le \|A\|\,\|x\|$, giving $|\lambda| \le \|A\|$. ■

DEFINITION 1.7.3 The quantity $\rho(A)$ defined by

$$\rho(A) = \max_i |\lambda_i|$$

where $\lambda_i, \ldots, \lambda_n$ are the eigenvalues of A, is called the **spectral radius** of A.

As a particular case of Theorem 1.7.5, we have:

$$\boxed{\rho(A) \le \|A\|}$$

In view of this result, we can now state the following corollary.

COROLLARY 1.7.1 A matrix A is convergent if $\|A\| < 1$, where $\|\cdot\|$ is a subordinate matrix norm.

Convergence of an Infinite Matrix Series

THEOREM 1.7.6 The matrix series

$$I + A + A^2 + \cdots$$

converges to $(I - A)^{-1}$ if A is a convergent matrix.

Proof: Since A is a convergent matrix, $A^k \to 0$ as $k \to \infty$. Then from Theorem 1.7.4, we must have $|\lambda_i| < 1$ for each eigenvalue λ_i of A. This means that the matrix $(I - A)$ is nonsingular, since the eigenvalues of $I - A$ are $1 - \lambda_1, 1 - \lambda_2, \ldots, 1 - \lambda_n$, and a matrix A is nonsingular if and only if its eigenvalues are nonzero. This is because the eigenvalues of $(I - A)$ are $1 - \lambda_1, 1 - \lambda_2, \ldots, 1 - \lambda_n$, and $|\lambda_i| < 1$ implies that none of them is zero. Thus, from the identity

$$(I - A)(I + A + A^2 + \cdots + A^k) = I - A^{k+1}$$

we have

$$(I + A + A^2 + \cdots + A^k) = (I - A)^{-1} - (I - A)^{-1}A^{k+1}$$

Since A is a convergent matrix,

$$A^{k+1} \to 0 \text{ as } k \to \infty$$

Thus, when $k \to \infty$, $I + A + A^2 + \cdots \to (I - A)^{-1}$. ■

1.7.4 Norms and Inverses

While analyzing the errors in an algorithm, we sometimes need to know, given a nonsingular matrix A, how much it can be perturbed so that the perturbed matrix $A + E$ is nonsingular and how to estimate the error in the inverse of the perturbed matrix.

We start with the identity matrix. **In the following, $\|\ \|$ is a subordinate matrix norm for which $\|I\| = 1$.**

THEOREM 1.7.7 Let $\|E\| < 1$; then $(I - E)$ is nonsingular and

$$\|(I - E)^{-1}\| \le (1 - \|E\|)^{-1}$$

Proof: Let $\lambda_1, \ldots, \lambda_n$ be the eigenvalues of E. Then the eigenvalues of $I - E$ are

$$1 - \lambda_1, 1 - \lambda_2, \ldots, 1 - \lambda_n$$

Since $\|E\| < 1$, $|\lambda_i| < 1$ for each i. Thus, none of the quantities $1 - \lambda_1, 1 - \lambda_2, \ldots, 1 - \lambda_n$ is zero. This proves that $I - E$ is nonsingular.

To prove the second part, we note that since $\|E\| < 1$, by Theorem 1.7.6

$$(I - E)^{-1} = I + E + E^2 + \cdots = \lim_{k\to\infty}\left(\sum_{i=0}^{k} E^i\right)$$

Because

$$\|E^i\| \le \|E\|^i$$

taking the norm on both sides, we have

$$\|(I - E)^{-1}\| \le \sum_{i=0}^{\infty} \|E\|^i = (1 - \|E\|)^{-1} \text{ (since } \|I\| = 1)$$

(Note that the infinite series $1 + x + x^2 + \cdots$ converges to

$$\frac{1}{1 - x}$$

if and only if $|x| < 1$.) ■

THEOREM 1.7.8 If $\|E\| < 1$ then

$$\|(I - E)^{-1} - I\| \le \frac{\|E\|}{1 - \|E\|}$$

Proof: For any two nonsingular matrices A and B, we can write

$$A^{-1} - B^{-1} = A^{-1}(B - A)B^{-1}$$

In this equation, substitute

$$B = I$$

and $A = I - E$; then we have

$$(I - E)^{-1} - I = (I - E)^{-1}E$$

(Note that $I^{-1} = I$.) Taking the norm on both sides yields

$$\|(I - E)^{-1} - I\| \le \|I - E\|^{-1}\|E\|$$

From Theorem 1.7.7 we know

$$\|I - E\|^{-1} \le (1 - \|E\|)^{-1}$$

and thus we have the result. ■

$$\|(I-E)^{-1} - I\| \le \frac{\|E\|}{1-\|E\|}$$

Implication of the Result

If the matrix E is very small, then $1 - \|E\|$ is close to unity. Thus the preceding result implies that if we invert a slightly perturbed identity matrix, then the error in the inverse of the perturbed matrix does not exceed the norm of the perturbation.

THEOREM 1.7.9 Let A be nonsingular and let $\|A^{-1}E\| < 1$. Then $A - E$ is nonsingular, and

$$\frac{\|A^{-1} - (A - E)^{-1}\|}{\|A^{-1}\|} \le \frac{\|A^{-1}E\|}{1 - \|A^{-1}E\|}$$

Proof: We can write

$$A - E = A(I - A^{-1}E)$$

Since $\|A^{-1}E\| < 1$, from Theorem 1.7.7, we have $I - A^{-1}E$ is nonsingular. Thus, $A - E$, which is the product of the nonsingular matrices A and $I - A^{-1}E$, is also nonsingular.

To prove the second part, we again recall the identity

$$A^{-1} - B^{-1} = A^{-1}(B - A)B^{-1}$$

Substituting $B = A - E$, we then have

$$A^{-1} - (A - E)^{-1} = -A^{-1}E(A - E)^{-1}$$

Taking the norm on each side yields

$$\|A^{-1} - (A - E)^{-1}\| \le \|A^{-1}E\|\|(A - E)^{-1}\|$$

Because

$$\begin{aligned} B &= A - (A - B) \\ &= A[I - A^{-1}(A - B)] \\ B^{-1} &= [I - A^{-1}(A - B)]^{-1}A^{-1} \end{aligned}$$

(Note that $(XY)^{-1} = Y^{-1}X^{-1}$.) If we substitute $B = A - E$, we have

$$(A - E)^{-1} = [I - A^{-1}E]^{-1}A^{-1}$$

Taking norms, we get

$$\|(A - E)^{-1}\| \le \|A^{-1}\|\,\|(I - A^{-1}E)^{-1}\|$$

But from Theorem 1.7.7, we know that

$$\|(I - A^{-1}E)^{-1}\| \le (1 - \|A^{-1}E\|)^{-1}$$

So, we have

$$\|(A - E)^{-1}\| \le \frac{\|A^{-1}\|}{1 - \|A^{-1}E\|}$$

Therefore,

$$\|A^{-1} - (A - E)^{-1}\| \le \frac{\|A^{-1}E\| \cdot \|A^{-1}\|}{1 - \|A^{-1}E\|}$$

or

$$\frac{\|A^{-1} - (A - E)^{-1}\|}{\|A^{-1}\|} \le \frac{\|A^{-1}E\|}{1 - \|A^{-1}E\|}$$ ■

Note: Theorem 1.7.8 is important in the context of estimating the largest error in the inverse of a matrix whose entries are not precisely known.

1.8 Norm Invariant Properties of Orthogonal Matrices

We conclude the chapter by listing some very useful norm properties of orthogonal matrices that are often used in practice.

THEOREM 1.8.1 Let O be an orthogonal matrix. Then

$$\|O\|_2 = 1$$

Proof:

$$\|O\|_2 = \sqrt{\text{maximum eigenvalue of } (O^T O)} = \sqrt{\rho(O^T O)}$$
$$= \sqrt{\rho(I)} = 1$$ ■

THEOREM 1.8.2

$$\|AO\|_2 = \|A\|_2$$

Proof:

$$\|AO\|_2 = \sqrt{\rho(O^T A^T AO)}$$
$$= \sqrt{\rho(A^T A)} = \|A\|_2$$

(*Note that the spectral radius is invariant under similarity transformation.* See Chapter 8.) ■

THEOREM 1.8.3

$$\|AO\|_F = \|A\|_F$$

Proof: $\|AO\|_F^2 = \text{trace}(O^T A^T AO) = \text{trace}(A^T A) = \|A\|_F^2$. ■

1.9 Review and Summary

The basic concepts that are required for understanding the rest of the book have been briefly summarized in this chapter. The most important ones are as follows:

1. *Special matrices:* Diagonal, triangular, orthogonal, permutation, Hessenberg, tridiagonal, diagonally dominant, and positive definite matrices were defined, and their important properties were discussed.
2. *Vector and matrix norms:* Some important matrix norms are row-sum norm, column-sum norm, Frobenius norm, and spectral norm. The relationship between different matrix norms is stated and proved in Theorem 1.7.1.

   Of special importance is the norm property of orthogonal matrices. Three simple but important results were stated and proved in Section 1.8 (Theorems 1.8.1, 1.8.2, and 1.8.3). These results say that

   (a) the spectral norm of an orthogonal matrix is 1, and
   (b) the spectral and the Frobenius norms remain invariant under matrix multiplication.
3. *Convergence of a matrix sequence:* The notion of the convergence of the sequence of matrix powers $\{A^k\}$ is important in the study of convergence of iterative methods for linear systems. The most important results in this context are as follows:

   (a) The sequence $\{A^k\}$ converges to the zero matrix iff $|\lambda_i| < 1$ for each eigenvalue of λ_i of A (Theorem 1.7.4).
   (b) The sequence $\{A^k\}$ converges to the zero matrix if $\|A\| < 1$ (corollary to Theorem 1.7.5).
4. *Norms and inverses:* If a nonsingular matrix A is perturbed by a matrix E, it is sometimes of interest to know if the perturbed matrix $A + E$ remains nonsingular and how to estimate the error in the inverse of $A + E$. Three theorems (Theorems 1.7.7, 1.7.8, and 1.7.9) are proved in this context in Section 1.7.4. These results play an important role in the perturbation analysis of linear systems (Chapter 6).

EXERCISES

Sections 1.2 and 1.3

1. Prove each of the following.

   (a) A set of n linearly independent vectors in $\mathbb{R}^n$ is a basis for $\mathbb{R}^n$.
   (b) The set $\{e_1, e_2, \ldots, e_n\}$ is a basis of $\mathbb{R}^n$.

(c) A set of m vectors in $\mathbb{R}^n$, where $m > n$, is linearly dependent.
(d) Any two bases in $\mathbb{R}^n$ have the same number of vectors.
(e) $\text{span}\{v_1, \ldots, v_k\}$ is a subspace of $\mathbb{R}^n$, where $\text{span}\{v_1, \ldots, v_k\}$ is the set of linear combinations of the k vectors $v_1, \ldots, v_k$ from a vector space $\mathbb{R}^n$.
(f) $\text{span}\{v_1, \ldots, v_k\}$ is the smallest subspace of $\mathbb{R}^n$ containing $v_1, \ldots, v_k$.

2. Prove that if $S = \{s, \ldots, s_k\}$ is an orthogonal set of nonzero vectors, then S is linearly independent.
3. Let S be an m-dimensional subspace of $\mathbb{R}^n$. Then prove that S has an orthonormal basis. (*Hint:* Let $\{v_1, \ldots, v_m\}$ be a given basis of S. Define a set of vectors $\{u_k\}$ by:

$$u_1 = \frac{v_1}{\|v_1\|}$$

$$u_{k+1} = \frac{v'_{k+1}}{\|v'_{k+1}\|}$$

where

$$v'_{k+1} = v_{k+1} - (v_{k+1}^T u_1)u_1 - (v_{k+1}^T u_2)u_2 - \cdots - (v_{k+1}^T u_k)u_k, \quad k = 1, 2, \ldots, m-1$$

Then show that $\{u_1, \ldots, u_m\}$ is an orthonormal basis of S. This is the classical **Gram-Schmidt process.**)
4. Using the Gram-Schmidt process, construct an orthonormal basis of $\mathbb{R}^3$.
5. Construct an orthonormal basis of $\mathbb{R}(A)$, where

$$A = \begin{pmatrix} 1 & 2 \\ 2 & 3 \\ 4 & 5 \end{pmatrix}$$

6. Let S_1 and S_2 be two subspaces of $\mathbb{R}^n$. Prove that

$$\dim(S_1 + S_2) = \dim(S_1) + \dim(S_2) - \dim(S_1 \cap S_2)$$

7. Prove Theorem 1.3.1 on the properties of the determinant of a matrix.
8. Prove each of the following.
(a) $\text{null}(A) = 0$ iff A has linearly independent columns.
(b) $\text{rank}(A) = \text{rank}(A^T)$.
(c) $\text{rank}(A) + \text{null}(A) = n$.
(d) If A is $m \times n$ and $m < n$, then $\text{rank}(A) \le m$.
(e) If A and B are $m \times n$ and $n \times p$ matrices, then

$$\text{rank}(AB) \le \min\{\text{rank}(A), \text{rank}(B)\}$$

(f) The rank of a matrix remains unchanged when the matrix is multiplied by an invertible matrix.
(g) If $B = UAV$, where U and V are invertible, then $\text{rank}(B) = \text{rank}(A)$.
(h) $N(A) = R(A^T)^\perp$ and $R(A)^\perp = N(A^T)$.

9. Let A be $m \times n$. Then A has rank 1 iff A can be written as $A = ab^T$, where a and b are nonzero column vectors.

10. Prove the following basic facts on nonsingularity and the inverse of A:
(a) $(A^{-1})^{-1} = A$
(b) $(A^T)^{-1} = (A^{-1})^T$
(c) $(cA)^{-1} = \frac{1}{c}A^{-1}$, where c is a nonzero scalar
(d) $(AB)^{-1} = B^{-1}A^{-1}$
11. Suppose a matrix A can be written as

$$A = LU$$

where L is a lower triangular matrix with 1s along the diagonal and $U = (u_{ij})$ is an upper triangular matrix. Prove that

$$\det A = \prod_{i=1}^{n} u_{ii}$$

12. Let $A = \begin{pmatrix} A_1 & 0 \\ A_2 & A_3 \end{pmatrix}$, where A_1 and A_3 are square. Prove that $\det(A) = \det(A_1)\det(A_3)$.
13. Suppose that A can be written as

$$A = LDL^T$$

where L is lower triangular with 1s as diagonal entries and $D = \text{diag}(d_{11}, \ldots, d_{nn})$ is a diagonal matrix. Prove that the leading principal minors (determinants) of A are $d_{11}, d_{11}d_{22}, \ldots, d_{11} \ldots d_{nn}$.
14. (a) Show that if P_S is an orthogonal projection onto S, then $I - P_S$ is the orthogonal projection onto $S_\perp$.
(b) Prove each of the following.
(i) $P_A = A(A^TA)^{-1}A^T$
(ii) $P_N = I - A(A^TA)^{-1}A^T$
(c) Prove each of the following.
(i) $b_R = P_A b$
(ii) $b_N = P_N b$
15. (a) Find P_A and P_N for the matrices

$$A = \begin{pmatrix} 1 & 2 \\ 2 & 3 \\ 0 & 0 \end{pmatrix}, \qquad A = \begin{pmatrix} 1 & 1 \\ 10^{-4} & 0 \\ 0 & 10^{-4} \end{pmatrix}$$

(b) For the vector $b = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$, find b_R and b_N for each of the preceding matrices.
(c) Find an orthonormal basis for each of the preceding matrices using the Gram-Schmidt process and then find P_A, P_N, b_R, and b_N. For a description of the Gram-Schmidt algorithm, see Chapter 7 or Exercise 3 of this chapter.
16. Let A be an $m \times n$ matrix with rank r. Consider the **singular-value decomposition** of A:

$$\begin{aligned} A &= U\Sigma V^T \\ &= (U_r, \hat{U}_r)\Sigma(V_r, \hat{V}_r)^T \end{aligned}$$

where $U_r = (u_1, \ldots, u_r)$ and $V_r = (v_1, \ldots, v_r)$, and Σ is an $m \times n$ diagonal matrix.

Then prove the following.

(a) $V_r V_r^T$ is the orthogonal projection onto $N(A)^\perp = R(A^T)$.
(b) $U_r U_r^T$ is the orthogonal projection onto $R(A)$.
(c) $(\hat{U}_r)(\hat{U}_r)^T$ is the orthogonal projection onto $R(A)^\perp = N(A^T)$.
(d) $\hat{V}_r(\hat{V}_r)^T$ is the orthogonal projection onto $N(A)$.

17. (Distance between two subspaces.) Let S_1 and S_2 be two subspaces of $\mathbb{R}^n$ such that $\dim(S_1) = \dim(S_2)$. Let P_1 and P_2 be the orthogonal projections onto S_1 and S_2 respectively. Then $\|P_1 - P_2\|_2$ is defined to be the distance between S_1 and S_2. Prove that the distance between S_1 and S_2, $\text{dist}(S_1, S_2) = \sin(\theta)$ where θ is the angle between S_1 and S_2 ($\|A\|_2 = \sqrt{\text{maximum eigenvalue of } A^T A}$).
18. Prove that if P_S is the orthogonal projection onto S, then $I - 2P_S$ is orthogonal.

Sections 1.4–1.6

19. Prove the following.
    (a) The product of two upper (lower) triangular matrices is an upper (lower) triangular matrix. In general, if $A = (a_{ij})$ is an upper (lower) triangular matrix, then $p(A)$, where $p(x)$ is a polynomial, is an upper (lower) triangular matrix whose diagonal elements are $p(a_{ii})$, $i = 1, \ldots, n$.
    (b) The inverse of a lower (upper) triangular matrix is another lower (upper) triangular matrix whose diagonal entries are the reciprocals of the diagonal entries of the triangular matrix.
    (c) The determinant of a triangular matrix is the product of its diagonal entries.
    (d) The eigenvalues of a triangular matrix are its diagonal entries.
    (e) If $A \in \mathbb{R}^{n\times n}$ is strictly upper triangular, then $A^n = 0$. ($A = (a_{ij})$ is strictly upper triangular if A is upper triangular and $a_{ii} = 0$ for each i.)
    (f) The inverse of a nonsingular matrix A can be written as a polynomial in A (use the Cayley-Hamilton theorem).
20. Prove that the product of an upper Hessenberg matrix and an upper triangular matrix is an upper Hessenberg matrix.
21. Prove that a symmetric Hessenberg matrix is symmetric tridiagonal.
22. A square matrix $A = (a_{ij})$ is a band matrix of bandwidth $2k + 1$ if $|i - j| > k$ implies that $a_{ij} = 0$. What are the bandwidths of tridiagonal and pentadiagonal matrices? Is the product of two banded matrices having the same bandwidth a banded matrix of the same bandwidth? Give reasons for your answer.
23. (a) Show that the matrix

$$H = I - 2\frac{uu^T}{u^T u}$$

where u is a column vector, is orthogonal. (The matrix H is called a **Householder matrix.**)

(b) Show that the matrix

$$J = \begin{pmatrix} c & s \\ -s & c \end{pmatrix}, \qquad \text{where } c^2 + s^2 = 1$$

is orthogonal. (The matrix J is called a **Givens matrix.**)

(c) Prove that the product of two orthogonal matrices is an orthogonal matrix.
(d) Prove that a triangular matrix that is orthogonal is diagonal.

24. Let A and B be two symmetric matrices.
    (a) Prove that $(A + B)$ is symmetric.
    (b) Prove that AB is not necessarily symmetric. Derive a condition under which AB is symmetric.
    (c) If A and B are symmetric positive definite, prove that $(A + B)$ is positive definite. Is AB also positive definite? Give reasons for your answer. When is $(A - B)$ symmetric positive definite?
25. Let $A = (a_{ij})$ be an $n \times n$ symmetric positive definite matrix. Prove the following.
    (a) Each diagonal entry of A must be positive.
    (b) A is nonsingular.
    (c) $(a_{ij})^2 < a_{ii}a_{jj}$ for $i = 1, \ldots, n$, $j = 1, \ldots, n$, $i \neq j$.
    (d) The largest element of the matrix must lie on the diagonal.
26. Let A be a symmetric positive definite matrix and x be a nonzero n-vector. Prove that $A + xx^T$ is positive definite.
27. Prove that a diagonally dominant matrix is nonsingular, and a diagonally dominant symmetric matrix with positive diagonal entries is positive definite.
28. Prove that if the eigenvalues of a matrix A are all distinct, then A is nonderogatory.
29. Prove that a symmetric matrix A is positive definite iff A^{-1} exists and is positive definite.
30. Let A be an $m \times n$ matrix ($m \geq n$) having full rank. Then $A^T A$ is positive definite.
31. Prove the following basic facts on the eigenvalues and eigenvectors.
    (a) A matrix A is nonsingular iff A does not have a zero eigenvalue. (*Hint:* $\det(A) = \lambda_1\lambda_2 \ldots \lambda_n$.)
    (b) The eigenvalues of A^T and A are the same.
    (c) If two matrices have the same eigenvalues, they need not be similar (construct your own example to show this).
    (d) A symmetric matrix is positive definite iff all its eigenvalues are positive.
    (e) The eigenvalues of a triangular matrix are its diagonal elements.
    (f) The eigenvalues of an orthogonal matrix have moduli 1.
    (g) Let A be a symmetric matrix and let Q be orthogonal such that $Q^T AQ$ is diagonal. Then show that the columns of Q are the eigenvectors of A.
    (h) The eigenvectors of a symmetric matrix can be chosen to be orthonormal.
32. Let A be a symmetric matrix with eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$, and orthonormal eigenvectors $v_1, v_2, \ldots, v_n$. Then show that

$$A = \lambda_1 v_1 v_1^T + \lambda_2 v_2 v_2^T + \cdots + \lambda_n v_n v_n^T$$

33. Let H be an unreduced upper Hessenberg matrix. Let $X = (x_1, \ldots, x_n)$ be defined by

$$x_1 = e_1 = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \qquad x_{i+1} = Hx_i, \qquad i = 1, 2, \ldots, n-1$$

Then prove that X is nonsingular and $X^{-1}HX$ is a companion matrix (upper Hessenberg companion matrix.)

34. What are the singular values of a symmetric matrix? What are the singular values of a symmetric positive definite matrix? Prove that a square matrix A is nonsingular iff it has no zero singular value.

35. Let $\text{trace}(A) = \sum_{i=1}^{n} a_{ii}$. Then prove the following.
    (a) $\text{trace}(AB) = \text{trace}(BA)$.
    (b) $\text{trace}(AA^T) = \sum_{i=1}^{m}\sum_{j=1}^{n} |a_{ij}|^2$, where $A = (a_{ij})$ is $m \times n$.
    (c) $\text{trace}(A + B) = \text{trace}(A) + \text{trace}(B)$.
    (d) $\text{trace}(TAT^{-1}) = \text{trace}(A)$.

Sections 1.7–1.8

36. Show that $\|x\|_1$, $\|x\|_\infty$, $\|x\|_2$ (as defined in Section 1.7) are vector norms.
37. Show that, if x and y are two vectors, then
$$\big|\|x\| - \|y\|\big| \le \|x - y\| \le \|x\| + \|y\|$$
38. If x and y are two n-vectors, then prove that
    (a) $|x^T y| \le \|x\|_2 \, \|y\|_2$ (*Cauchy-Schwarz inequality*)
    (b) $\|xy^T\|_2 = \|x\|_2 \, \|y\|_2$
39. Let x and y be two orthogonal vectors; then prove that
$$\|x + y\|_2^2 = \|x\|_2^2 + \|y\|_2^2$$
40. Prove that for any vector x,
$$\|x\|_\infty \le \|x\|_2 \le \|x\|_1$$
41. Prove that $\|A\|_1$, $\|A\|_\infty$, $\|A\|_2$ are matrix norms.
42. Let $A = (a_{ij})$ be $m \times n$. Define $A_\ell = \max_{ij} |a_{ij}|$. Is A_ℓ a matrix norm? Give reasons for your answer.
43. (a) Prove that the vector length is preserved by orthogonal matrix multiplication. That is, if $x \in \mathbb{R}^n$ and $Q \in \mathbb{R}^{n\times n}$ be orthogonal, then $\|Qx\|_2 = \|x\|_2$ (*isometry lemma*).
    (b) Is the statement in part (a) true if the 1 and infinity norms are used? Give reasons. What if the Frobenius norm is used?
44. Prove that $\|I\|_2 = 1$. Prove that $\|I\|_F = \sqrt{n}$.
45. Prove that if Q and P are orthogonal matrices, then
    (a) $\|QAP\|_F = \|A\|_F$
    (b) $\|QAP\|_2 = \|A\|_2$
46. Prove that the spectral norm of a symmetric matrix is the same as its spectral radius.
47. Let $A \in \mathbb{R}^{n\times n}$ and let x, y, and z be n-vectors such that $Ax = b$ and $Ay = b + z$. Then prove that
$$\frac{\|z\|_2}{\|A\|_2} \le \|x - y\|_2 \le \|A^{-1}\|_2 \, \|z\|_2$$
(assuming that A^{-1} exists).

48. Prove that $\|A\|_2 \le \|A\|_F$ (use the Cauchy-Schwarz inequality).
49. Prove that $\|A\|_2$ is just the largest singular value of A. How is $\|A^{-1}\|_2$ related to a singular value of A?
50. Prove that $\|A^T A\|_2 = \|A\|_2^2$.
51. Prove that
$$\|AB\|_F \le \|A\|_F \, \|B\|_F$$
$$\|AB\|_F \le \|A\|_2 \, \|B\|_F$$
52. Let $A = (a_1, \ldots, a_n)$, where a_j is the jth column of A. Then prove that
$$\|A\|_F^2 = \sum_{i=1}^{n} \|a_i\|_2^2$$
53. Prove that if A and $A + E$ are both nonsingular, then
$$\|(A + E)^{-1} - A^{-1}\| \le \|E\| \, \|A^{-1}\| \, \|(A + E)^{-1}\|$$
(*Banach lemma*). What is the implication of this result?
54. Let $A \in \mathbb{R}^{m\times n}$ have full rank. Prove that $A + E$ has also full rank if E is such that $\|E\|_2 < \frac{1}{\|A^\dagger\|_2}$, where $A^\dagger = (A^T A)^{-1} A^T$.
55. Show that the matrices
$$A = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 4 & 5 & 6 \end{pmatrix}, \qquad A = \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \end{pmatrix}$$
$$A = \begin{pmatrix} 1 & 2 & 3 \\ 0 & 5 & 4 \\ 0 & 0 & 1 \end{pmatrix}, \qquad A = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 2 & 3 \end{pmatrix}$$
are not convergent matrices.
56. Construct a simple example where the norm test for convergent matrices fails, but the matrix is still convergent.
57. Prove that the series $(I + A + A^2 + \cdots)$ converges if $\|B\| < 1$, where $B = PAP^{-1}$. What is the implication of this result? Construct a simple example to see the usefulness of the result in practical computations. (For details, see Wilkinson, *The Algebraic Eigenvalue Problem*, p. 60.)

Suggestions for Further Reading

The material covered in this chapter can be found in any standard book on linear algebra and matrix theory. In particular, we suggest the following books for further reading:

Franklin, Joel N. *Matrix Theory*. Englewood Cliffs, N.J.: Prentice Hall, 1968.
Hill, Richard O., Jr. *Elementary Linear Algebra with Applications*. Orlando, Fla.: Harcourt Brace Jovanovich, 1991.
Horn, Roger and Charles Johnson. *Matrix Analysis*. New York: Cambridge University Press, 1985.

Householder, A. S. *The Theory of Matrices in Numerical Analysis*. New York: Dover Publications, 1964.
Lancaster, Peter. *The Theory of Matrices*. New York: Academic Press, 1969.
Lancaster, Peter and M. Tismenetsky. *The Theory of Matrices with Applications*. 2d ed. Orlando, Fla.: Academic Press, 1985.
Lay, David C. *Linear Algebra and Its Applications*. Reading, Mass.: Addison-Wesley, 1994.
Leon, Steven J. *Linear Algebra with Applications*. New York: Macmillan, 1994.
Noble, Ben and James Daniel. *Applied Linear Algebra*, 3rd ed. Englewood Cliffs, NJ: Prentice Hall, 1988.
Schneider, Hans and George Philip Barker. *Matrices and Linear Algebra*. New York: Dover Publications, 1989.
Strang, Gilbert. *Introduction to Linear Algebra*. Wellesley, Mass.: Wellesley-Cambridge Press, 1993.
Strang, Gilbert. *Linear Algebra and Its Applications*. 3rd ed. San Diego: Harcourt Brace Jovanovich, 1988.
Wilkinson, James H. *The Algebraic Eigenvalue Problem*. Oxford: Clarendon Press, 1965, Chapter 1.

CHAPTER TWO

Floating-Point Numbers and Errors in Computations

2.1 Floating-Point Number Systems

Because of limited storage capacity, a real number may or may not be represented exactly on a computer. Thus, when using a computer, we have to deal with approximations of the real number system using finite computer representations. This chapter will be confined to the study of the arithmetic of such approximate numbers. In particular, we will examine the widely used and accepted (as by SUN® and IBM® workstations, personal computers, and so on) IEEE standard for binary floating-point arithmetic (IEEE 1985).

A nonzero **normalized** floating-point number x in base 2 has the form:

$$x = (-1)^s d_1.d_2d_3 \cdots d_t 2^e$$

or

$$x = \pm d_1.d_2 \cdots d_t 2^e$$

or

$$x = \pm r 2^e$$

where e is called the *exponent*, r is called the *significand*, t is the *precision*, $(-1)^s$ is the *sign*, and $(.d_2d_3 \cdots d_t)$ is the *fraction*. (Note that t is finite.)

$$d_1 = 1$$
$$d_i = 0 \quad \text{or} \quad 1, \qquad 2 \le i \le t$$

Three parameters specify all numerical values that can be represented. These are the precision, t, and the minimum and maximum exponents, L and U. The numbers L and U vary among computers, even those that adhere to the IEEE standard, since the standard recommends only minimums. As an example, the standard recommends, for single precision, that $t = 24, L = -126$, and $U = 127$. The recommendation for double precision is $t = 53, L = -1022$, and $U = 1023$.

Consider, for example, a 32-bit word:

1	8	23
s	e	f

Here s is the sign of the number, e is the field for the exponent, and f is the fraction.

Note that for normalized floating-point numbers in base 2, it is known that $d_1 = 1$, and thus d_1 can be stored implicitly.

The actual storage of the exponent is accomplished by storing the true exponent plus an offset, or bias. The bias is chosen so that e is always nonnegative. The IEEE standard also requires that the unbiased exponent have two unseen values of $L - 1$ and $U + 1$. $L - 1$ is used to encode ± 0 and denormalized numbers (i.e., those for which $d_1 \neq 1$). $U + 1$ is used to encode $\pm\infty$ and nonnumbers, such as $(+\infty) + (-\infty)$, which are denoted by NaN.

Note that for the single-precision example just given, the bias is 127. Thus, if the biased exponent is 255, then $\pm\infty$ (or NaN) is inferred. Likewise if the biased exponent is 0, then ± 0 (or denormalized number) is inferred. The standard specifies how to determine the different cases for the special situations. It is not important here to go into such detail. Curious readers should consult IEEE (1985).

From this discussion above, we see that the IEEE standard for single precision provides approximately seven decimal digits of accuracy, since $2^{-23} \cong 1.2 \times 10^{-7}$. Similarly, double precision provides approximately 16 decimal digits of accuracy ($2^{-52} \cong 2.2 \times 10^{-16}$).

There is also an IEEE standard for floating-point numbers that are not necessarily of base 2. By allowing a choice of base, β, we see that the set of all floating-point numbers, called the **floating-point number system**, is thus characterized by four parameters:

β	the number base
t	the precision
L, U	lower and upper limits of the exponent

We denote the set of normalized floating-point numbers of precision t by F_t. In F_t, $d_1 \neq 0$. The set of F_t is *not* closed under arithmetic operations; that is, the *sum, difference, product, or quotient of two floating-point numbers in F_t is not necessarily a number in F_t*. To see this, consider the simple example in the floating system with $\beta = 10, t = 3, L = -1, U = 2$:

$$a = 11.2 = 1.12 \times 10^1$$
$$b = 1.13 = 1.13 \times 10^0$$

The product $c = a \times b = 12.656 = 1.2656 \times 10^1$ is not in F_t. This example shows that during the course of a computation, a computed number may very well fall outside of F_t.

There are, of course, two ways a computed number can fall outside the range of F_t; first, the exponent of the number may fall outside the interval $[L, U]$, and second, the fractional part may contain more than $(t - 1)$ digits (this is exactly what happened in this example).

If the computations produce an exponent too large (too small) to fit in a given computer, then the situation is called **overflow (underflow)**.

Overflow is a serious problem; for most systems the result of an overflow is $\pm\infty$. Underflow is usually considerably less serious. The result of an underflow may be to set a value to zero, a denormalized number, or $\pm 2^L$.

Example 2.1.1 Overflow and Underflow

1. Let $\beta = 10, t = 3, L = -3, U = 3$.

$$a = 1.11 \times 10^3$$
$$b = 1.20 \times 10^3$$
$$c = a \times b = 1.33 \times 10^6$$

results in an *overflow*, because the exponent 6 is too large.

2. Let $\beta = 10, t = 3, L = -2, U = 3$

$$a = 1.0 \times 10^{-2}$$
$$b = 2.0 \times 10^{-2}$$
$$c = ab = 2 \times 10^{-4}$$

results in an *underflow*.

Simple mathematical computations such as finding a square root, finding a power of a number, or computing factorials can give overflow. For example, consider computing

$$c = \sqrt{a^2 + b^2}$$

If a or b is very large, then we will get overflow when we compute $a^2 + b^2$.

The IEEE standard also sets forth the results of operations with infinities and NaNs. All operations with infinities correspond to the limiting case in real analysis. Those ambiguous situations, such as $0 \cdot \infty$, result in NaNs, and all binary operations with one or two NaNs result in a NaN.

Overflow and underflow can sometimes be avoided just by organizing the computations differently. Consider, for example, the task of computing the length of an n-vector x with components $x_1, \ldots, x_n$

$$\|x\|_2^2 = x_1^2 + x_2^2 + \cdots + x_n^2$$

If some x_i is too big or too small, then we can get overflow or underflow with the usual way of computing $\|x\|_2$. However, if we normalize each component of the vector by dividing it by $m = \max(|x_1|, \ldots, |x_n|)$ and then form the squares and the sum, then overflow problems can be avoided. Thus, a better way to compute $\|x\|_2^2$ is as follows:

1. $m = \max(|x_1|, \ldots, |x_n|)$
2. $y_i = x_i/m, i = 1, \ldots, n$
3. $\|x\|_2 = m\sqrt{(y_1^2 + y_2^2 + \cdots + y_n^2)}$

2.2 Rounding Errors

If a computed result of a given real number is not machine representable, then there are two ways it can be represented in the machine. Consider

$$\pm 0.d_1 \ldots d_t d_{t+1} \ldots$$

The first method, **chopping**, is the method in which the digits after d_t are simply chopped off. The second method is **rounding**, in which the digits after d_t are not only chopped off, but the digit d_t is also rounded up or down, depending on whether $d_{t+1} \geq \beta/2$ or $d_{t+1} < \beta/2$.

Let $\mathrm{fl}(x)$ denote the floating-point representation of a real number x. Unless mentioned explicitly, all errors will be assumed to be rounding errors.

Example 2.2.1 Rounding Consider base 10. Let $x = 3.141596$.

$t = 2$	$t = 3$	$t = 4$
$\mathrm{fl}(x) = 3.1$	$\mathrm{fl}(x) = 3.14$	$\mathrm{fl}(x) = 3.142$

We now give an expression to measure the error made in representing a real number x on the computer and then show how this measure can be used to give bounds for errors in other floating point computations.

DEFINITION 2.2.1 Let $\hat{x}$ denote an approximation of x; then there are two ways we can measure the error:

$$\text{Absolute error} = |\hat{x} - x|$$

$$\text{Relative error} = \frac{|\hat{x} - x|}{|x|}, \qquad x \neq 0$$

Note that the relative error makes more sense than the absolute error. The following simple example shows this.

Example 2.2.2 Consider

$$x_1 = 1.31, \qquad \hat{x}_1 = 1.30$$

and

$$x_2 = 0.12, \qquad \hat{x}_2 = 0.11$$

The absolute errors in both cases are the same:

$$|\hat{x}_1 - x_1| = |\hat{x}_2 - x_2| = 0.01$$

On the other hand, the relative error in the first case is

$$\frac{|\hat{x}_1 - x_1|}{|x_1|} = 0.0076$$

and the relative error in the second case is

$$\frac{|\hat{x}_2 - x_2|}{|x_2|} = 0.0833$$

Thus, the relative errors show that $\hat{x}_1$ is closer to x_1 than $\hat{x}_2$ is to x_2, whereas the absolute errors give no indication of this at all.

The relative error gives an indication of the number of significant digits in an approximate answer. More specifically, we have the following.

DEFINITION 2.2.2 $\hat{x}$ is said to approximate x to s significant digits if s is the largest nonnegative integer for which the relative error $|x - \hat{x}|/|x| < 5(10^{-s})$.

Thus, in these examples, $\hat{x}_1$ and x_1 agree to *two* significant digits, whereas $\hat{x}_2$ and x_2 agree to about *one* significant digit.

We now give an expression for the relative error in representing a real number x by its floating-point representation $\mathrm{fl}(x)$.

THEOREM 2.2.1 Let $\mathrm{fl}(x)$ denote the floating-point representation of a real number x. Then

$$\frac{|\mathrm{fl}(x) - x|}{|x|} \leq \mu = \begin{cases} \frac{1}{2}\beta^{1-t} & \text{for rounding} \\ \beta^{1-t} & \text{for chopping} \end{cases} \tag{2.2.1}$$

Proof: We establish the bound for rounding and leave the other part as Exercise 1b.

Let x be written as

$$x = (0.d_1 d_2 \ldots d_t d_{t+1} \ldots) \times \beta^e$$

where $d_1 \neq 0$ and $0 \leq d_i < \beta$. When we round x, we obtain one of the following floating-point numbers:

$$x' = (0.d_1 d_2 \ldots d_t) \times \beta^e$$
$$x'' = [(0.d_1 d_2 \ldots d_t) + \beta^{-t})] \times \beta^e$$

Obviously we have $x \in (x', x'')$. Assume, without any loss of generality, that x is closer to x'. We then have

$$|x - x'| \leq \tfrac{1}{2}|x' - x''| = \tfrac{1}{2}\beta^{e-t}$$

Thus, the relative error is

$$\begin{aligned} \frac{|x - x'|}{|x|} &\leq \frac{1}{2}\frac{\beta^{-t}}{0.d_1 d_2 \ldots d_t \ldots} \\ &\leq \frac{1}{2}\frac{\beta^{-t}}{\frac{1}{\beta}} \qquad (\text{since } d_i < \beta) \\ &= \frac{1}{2}\beta^{1-t} \end{aligned}$$

Example 2.2.3 Consider the three-digit representation of the decimal number $x = 0.2346$ ($\beta = 10$, $t = 3$). Then, if rounding is used, we have:

$$\mathrm{fl}(x) = 0.235$$

$$\text{Relative error} \approx 0.001705 < \tfrac{1}{2}10^{-2}$$

If chopping is used, we have:

$$\text{fl}(x) = .234$$

$$\text{Relative error} = 0.0025575 < 10^{-2}$$

DEFINITION 2.2.3 The number μ in (2.2.1) is called the **machine precision**, or **unit roundoff error**. It is the smallest positive floating point number such that

$$\text{fl}(1 + \mu) > 1$$

μ is usually of order 10^{-16} and 10^{-6} (on most machines) for double and single precision, respectively. For the IBM 360 and 370, $\beta = 16$, $t = 6$, and $\mu = 4.77 \times 10^{-7}$.

The machine precision is very important in scientific computations. If the values of β, t, L, and U for a computer are not known, the following simple FORTRAN program can be run to estimate μ for that computer (Forsythe, Malcolm, and Moler 1977, 14.)

```
REAL MEU, MEU_1, SAVE
MEU  =1.0
MEU_1=2.0
DO WHILE (MEU_1 .GT. 1.0)
   SAVE=MEU
   MEU=0.5*MEU
   MEU_1=MEU+1.0
END DO
MEU=SAVE
PRINT *, 'MACHINE EPSILON IS ',MEU
```

This FORTRAN program computes an approximation of μ which differs from μ by at most a factor of 2. This approximation is quite acceptable, since an exact value of μ is not that important and is seldom needed.

The book by Forsythe, Malcolm, and Moler (1977) also contains an extensive list of L and U for various computers.

2.3 Laws of Floating-Point Arithmetic

The formula

$$\frac{|\text{fl}(x) - x|}{|x|} \le \mu = \begin{cases} \beta^{1-t} & \text{for chopping} \\ \frac{1}{2}\beta^{1-t} & \text{for rounding} \end{cases}$$

can be written as

$$\text{fl}(x) = x(1 + \delta) \qquad \text{where } |\delta| \le \mu$$

Assuming that the IEEE standard holds, we can easily derive the following simple *laws of floating-point arithmetic*.

THEOREM 2.3.1 Let x and y be two floating-point numbers, and let $\text{fl}(x + y)$, $\text{fl}(x - y)$, $\text{fl}(xy)$, and $\text{fl}(x/y)$ denote the computed sum, difference, product, and quotient,

respectively. Then

1. $\text{fl}(x \pm y) = (x \pm y)(1 + \delta)$, where $|\delta| \le \mu$.
2. $\text{fl}(xy) = (xy)(1 + \delta)$, where $|\delta| \le \mu$.
3. If $y \ne 0$, then $\text{fl}(x/y) = (x/y)(1 + \delta)$, where $|\delta| \le \mu$.

On computers that do not use the IEEE standard, the following floating-point law of addition might hold:

4. $\text{fl}(x + y) = x(1 + \delta_1) + y(1 + \delta_2)$ where $|\delta_1| \le \mu$, and $|\delta_2| \le \mu$.

Example 2.3.1 Simple Floating-Point Operations with Rounding

Let

$$\beta = 10, \qquad t = 3$$

in parts a through c.

a. $x = 0.999 \times 10^2$, $y = 0.111 \times 10^0$

$$x + y = 100.0110 = 0.100011 \times 10^3$$

$$\text{fl}(x + y) = 0.100 \times 10^3$$

Thus, $\text{fl}(x + y) = (x + y)(1 + \delta)$, where

$$\delta = -1.0999 \times 10^{-4}, \qquad |\delta| < \tfrac{1}{2}(10^{-2})$$

b. $x = 0.999 \times 10^2$, $y = 0.111 \times 10^0$

$$xy = 11.0889$$

$$\text{fl}(xy) = 0.111 \times 10^2$$

Thus, $\text{fl}(xy) = xy(1 + \delta)$, where

$$\delta = 1.00100 \times 10^{-3}, \qquad |\delta| \le \tfrac{1}{2}(10^{1-3})$$

c. $x = 0.999 \times 10^2$, $y = 0.111 \times 10^0$

$$\frac{x}{y} = 900$$

$$\text{fl}\left(\frac{x}{y}\right) = 0.900 \times 10^3$$

$$\delta = 0$$

d. Let

$$\beta = 10, \qquad t = 4$$

$$x = 0.1112$$

$$y = 0.2245 \times 10^5$$

$$xy = 0.249644 \times 10^4$$

$$\text{fl}(xy) = 0.2496 \times 10^4$$

Thus, $|\text{fl}(xy) - xy| = .44$ and

$$|\delta| = 1.7625 \times 10^{-4}$$

$$< \tfrac{1}{2} \times 10^{-3}$$

Computing Without a Guard Digit

Theorem 2.3.1 and Example 2.3.1 show that the relative errors in computing the sum, difference, product, and quotient in floating-point arithmetic are small. However, there are computers without guard digits in which additions and subtractions may not be accurate. We describe this situation in some detail.

A guard digit is an extra digit on the lower end of the arithmetic register whose purpose is to catch the low-order digit that would otherwise be pushed out of existence when the decimal points are aligned. The following example shows the difference between two models. Let

$$\beta = 10, \qquad t = 3, \qquad \mu = 0.005$$

Example 2.3.2 Addition with a Guard Digit

$$x = 0.101 \times 10^2, \qquad y = -0.994 \times 10^1$$

Step 1: Align the two numbers.

		Guard Digit
$x =$	0.101	$0 \quad \times 10^2$
$y =$	-0.099	$4 \quad \times 10^2$

Step 2: Add (with an extra digit).

$$\begin{array}{rl} 0.1010 & \times 10^2 \\ -0.0994 & \times 10^2 \\ \hline \mathrm{fl}(x+y) = 0.001\ 6 & \times 10^2 \end{array}$$

Step 3: Normalize.

$$\mathrm{fl}(x+y) = 1.60 \times 10^{-1}$$

Result: $\mathrm{fl}(x+y) = (x+y)(1+\delta)$ with $\delta = 0$

Example 2.3.3 Addition without a Guard Digit

$$x = 0.101 \times 10^2, \qquad y = -0.994 \times 10^1$$

Step 1: Align two numbers.

$$\begin{array}{rll} x = & 0.101 & \times 10^2 \\ y = & -0.099[4] & \times 10^2 \end{array}$$

The low-order digit [4] is pushed out.

Step 2: Add.

$$\begin{array}{rl} 0.101 & \times 10^2 \\ -0.099 & \times 10^2 \\ \hline \mathrm{fl}(x+y) = 0.002 & \times 10^2 \end{array}$$

Step 3: Normalize.

$$\mathrm{fl}(x+y) = 2.00 \times 10^{-1}$$

Result: $\mathrm{fl}(x+y) = (x+y)(1+\sigma)$ with $\sigma = 0.25 = 50\mu$.

Thus, we repeat that for computers with a guard digit,

$$\mathrm{fl}(x \pm y) = (x+y)(1+\delta) \qquad |\delta| \le \mu$$

However, for those without a guard digit

$$\mathrm{fl}(x \pm y) = x(1+\delta_1) \pm y(1+\delta_2)$$
$$|\delta_1| \le \mu, \qquad |\delta_2| \le \mu$$

A Final Remark: Throughout this book, we will assume that the computations have been performed with a guard digit, as they are on almost all available machines.

We shall call results 1 through 3 of Theorem 2.3.1 along with (2.2.1) of Theorem 2.2.1 the *fundamental laws of floating-point arithmetic*. These fundamental laws are the basis for establishing bounds for floating-point computations.

For example, consider the floating-point computation of $x(y+z)$:

$$\begin{aligned} \mathrm{fl}(x(y+z)) &= [x \cdot \mathrm{fl}(y+z)](1+\delta_1) \\ &= x(y+z)(1+\delta_2)(1+\delta_1) \\ &= x(y+z)(1+\delta_1\delta_2+\delta_1+\delta_2) \\ &\approx x(y+z)(1+\delta_3) \end{aligned}$$

where $\delta_3 = \delta_1 + \delta_2$; because δ_1 and δ_2 are small, their product is neglected.

We can now easily establish the bound of δ_3. Suppose $\beta = 10$ and that rounding is used. Then

$$\begin{aligned} |\delta_3| = |\delta_1 + \delta_2| &\le |\delta_1| + |\delta_2| \\ &\le \tfrac{1}{2} \times 10^{1-t} + \tfrac{1}{2} \times 10^{1-t} \\ &= 10^{1-t} \end{aligned}$$

Thus, *the relative error due to rounding in computing* $\mathrm{fl}(x(y+z))$ *is about* 10^{1-t} *in the worst case.*

2.4 Addition of *n* Floating-Point Numbers

Consider adding n floating-point numbers $x_1, x_2, \ldots, x_n$ with rounding. Define $s_2 = \mathrm{fl}(x_1 + x_2)$. Then

$$s_2 = \mathrm{fl}(x_1 + x_2) = (x_1 + x_2)(1+\delta_2)$$

where $|\delta_2| \le \frac{1}{2}\beta^{1-t} = \mu$. That is, $s_2 - (x_1 + x_2) = \delta_2(x_1 + x_2)$. Define $s_3, s_4, \ldots, s_n$ recursively by

$$s_{i+1} = \mathrm{fl}(s_i + x_{i+1}), \qquad i = 2, 3, \ldots, n-1$$

Then $s_3 = \mathrm{fl}(s_2 + x_3) = (s_2 + x_3)(1+\delta_3)$. That is,

$$\begin{aligned} s_3 - (x_1 + x_2 + x_3) &= (x_1 + x_2)\delta_2 + (x_1 + x_2)(1+\delta_2)\delta_3 + x_3\delta_3 \\ &\approx (x_1 + x_2)\delta_2 + (x_1 + x_2 + x_3)\delta_3 \end{aligned}$$

(neglecting the term $\delta_2\delta_3$, which is small, and so on). Thus, by induction we can show that

$$s_n - (x_1 + x_2 + \cdots + x_n) \approx (x_1 + x_2)\delta_2 + (x_1 + x_2 + x_3)\delta_3 + \cdots + (x_1 + x_2 + \cdots + x_n)\delta_n$$

(again neglecting the terms $\delta_i\delta_j$, which are small).

This result can be written as

$$s_n - (x_1 + x_2 + \cdots + x_n) \approx x_1(\delta_2 + \delta_3 + \cdots + \delta_n) + x_2(\delta_2 + \cdots + \delta_n) + x_3(\delta_3 + \cdots + \delta_n) + \cdots + x_n\delta_n$$

where each $|\delta_i| \le \frac{1}{2}\beta^{1-t} = \mu$. Defining $\delta_1 = 0$, we can write:

THEOREM 2.4.1 Let $x_1, x_2, \ldots, x_n$ be n floating-point numbers. Then

$$\mathrm{fl}(x_1 + x_2 + \cdots + x_n) - (x_1 + x_2 + \cdots + x_n) \approx x_1(\delta_1 + \delta_2 + \cdots + \delta_n) + x_2(\delta_2 + \cdots + \delta_n) + \cdots + x_n\delta_n$$

where each $|\delta_i| \le \mu$, $i = 1, 2, \ldots, n$.

Remark: From the preceding formula we see that we should expect smaller error in general when adding n floating-point numbers in ascending order of magnitude:

$$|x_1| \le |x_2| \le |x_3| \le \cdots \le |x_n|$$

If the numbers are arranged in ascending order of magnitude, then the larger errors will be associated with the smaller numbers.

2.5 Multiplication of *n* Floating-Point Numbers

Proceeding as in the case of addition of n floating-point numbers in the last section, it can be shown that the following theorem holds.

THEOREM 2.5.1

$$\mathrm{fl}(x_1 \times x_2 \times \cdots \times x_n) \approx (1 + \epsilon)\prod_{i=1}^{n} x_i$$

where $\epsilon = |(1 + \delta_2)(1 + \delta_3)\cdots(1 + \delta_n) - 1|$ and $|\delta_i| \le \mu$, $i = 1, 2, \ldots, n$.

A Bound for ϵ

Assuming that $(n - 1)\mu < 0.1$, we will prove that $\epsilon < 1.06(n - 1)\mu$. (This assumption is quite realistic; on most machines this assumption will hold for fairly large values of n).

Since $|\delta_i| \le \mu$, we have $\epsilon \le (1 + \mu)^{n-1} - 1$. Again, since

$$\ln(1 + \mu)^{n-1} = (n - 1)\ln(1 + \mu) < (n - 1)\mu$$

we have

$$(1 + \mu)^{n-1} < e^{(n-1)\mu}$$

Thus,

$$(1 + \mu)^{n-1} - 1 < e^{(n-1)\mu} - 1 = (n - 1)\mu + \frac{((n-1)\mu)^2}{2} + \cdots$$
$$= (n - 1)\mu\left[1 + \frac{(n-1)\mu}{2} + \frac{((n-1)\mu)^2}{6} + \cdots\right]$$
$$< (n - 1)\mu\left[1 + \frac{0.05}{1 - 0.05}\right]$$

(using the geometric series). (Note that $(n - 1)\mu < 0.1$.)

Thus,

$$\epsilon \le (1 + \mu)^{n-1} - 1 < (n - 1)\mu\left[1 + \frac{0.05}{1 - 0.05}\right] < 1.06(n - 1)\mu, \qquad (2.5.1)$$

Thus, combining Theorem 2.5.1 and (2.5.1), we can write the following.

THEOREM 2.5.2 The relative error in computing the product of n floating-point numbers is at most $1.06(n - 1)\mu$, assuming that $(n - 1)\mu < 0.1$.

2.6 Inner Product Computation

A computational task that often occurs in numerical linear algebra is the computation of the **inner product** of two n-vectors x and y:

$$x^T y = x_1y_1 + x_2y_2 + \cdots + x_ny_n \qquad (2.6.1)$$

where x_i and y_i, $i = 1, \ldots, n$, are the components of x and y.

Let x_i and y_i, $i = 1, \ldots, n$ be floating-point numbers. Define

$$S_1 = \mathrm{fl}(x_1y_1) \qquad (2.6.2)$$
$$S_2 = \mathrm{fl}(S_1 + \mathrm{fl}(x_2y_2)) \qquad (2.6.3)$$
$$\vdots$$
$$S_k = \mathrm{fl}(S_{k-1} + \mathrm{fl}(x_ky_k)), \qquad k = 3, 4, \ldots, n \qquad (2.6.4)$$

We then have, using Theorem 2.3.1,

$$S_1 = x_1y_1(1 + \delta_1) \qquad (2.6.5)$$
$$S_2 = [S_1 + x_2y_2(1 + \delta_2)](1 + \eta_2) \qquad (2.6.6)$$
$$\vdots$$
$$S_n = [S_{n-1} + x_ny_n(1 + \delta_n)](1 + \eta_n) \qquad (2.6.7)$$

where each $|\delta_i| \le \mu$ and $|\eta_i| \le \mu$. Substituting the values of S_1 through S_{n-1} in S_n and making some rearrangements, we can write

$$S_n = \sum_{i=1}^{n} x_i y_i (1 + \epsilon_i) \tag{2.6.8}$$

where

$$\begin{aligned} 1 + \epsilon_i &= (1 + \delta_i)(1 + \eta_i)(1 + \eta_{i+1}) \cdots (1 + \eta_n) \\ &\approx 1 + \delta_i + \eta_i + \eta_{i+1} + \cdots + \eta_n \qquad (\eta_1 = 0) \end{aligned} \tag{2.6.9}$$

(ignoring the products $\delta_i \eta_j$ and $\eta_j \eta_k$, which are small).

For example, when $n = 2$, it is easy to check that

$$S_2 = x_1 y_1 (1 + \epsilon_1) + x_2 y_2 (1 + \epsilon_2) \tag{2.6.10}$$

where $1 + \epsilon_1 \approx 1 + \delta_1 + \eta_2$, $1 + \epsilon_2 \approx 1 + \delta_2 + \eta_2$ (neglecting the products of $\delta_1 \eta_2$ and $\delta_2 \eta_2$, which are small).

As in the last section, it can be shown that if $n\mu < 0.1$, then

$$|\epsilon_i| \le 1.06(n + 2 - i)\mu, \qquad i = 1, 2, \ldots, n \tag{2.6.11}$$

From (2.6.8) and (2.6.11), we can write

$$\begin{aligned} |\,\mathrm{fl}(x^T y) - x^T y| &\le \sum_{i=1}^{n} |x_i|\,|y_i|\,|\epsilon_i| \\ &\le \phi(n)\mu |x|^T |y| \\ &\le \phi(n)\mu \,\|x\|_2 \,\|y\|_2 \qquad \text{(using the Cauchy-Schwarz inequality (Section 1.7))} \end{aligned}$$

where $|\alpha| = (|\alpha_1|, |\alpha_2|, \ldots, |\alpha_n|)^T$. $\phi(n)$ is a small function of n.

THEOREM 2.6.1

$$|\,\mathrm{fl}(x^T y) - x^T y| \le \phi(n)\mu |x|^T |y| \le \phi(n)\mu \,\|x\|_2 \,\|y\|_2$$

Computing the Inner Product in Double Precision

While talking about inner product computation, it is important to note that because most computers allow double-precision computations, it is recommended that the inner product be computed in double precision (using $2t$ digits arithmetic) to retain greater accuracy. The rationale here is that if we use single precision to compute $x^T y$, then there will be $(2n - 1)$ single-precision rounding errors (one for each multiplication and each addition). A better strategy is to convert each x_i and y_i to double precision by extending their mantissas with zeros, multiply them in double precision, add them in double precision, and give the final result in single precision. This process is known as *accumulation of inner product in double precision* (or extended precision). We summarize the process as follows:

1. Compute the individual products $x_i y_i$ in double precision.
2. Compute the sum $\sum_{i=1}^{n} x_i y_i$ in double precision in ascending order.
3. Round the sum in single precision.

The process gives low rounding error at little extra cost. It can be shown (Wilkinson, 1963, 117–118) that *the error in this case is essentially independent of* n. Specifically, it can be shown that if the inner product is accumulated in double precision and $\mathrm{fl}_2(x^T y)$ denotes the result of such computations, then the following holds.

THEOREM 2.6.2

$$|\,\mathrm{fl}_2(x^T y) - x^T y| \le c\mu |x^T y|$$

unless severe cancellation takes place in any of the terms of $x^T y$, where $n\mu < 0.1$ and c is a small constant.

Remark: The last sentence in Theorem 2.6.2 is important. We can construct a very simple example (Exercise 6(b)) to see that if cancellation takes place, the conclusion of Theorem 2.6.2 does not hold. *The phenomenon of catastrophic cancellation is discussed in the next section.*

2.7 Error Bounds for Floating-Point Matrix Computations

THEOREM 2.7.1 Let $|M| = (|m_{ij}|)$. Let A and B be two floating-point matrices and c be a floating-point number. Then

1. $\mathrm{fl}(cA) = cA + E$, $|E| \le \mu |cA|$
2. $\mathrm{fl}(A + B) = (A + B) + E$, $|E| \le \mu(|A + B|)$

If A and B are two matrices compatible for matrix multiplication, then from (2.6.4) we can show that

3. $\mathrm{fl}(AB) = AB + E$, $|E| \le n\mu |A|\,|B| + O(\mu^2)$

$|A| = \{|a_{ij}|\}$

„entrywise" errors (error in each element of matrix product)

Meaning of $O(\mu^2)$

In Theorem 2.7.1 the notation $O(\mu^2)$ stands for a complicated expression that is bounded by $c\mu^2$, where c is a constant, depending upon the problem. The expression $O(\mu^2)$ is used frequently in this book.

Remark: Result 3 of Theorem 2.7.1 shows that the matrix multiplication in floating-point arithmetic can be very inaccurate, since $|A|\,|B|$ may be much larger than $|AB|$ itself (Exercise 9). For this reason, whenever possible, while computing matrix-matrix or matrix-vector products, accumulation of inner products in double precision should be used, because in this case the entries of the error matrix can be shown to be bounded predominantly by the entries of the matrix $|AB|$ rather than those of $|A|\,|B|$; see Wilkinson (1965, 118).

Error Bounds in Terms of Norms

Traditionally, for matrix computations the bounds for error matrices are given in terms of the norms of the matrices rather than in terms of absolute values of the matrices, as given before. Here we rewrite the bound for error matrices for matrix multiplications using norms for easy reference later in the book. Note, however, that entrywise error bounds are more meaningful than normwise errors (see remarks in Section 3.2). Consider again the equation

$$\mathrm{fl}(AB) = AB + E, \quad |E| \le n\mu |A|\,|B| + O(\mu^2)$$

Because $\|E\| \le \| \, |E| \, \|$, we may rewrite the equation as

$$\mathrm{fl}(AB) = AB + E$$

where $\|E\| \le \| \, |E| \, \| \le n\mu \, \| \, |A| \, \| \, \| \, |B| \, \| + O(\mu^2)$
In particular, for $\| \ \|_2$ and $\| \ \|_\infty$ norms, we have

$$\|E\|_\infty \le n\mu \, \|A\|_\infty \, \|B\|_\infty + O(\mu^2)$$
$$\|E\|_2 \le n^2\mu \, \|A\|_2 \, \|B\|_2 + O(\mu^2)$$

THEOREM 2.7.2 $\mathrm{fl}(AB) = AB + E$, where $\|E\|_2 \le n^2\mu \, \|A\|_2 \, \|B\|_2 + O(\mu^2)$.

There are two special cases we must consider.

COROLLARY 2.7.1 Matrix–Vector Multiplication If b is a vector, then from the preceding, we have

$$\mathrm{fl}(Ab) = Ab + e$$

where $\|e\|_2 \le n^2\mu \, \|A\|_2 \, \|b\|_2 + O(\mu^2)$
(Also see Problem 11 and the remarks made there.)

COROLLARY 2.7.2 Matrix Multiplication by an Orthogonal Matrix Let $A \in R^{n\times n}$ and $Q \in R^{n\times n}$ be orthogonal. Then

$$\mathrm{fl}(QA) = Q(A + E)$$

where $\|E\|_2 \le O(\mu) \, \|A\|_2$.

> Corollary 2.7.2 says that, although matrix multiplication can be inaccurate in general, if one of the matrices is orthogonal, then the floating-point matrix multiplication gives only a small and acceptable error. As we will see in later chapters, this result forms the basis of many numerically viable algorithms discussed in this book.

For example, the following result, to be used very often in this book, is a consequence of the preceding result.

COROLLARY 2.7.3 Let P be an orthogonal matrix defined by

$$P = I - 2\frac{uu^T}{u^Tu}$$

where u is a column vector. Let $\hat{P}$ be the computed version of P in floating-point arithmetic. Then

$$\mathrm{fl}(\hat{P}A) = P(A + E)$$

where

$$\|E\|_2 \le O(\mu) \, \|A\|_2$$

and c is a constant of order unity.

Proof: See Wilkinson (1965, 152–160). ■

2.8 Roundoff Errors Due to Cancellation and Recursive Computations

Intuitively, it is clear that if a large number of floating-point computations is done, then the accumulated error can be quite large. However, roundoff error can be disastrous even in a single step of computation. For example, consider the subtraction of two numbers:

$$x = 0.54617$$
$$y = 0.54601$$

The exact value is

$$d = x - y = 0.00016$$

Suppose we now use four-digit arithmetic with rounding. Then we have

$$\hat{x} = 0.5462 \qquad \text{(correct to four significant digits)}$$
$$\hat{y} = 0.5460 \qquad \text{(correct to five significant digits)}$$
$$\hat{d} = \hat{x} - \hat{y} = 0.0002$$

How good is the approximation of $\hat{d}$ to d? The relative error is

$$\frac{|d - \hat{d}|}{|d|} = 0.25 \qquad \text{(quite large)}$$

What happened is this: In four-digit arithmetic, the numbers 0.5462 and 0.5460 are of almost the same size. So, when the first one was subtracted from the second, the most significant digits canceled and the very least significant digit was left in the answer. This phenomenon, known as **catastrophic cancellation**, occurs when two numbers of approximately the same size are subtracted. Fortunately, in many cases

catastrophic cancellation can be avoided. For example, consider the case of solving the quadratic equation:

$$ax^2 + bx + c = 0, \qquad a \neq 0$$

The usual way the two roots x_1 and x_2 are computed is:

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

$$x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

It is clear that if a, b, and c are numbers such that $-b$ is about the same size as $\sqrt{b^2 - 4ac}$ (with respect to the arithmetic used), then a catastrophic cancellation will occur in computing x_2; as a result, the computed value of x_2 can be completely erroneous.

Example 2.8.1 As an illustration, take $a = 1$, $b = -10^5$, and $c = 1$ (Forsythe, Malcolm, and Moler 1977, 20–22). Then, using $\beta = 10$, $t = 8$, and $L = -U = -50$, we see that

$$x_1 = \frac{10^5 + \sqrt{10^{10} - 4}}{2} = 10^5 \qquad \text{(good answer)}$$

$$x_2 = \frac{10^5 - 10^5}{2} = 0 \qquad \text{(completely wrong)}$$

The true x_2 is 0.000010000000001 (correctly rounded to 11 significant decimals). The catastrophic cancellation took place in computing x_2, since $-b$ and $(\sqrt{b^2 - 4ac})$ are of the same order. Note that in eight-digit arithmetic, $\sqrt{10^{10} - 4} = 10^5$.

Avoiding Cancellation

Cancellation can be avoided if an equivalent pair of formulas is used:

$$x_1 = \frac{-b - \text{sign}(b)\sqrt{b^2 - 4ac}}{2a}$$

$$x_2 = \frac{c}{ax_1}$$

where sign(b) is the sign of b. Using these formulas, we easily see that

$$x_1 = 100000.00$$

$$x_2 = \frac{1.0000000}{100000.00} = 0.00001000000$$

Example 2.8.2 For yet another example to see how cancellation can lead to inaccuracy, consider the problem of evaluating

$$f(x) = e^x - x - 1 \qquad \text{at } x = 0.01$$

Using five-digit arithmetic, the correct answer is 0.000050167. If $f(x)$ is evaluated directly from the expression, we have

$$f(0.01) = 1.0101 - (0.01) - 1 = 0.0001$$

$$\text{Relative error} = \frac{0.0001 - 0.000050167}{0.000050167} = 0.99 \times 10^0$$

indicating that we cannot trust even the first significant digit.

Fortunately, cancellation can again be avoided using the convergent series

$$e^x = 1 + x + \frac{x^2}{2} + \frac{x^3}{3!} + \cdots$$

In this case we have

$$e^x - x - 1 = \left(1 + x + \frac{x^2}{2} + \frac{x^3}{3!} + \cdots\right) - x - 1 = \frac{x^2}{2} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots$$

For $x = 0.01$, this formula gives

$$\frac{(0.01)^2}{2} + \frac{(0.01)^3}{3!} + \frac{(0.01)^4}{4!} + \cdots$$
$$= 0.00005 + 0.000000166666 + 0.00000000004166 + \cdots$$
$$= 0.000050167 \qquad \text{(correct up to five significant digits)}$$

Remark: Note that if x were negative, then use of the convergent series for e^x would not have helped. To compute e^x for a negative value of x, cancellation can be avoided by using

$$e^{-x} = \frac{1}{e^x} = \frac{1}{1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots}$$

Remark: Note that although cancellation leads to inaccuracies, it is not to be blamed for wrong results. The source may be somewhere else, perhaps in some previous computations. For example, in Example 2.8.1, the source was the computation of $\sqrt{10^{10} - 4}$ in eight-digit arithmetic.

Recursive Computations

In the preceding examples, we saw how subtractive cancellations can give inaccurate answers. There are, however, other common computations giving rise to errors; for example, consider recursive computations, which are computations performed recursively so that the computation of one step depends upon the results of previous steps. In such cases, even if the error made in the first step is negligible, due to the accumulation and magnification of error at every step, the final error can be quite large, giving a completely erroneous answer.

Certain recursions propagate errors in very unhealthy fashions. Consider a very nice example involving recursive computations, again from the book by Forsythe, Malcolm, and Moler (1977, 16–17).

Example 2.8.3 Suppose we need to compute the integral

$$E_n = \int_0^1 x^n e^{x-1}\,dx$$

for different values of n. Integrating by parts gives

$$E_n = \int_0^1 x^n e^{x-1}\,dx = (x^n e^{x-1})_0^1 - \int_0^1 n x^{n-1} e^{x-1}\,dx$$

or

$$E_n = 1 - nE_{n-1}, \qquad n = 2, 3, \ldots$$

Thus, if E_1 is known, then for different values of n, E_n can be computed using the preceding recursive formula.

Indeed, with $\beta = 10$ and $t = 6$, and starting with $E_1 = 0.367879$ as a six-digit approximation to $E_1 = 1/e$, we have

$$\begin{aligned} E_1 &= 0.367879 \\ E_2 &= 0.264242 \\ E_3 &= 0.207274 \\ E_4 &= 0.170904 \\ &\vdots \\ E_9 &= -0.0684800 \end{aligned}$$

Although the integrand is positive throughout the interval (0, 1), *the computed value of E_9 is negative.* This phenomenon can be explained as follows.

The error in computing E_2 was -2 times the error in computing E_1, the error in computing E_3 was -3 times the error in E_2 (therefore, the error at this step was 6 times the error in E_1). Thus, the error in computing E_9 was $(-2)(-3)(-4)\cdots(-9) = 9!$ times the error in E_1. The error in E_1 was due to the rounding of $1/e$ using six significant digits, which is about 4.412×10^{-7}. However, this small error multiplied by 9! gave $9! \times 4.412 \times 10^{-7} \approx 0.1601$, which is quite large.

Rearranging the Recurrence

Again, for Example 2.8.3 it turns out that we can get a much better result by simply rearranging the recursion so that the error at every step, instead of being magnified, is reduced. Indeed, if we rewrite the recursion as

$$E_{n-1} = \frac{1 - E_n}{n}, \qquad n = \ldots, 3, 2$$

then the error at each step will be reduced by a factor of $1/n$. Thus, starting with a large value of n (say, $n = 20$) and working backward, we see that E_9 will be accurate to full six-digit precision.

To obtain a starting value, we note that

$$E_n = \int_0^1 x^n e^{x-1}\,dx \le \int_0^1 x^n\,dx = \frac{1}{n+1}$$

With $n = 20$, $E_{20} \le \frac{1}{21}$. Let's take $E_{20} = 0$. Then, starting with $E_{20} = 0$, it can be shown (Forsythe, Malcolm, and Moler 1977, 17) that $E_9 = 0.0916123$, which is correct to full six-digit precision.

The reason for obtaining this accuracy was that the error in E_{20} was at most $\frac{1}{21}$; this error was multiplied by $\frac{1}{20}$ in computing E_{19}, giving an error of at most $\frac{1}{20} \cdot \frac{1}{21} = 0.0024$ in the computation of E_{19}, and so on.

2.9 Review and Summary

The concepts of floating-point numbers and rounding errors were introduced and discussed in this chapter.

1. *Floating-point numbers:* A normalized floating point number has the form

   $$x = \pm r\beta^e$$

   where e is called exponent, r is the significand, and β is the base of the number system. The floating-point number system is characterized by four parameters:

β	base
t	precision
L, U	lower and upper limits of the exponent

2. *Errors:* The errors in a computation are measured either by absolute error or relative error. The relative errors make more sense than absolute errors. The relative error gives an indication of the number of significant digits in an approximate answer. The relative error in representing a real number x by its floating-point representation $\mathrm{fl}(x)$ is bounded by a number μ, called the machine precision (Theorem 2.2.1).
3. *Laws of floating-point arithmetic:*

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

   where $*$ indicates any of the four basic arithmetic operations, $+$, $-$, $\times$, or $\div$, and $|\delta| \le \mu$.
4. *Addition, multiplication, and inner product computations:* The results of addition and multiplication of n floating point numbers are given in Theorems 2.4.1 and 2.5.1, respectively.
   (a) Add n floating-point numbers in ascending order of magnitude.
   (b) When computing the inner product of two vectors, accumulate the inner product in double precision whenever possible.
5. *Floating-point matrix multiplications:* The entrywise and normalized error bounds for matrix multiplication of two floating-point matrices are given in Theorems 2.7.1 and 2.7.2, respectively.

(a) Matrix multiplication in floating-point arithmetic can be very inaccurate, unless one of the matrices is orthogonal. Accumulation of the inner product is suggested, whenever possible, in computing a matrix-matrix or a matrix-vector product.

(b) The high accuracy in a matrix product computation involving an orthogonal matrix makes the use of orthogonal matrices in matrix computations quite attractive.

6. *Round-off errors due to cancellation and recursive computation:* Two major computations giving rise to roundoff errors are subtraction of two nearly equal numbers and recursive computations, which are discussed in some detail in Section 2.8. Examples were given to show how these errors come up in many basic computations. An encouraging message here is that in many instances, computations can be reorganized so that cancellation can be avoided, and the error in recursive computations can be diminished at each step of computation.

EXERCISES

1. (a) Show that

$$\frac{|\mathrm{fl}(x) - x|}{|x|} \le \mu = \begin{cases} \frac{1}{2}\beta^{1-t} & \text{for rounding} \\ \beta^{1-t} & \text{for chopping} \end{cases}$$

(b) Show that (a) can be written in the form

$$\mathrm{fl}(x) = x(1 + \delta), \qquad |\delta| \le \mu$$

2. Let x be a floating-point number and k be a positive integer. Then show

$$\mathrm{fl}\left(\frac{x^k}{k!}\right) = \frac{x^k}{k!}(1 + e_k)$$

where

$$|e_k| \le 2k\,\mu + O(\mu^2)$$

3. Construct an example to show that the distributive law for floating-point addition and multiplication does not hold. What can you say about the commutativity and associativity for these operations? Give reasons for your answers.
4. Let $x_1, x_2, \ldots, x_n$ be n floating-point numbers. Define

$$s_2 = \mathrm{fl}(x_1 + x_2), \qquad s_k = \mathrm{fl}(s_{k-1} + x_k), \qquad k = 3, \ldots, n$$

Then from Theorem 2.4.1 show that

$$s_n = \mathrm{fl}(x_1 + x_2 + \cdots + x_n) = x_1(1 + \nu_1) + x_2(1 + \nu_2) + \cdots + x_n(1 + \nu_n)$$

Give a bound for each ν_i, $i = 1, \ldots, n$.

5. (a) Construct an example to show that, when adding a list of floating-point numbers, the rounding error may be less if the numbers are added in order of increasing magnitude.

(b) Find another example to show that this is not always true.

6. (a) Prove that the error in computing an inner product with accumulation in double precision is essentially independent of n. That is, show that if $\mathrm{fl}_2(x^T y)$ denotes the computation of the inner product with accumulation in double precision, then unless severe cancellation takes place,

$$|\mathrm{fl}_2(x^T y) - x^T y| \le c\mu\,|x^T y| + O(\mu^2)$$

(Theorem 2.6.2).

(b) Show by means of a simple example that if there is cancellation, then $\mathrm{fl}_2(x^T y)$ can differ significantly from $x^T y$ (take $t = 3$).

(c) If s is a scalar, then prove that

$$\mathrm{fl}_2\left(\frac{x^T y}{s}\right) = \frac{x_1 y_1(1 + \delta_1) + \cdots + x_n y_n(1 + \delta_n)}{s/(1 + \delta)}$$

Find bounds for δ and δ_i, $i = 1, \ldots, n$.

7. Show that

(a) $\mathrm{fl}(cA) = cA + E$, $|E| \le \mu\,|cA|$

(b) $\mathrm{fl}(A + B) = (A + B) + E$, $|E| \le \mu(|A| + |B|)$

(c) $\mathrm{fl}(AB) = AB + E$, $|E| \le n\mu\,|A|\,|B| + O(\mu^2)$

8. Construct a simple example to show that the matrix multiplication in floating-point arithmetic need not be accurate. Rework your example using accumulation of inner product in double precision.
9. Let A and B be $n \times n$ matrices; then show that

$$\mathrm{fl}(AB) = AB + E$$

where

$$|e_{ij}| \le n^2\,|f_{ij}|\,\mu + 0(\mu^2)$$

f_{ij} = inner product of the ith row of A and jth column of B.

10. Prove that if Q is orthogonal, then

$$\mathrm{fl}(QA) = Q(A + E), \qquad \text{where } \|E\|_2 \le n^2\mu\,\|A\|_2 \mid O(\mu^2)$$

11. Let b be a column vector and $x = Ab$. Let $\hat{x} = \mathrm{fl}(x)$. Then show that

$$\frac{\|\hat{x} - x\|}{\|x\|} \le p(n)\mu\,\|A^{-1}\|\,\|A\|$$

where $p(n)$ is a polynomial in n of low degree. (The number $\|A^{-1}\|\,\|A\|$ is called the **condition number** of A. There are matrices for which this number can be very big. For those matrices, we then conclude that the *relative error in a matrix-vector product can be quite large.*)

12. Using Theorem 2.7.1, prove that, if B is nonsingular,

$$\frac{\|\mathrm{fl}(AB) - AB\|_F}{\|AB\|_F} \le n\mu\,\|B\|_F\,\left\|B^{-1}\right\|_F + O(\mu^2)$$

13. Let $y_1, \ldots, y_n$ be n column vectors defined recursively:

$$y_{i+1} = Ay_i, \qquad i = 1, 2, \ldots, n-1$$

Let $\hat{y}_i = \mathrm{fl}(y_i)$. Find a bound for the relative error in computing each y_i, $i = 1, \ldots, n$.

14. Let $\beta = 10, t = 4$. Compute

$$\mathrm{fl}(A^T A)$$

where

$$A = \begin{pmatrix} 1 & 1 \\ 10^{-4} & 0 \\ 0 & 10^{-4} \end{pmatrix}$$

Repeat your computation with $t = 9$. Compare the results.

15. Show how to arrange computations in each of the following so that the loss of significant digits can be avoided. Do one numerical example in each case to support your answer.
    (a) $e^x - x - 1$, for negative values of x
    (b) $\sqrt{x^4 + 1} - x^2$, for large values of x
    (c) $\dfrac{1}{x} - \dfrac{1}{x+1}$, for large values of x
    (d) $x - \sin x$, for values of x near zero
    (e) $1 - \cos x$, for values of x near zero

16. What are the relative and absolute errors in approximating each of the following?
    (a) π by $\dfrac{22}{7}$
    (b) $\dfrac{1}{3}$ by 0.333
    (c) $\dfrac{1}{6}$ by 0.166

    How many significant digits are there in each approximation?

17. Let $\beta = 10$ and $t = 4$. Consider computing

$$a = \frac{(\frac{1}{6} - 0.1666)}{0.1666}$$

How many correct significant digits of the exact answer will you get?

18. Consider evaluating

$$e = \sqrt{a^2 + b^2}$$

How can the computation be organized so that overflow in computing $a^2 + b^2$ for large values of a or b can be avoided?

19. What answers will you get if you compute the following numbers on your calculator or computer?

    (a) $\sqrt{10^8 - 1}$
    (b) $\sqrt{10^{-20} + 1}$
    (c) $10^{16} - 50$

    Compute the absolute and relative errors in each case.

20. What problem do you foresee in solving the quadratic equations
    (a) $x^2 - 10^6 x + 1 = 0$
    (b) $10^{-10} x^2 - 10^{10} x + 10^{10} = 0$

    using the well-known formula

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

What remedy do you suggest? Now solve the equations with your suggested remedy.

21. Show that the integral

$$y_i = \int_0^1 \frac{x^i}{x+5}\, dx$$

can be computed by using the recursion formula

$$y_i = \frac{1}{i} - 5y_{i-1}$$

Compute $y_1, y_2, \ldots, y_{10}$ using this formula, taking

$$y_0 = \ln(x+5)\big|_{x=0}^{1} = \ln 6 - \ln 5 = \ln(1.2)$$

What abnormalities do you observe in this computation? Explain what happened. Now rearrange the recursion so that the values of y_i can be computed more accurately.

22. Suppose x^* approximates x to five significant figures. If $x = 10^4$, 50,000, or 55,596, find an interval in each case containing x^*.

Suggestions for Further Reading

For details of the IEEE standard, see the monograph "An American National Standard: IEEE Standard for Binary Floating-Point Arithmetic" (Piscataway, N.J.: IEEE, 1985) and Goldberg (1991).

For results on error bounds for basic floating-point matrix operations, the book by James H. Wilkinson, *Rounding Errors in Algebraic Processes* (Englewood Cliffs, N.J.: Prentice-Hall, 1963), is an extremely useful and valuable resource.

Discussion on basic floating-point operations and rounding errors due to cancellations and recursive computations are given nowadays in many elementary numerical analysis textbooks. We shall name a few here which we have used and found useful.

Atkinson, Kendall. *Elementary Numerical Analysis*. New York: John Wiley, 1993.
Cheney, Ward, and David Kincaid. *Numerical Mathematics and Computing*, 3d ed., Pacific Grove, Calif.: Brooks/Cole, 1994.

Conte, S. D., and C. de Boor. *Elementary Numerical Analysis: An Algorithmic Approach*, 3d ed. New York: McGraw-Hill, 1980.
Dahlquist, G., and A. Björck. *Numerical Methods*. Englewood Cliffs, N.J.: Prentice-Hall, 1974.
Faires, J. D., and R. L. Burden. *Numerical Methods*. Boston: PWS-Kent, 1993.
Forsythe, George E., Michael A. Malcolm, and Cleve B. Moler. *Computer Methods for Mathematical Computations*. Englewood Cliffs, N.J.: Prentice-Hall, 1977.
Golub, Gene H., and James M. Ortega. *Scientific Computing and Differential Equations: An Introduction to Numerical Methods*. San Diego: Academic Press, 1992.
Johnston, R. L. *Numerical Methods: A Software Approach*. Toronto: John Wiley, 1982.
Kahaner, D., C. B. Moler, and S. Nash. *Numerical Methods and Software*. Englewood Cliffs, N.J.: Prentice-Hall, 1988.
Ortega, James M., and William G. Poole, Jr. *Numerical Methods for Differential Equations*. Mass.: Pitman Publishing, 1981.
Rice, John R. *Numerical Methods, Software and Analysis*. New York: McGraw-Hill, 1983.
Skeel, Robert D., and J. B. Keiper. *Elementary Numerical Computing with Mathematica*. New York: McGraw-Hill, 1993.

CHAPTER THREE

Stability of Algorithms and Conditioning of Problems

3.1 Some Basic Algorithms

We begin with a definition.

DEFINITION 3.1.1 An **algorithm** is an ordered set of operations, logical and arithmetic, which—when applied to a computational problem defined by a given set of data, called the **input data**—produces a solution to the problem. A solution comprises of a set of data called the **output data**.

In this book, for the sake of convenience and simplicity, we will very often describe algorithms by means of **pseudocodes**, which can easily be translated into computer codes. Describing algorithms by pseudocodes has been popularized by Stewart (1973).

3.1.1 Computing the Norm of a Vector

Given $x = (x_1, \ldots, x_n)^T$, compute $\|x\|_2$.

ALGORITHM 3.1.1 Computing the Norm of a Vector

Input Data: $n, x_1, \ldots, x_n$

Step 1: Compute $r = \max(|x_1|, \ldots, |x_n|)$

Step 2: Compute $y_i = x_i/r$, $i = 1, \ldots, n$

Step 3: Compute $s = \|x\|_2 = r\sqrt{(y_1^2 + \cdots + y_n^2)}$

Output Data: s

Pseudocodes

$r = \max(|x_1|, \ldots, |x_n|)$
$s = 0$

For $i = 1$ to n do
$y_i = x_i/r$
$s = s + y_i^2$
$s = r(s)^{1/2}$

An Algorithmic Note: In order to avoid overflow, each entry of x has been normalized before using the formula

$$\|x\|_2 = \sqrt{x_1^2 + \cdots + x_n^2}$$

3.1.2 Computing the Inner Product of Two Vectors

Given x and y two n-vectors, compute the inner product

$$x^T y = x_1 y_1 + x_2 y_2 + \cdots + x_n y_n$$

ALGORITHM 3.1.2 Computing the Inner Product of Two Vectors

Input Data: $n; x_1, x_2, \ldots, x_n; y_1, y_2, \ldots, y_n$

Step 1: Compute the partial products: $s_i = x_i y_i$, $i = 1, \ldots, n$

Step 2: Add the partial products: sum $= \sum_{i=1}^{n} s_i$

Output Data: Sum

Pseudocodes

Sum $= 0$
For $i = 1, \ldots, n$ do
Sum = Sum + $x_i y_i$

3.1.3 Solution of an Upper Triangular System

Consider the system

$$Ty = b$$

where $T = (t_{ij})$ is a nonsingular upper triangular matrix and $y = (y_1, y_2, \ldots, y_n)^T$. Specifically,

$$\begin{aligned}
t_{11}y_1 + t_{12}y_2 + \cdots + t_{1n}y_n &= b_1 \\
t_{22}y_2 + \cdots + t_{2n}y_n &= b_2 \\
t_{33}y_3 + \cdots + t_{3n}y_n &= b_3 \\
&\vdots \\
t_{n-1,n-1}y_{n-1} + t_{n-1,n}y_n &= b_{n-1} \\
t_{nn}y_n &= b_n
\end{aligned}$$

where each $t_{ii} \neq 0$ for $i = 1, 2, \ldots, n$.

The last equation is solved first to obtain y_n; then this value is inserted in the next-to-the-last equation to obtain y_{n-1}, and so on. This process is known as **back substitution**. The algorithm can easily be written down.

ALGORITHM 3.1.3 Back Substitution

Input Data: $T = (t_{ij})$, an $n \times n$ nonsingular upper triangular matrix, and b an n-vector

Step 1: Compute $y_n = \dfrac{b_n}{t_{nn}}$

Step 2: Compute y_{n-1} through y_1 successively:

$$y_i = \frac{1}{t_{ii}}\left(b_i - \sum_{j=i+1}^{n} t_{ij}y_j\right), \qquad i = n-1, \ldots, 2, 1$$

Output Data: $y = (y_1, \ldots, y_n)^T$

Pseudocodes

For $i = n, n-1, \ldots, 3, 2, 1$ do

$$y_i = \frac{1}{t_{ii}}\left(b_i - \sum_{j=i+1}^{n} t_{ij}y_j\right)$$

Note: When $i = n$, the summation $(\sum)$ is skipped.

MATCOM Note: The algorithm has been implemented in MATCOM program BACKSUB. See Chapter 4 for the description of MATCOM.

3.1.4 Computing the Inverse of an Upper Triangular Matrix

Finding the inverse of an $n \times n$ matrix A is equivalent to finding a matrix X such that

$$AX = I$$

Let $X = (x_1, \ldots, x_n)$ and $I = (e_1, \ldots, e_n)$, where x_i is the ith column of X and e_i is the ith column of I. Then the matrix equation $AX = I$ amounts to solving n linear systems:

$$Ax_i = e_i, \qquad i = 1, \ldots, n$$

The job is then particularly simple when A is a triangular matrix. Let T be an upper triangular matrix. Then finding its inverse $S = (s_1, \ldots, s_n)$ amounts to solving n upper triangular linear systems:

$$Tx_i = e_i, \qquad i = 1, \ldots, n$$

Let $s_i = (s_{1i}, s_{2i}, \ldots, s_{ni})^T$.
For $i = 1$: $Ts_1 = e_1$ gives

$$s_{11} = \frac{1}{t_{11}}$$

(The entries s_{21} through s_{n1} are all zero.)

For $i = 2$: $Ts_2 = e_2$ gives

$$s_{22} = \frac{1}{t_{22}}, \quad s_{12} = -\frac{1}{t_{11}}(t_{12}s_{22})$$

(The entries s_{32} through s_{n2} are all zero.)
For $i = k$: $Ts_k = e_k$ gives

$$s_{kk} = \frac{1}{t_{kk}}, \quad s_{ik} = -\frac{1}{t_{ii}}(t_{i,i+1}s_{i+1,k} + \cdots + t_{ik}s_{kk}), \quad i = k-1, k-2, \ldots, 1$$

(The other entries of the column s_k are all zero.) The pseudocodes of the algorithm can now be easily written down.

> **A Convention**
> From now on, we shall use the following format for algorithm descriptions.

ALGORITHM 3.1.4 The Inverse of an Upper Triangular Matrix Let T be an $n \times n$ nonsingular upper triangular matrix. The following algorithm computes S, the inverse of T.

For $k = n, n-1, \ldots, 1$ do

$$1: \quad s_{kk} = \frac{1}{t_{kk}}$$

$$2: \quad s_{ik} = -t_{ii}^{-1} \sum_{j=i+1}^{k} t_{ij}s_{jk} \; (i = k-1, k-2, \ldots 1)$$

Example 3.1.1 Compute S given

$$T = \begin{pmatrix} 5 & 2 & 3 \\ 0 & 2 & 1 \\ 0 & 0 & 4 \end{pmatrix}$$

$k = 3$:

$$s_{33} = \frac{1}{4}$$

$$s_{23} = -\frac{1}{t_{22}}(t_{23}s_{33}) = -\frac{1}{2}\left(1 \cdot \frac{1}{4}\right) = -\frac{1}{8}$$

$$s_{13} = -\frac{1}{t_{11}}(t_{12}s_{23} + t_{13}s_{33}) = -\frac{1}{5}\left(2 \cdot (-\frac{1}{8}) + 3 \cdot \frac{1}{4}\right) = -\frac{1}{10}$$

$k = 2$:

$$s_{22} = \frac{1}{t_{22}} = \frac{1}{2}$$

$$s_{12} = -\frac{1}{t_{11}}(t_{12}s_{22}) = -\frac{1}{5}\left(2 \cdot \frac{1}{2}\right) = -\frac{1}{5}$$

$k = 1$:

$$s_{11} = \frac{1}{t_{11}} = \frac{1}{5}$$

$$T^{-1} = S = \begin{pmatrix} \frac{1}{5} & -\frac{1}{5} & -\frac{1}{10} \\ 0 & \frac{1}{2} & -\frac{1}{8} \\ 0 & 0 & \frac{1}{4} \end{pmatrix}$$

3.1.5 Gaussian Elimination for Solving $Ax = b$

Consider the problem of solving the linear system of n equations in n unknowns:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n \end{aligned}$$

or, in matrix notation,

$$Ax = b$$

where $A = (a_{ij})$ and $b = (b_1, \ldots, b_n)^T$.

A well-known approach to solving the problem is the classical elimination scheme known as **Gaussian elimination**. A detailed description and mechanism of development of this historical algorithm and its important practical variations is given in Chapters 5 and 6. However, for a better understanding of some of the material presented in this chapter, we just give a brief description of the basic Gaussian elimination scheme.

The basic idea is to reduce the system to an equivalent upper triangular system so that the reduced upper triangular system can be solved easily using the back-substitution algorithm (Algorithm 3.1.3).

The **reduction process** consists of $n-1$ steps. In the following,

$$b^{(k)} = \begin{pmatrix} b_1^{(k)} \\ b_2^{(k)} \\ \vdots \\ b_n^{(k)} \end{pmatrix}, \quad A^{(k)} = \left(a_{ij}^{(k)}\right),$$

$$A^{(0)} = A, \; b^{(0)} = b.$$

Step 1: At step 1, the unknown x_1 is eliminated from the second through the nth equations. This is done by multiplying the first equation by

$$\left(-\frac{a_{21}}{a_{11}}\right), \left(-\frac{a_{31}}{a_{11}}\right), \ldots, \left(-\frac{a_{n1}}{a_{11}}\right)$$

and adding it to the second through nth equations. The quantities

$$m_{i1} = -\frac{a_{i1}}{a_{11}}, \qquad i = 2, \ldots, n$$

are called **multipliers**. At the end of step 1, the system $Ax = b$ becomes $A^{(1)}x = b^{(1)}$, where the entries of $A^{(1)} = (a_{ij}^{(1)})$ and those of $b^{(1)}$ are related to the entries of A and b as follows:

$$a_{ij}^{(1)} = a_{ij} + m_{i1}a_{1j} \qquad (i = 2, \ldots, n; j = 2, \ldots, n)$$

$$b_i^{(1)} = b_i + m_{i1}b_1 \qquad (i = 2, \ldots, n)$$

(*Note that* $a_{21}^{(1)}, a_{31}^{(1)}, \ldots, a_{n1}^{(1)}$ *are all zero.*)

Step 2: At step 2, x_2 is eliminated from the third through the nth equations of $A^{(1)}x = b^{(1)}$ by multiplying the second equations of $A^{(1)}x = b^{(1)}$ by the multipliers

$$m_{i2} = -\frac{a_{i2}^{(1)}}{a_{22}^{(1)}}, \qquad i = 3, \ldots, n$$

and adding the result to the third through nth equations. The system now becomes $A^{(2)}x = b^{(2)}$, whose entries are given as follows:

$$a_{ij}^{(2)} = a_{ij}^{(1)} + m_{i2}a_{2j}^{(1)} \qquad (i = 3, \ldots, n; \quad j = 3, \ldots, n)$$

$$b_i^{(2)} = b_i^{(1)} + m_{i2}b_2^{(1)} \qquad (i = 3, \ldots, n)$$

$$a_{i2}^{(2)} = 0 \qquad (i = 3, \ldots, n)$$

The other entries of $A^{(2)}$ and those of $b^{(2)}$ are the same as those of $A^{(1)}$ and $b^{(1)}$ respectively.

Step k: At step k, the $(n - k)$ multipliers

$$m_{ik} = -\frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}}, \qquad i = k + 1, \ldots, n$$

are formed; using them, x_k is eliminated from the $(k + 1)$st through the nth equations of $A^{(k-1)}x = b^{(k-1)}$. The entries of $A^{(k)}$ and those of $b^{(k)}$ are given by

$$a_{ij}^{(k)} = a_{ij}^{(k-1)} + m_{ik}a_{kj}^{(k-1)} \qquad (i = k + 1, \ldots, n; \; j = k + 1, \ldots, n)$$

$$b_i^{(k)} = b_i^{(k-1)} + m_{ik}b_k^{(k-1)} \qquad (i = k + 1, \ldots, n)$$

$$a_{ik}^{(k)} = 0 \qquad (i = k + 1, \ldots, n)$$

The other entries of $A^{(k)}$ and $b^{(k)}$ are the same as those of $A^{(k-1)}$ and $b^{(k-1)}$ respectively.

Step n − 1: At the end of the $(n - 1)$st step, the reduced matrix $A^{(n-1)}$ is upper triangular and the original vector b is transformed to $b^{(n-1)}$.

We are now ready to write down the pseudocodes of the Gaussian elimination scheme.

> The following observations will help write the pseudocodes:
>
> 1. There are $n - 1$ steps:
>
> $$k = 1, 2, \ldots, n - 1$$
>
> 2. For each value of k, there are $n - k$ multipliers: m_{ik}, $(i = k + 1, \ldots, n)$.
> 3. For each value of k, only $(n - k)^2$ entries of $A^{(k)} = (a_{ij}^{k})$ are modified $(i = k + 1, \ldots, n; j = k + 1, \ldots, n)$. The $n - k$ entries below the (k, k)th entry of the kth column are zeros, and the remaining other entries that are not modified remain the same as those of the corresponding entries of $A^{(k-1)}$.

ALGORITHM 3.1.5 Basic Gaussian Elimination

For $k = 1, 2, \ldots, n - 1$ do
  For $i = k + 1, \ldots, n$ do

$$m_{i,k} = -\frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}} \qquad \text{(assuming that } a_{kk}^{k-1} \neq 0\text{)}.$$

$$b_i^{(k)} = b_i^{(k-1)} + m_{ik}b_k^{(k-1)}$$

For $j = k + 1, \ldots, n$ do

$$a_{ij}^{(k)} = a_{ij}^{(k-1)} + m_{ik}a_{kj}^{(k-1)}$$

(*Note that* $A = (a_{ij}) = (a_{ij}^{(0)})$, $b = b^{(0)}$.)

Remarks:

1. The basic Gaussian elimination algorithm is commonly known as the *Gaussian elimination algorithm without row interchanges* or the *Gaussian elimination algorithm without pivoting*. The reason for having such a name will be clear from the discussion of this algorithm in Chapter 5.
2. The basic Gaussian elimination algorithm as presented here is *not* commonly used in practice. Two practical variations of this algorithm, known as Gaussian elimination with partial and complete pivoting, are described in Chapters 5 and 6.
3. *We have assumed that the quantities* $a_{11}, a_{22}^{(1)}, \ldots, a_{nn}^{(n-1)}$ *are different from zero. If any of them is zero, the algorithm will stop.*

Example 3.1.2 Solve

$$5x_1 + x_2 + x_3 = 7$$
$$x_1 + x_2 + x_3 = 3$$
$$2x_1 + x_2 + 3x_3 = 6$$

or

$$\begin{pmatrix} 5 & 1 & 1 \\ 1 & 1 & 1 \\ 2 & 1 & 3 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 7 \\ 3 \\ 6 \end{pmatrix}$$

$$Ax = b$$

Step 1: $k = 1$

$i = 2, 3$

$$m_{21} = -\frac{a_{21}}{a_{11}} = -\frac{1}{5}$$

$$b_2^{(1)} = b_2 + m_{21}b_1 = \frac{8}{5}$$

$j = 2, 3$

$$i = 2, j = 2: \quad a_{22}^{(1)} = a_{22} + m_{21}a_{12} = \frac{4}{5}$$

$$i = 2, j = 3: \quad a_{23}^{(1)} = a_{23} + m_{21}a_{13} = \frac{4}{5}$$

$$i = 3, j = 2: \quad a_{32}^{(1)} = a_{32} + m_{31}a_{12} = \frac{3}{5}$$

$$m_{31} = -\frac{a_{31}}{a_{11}} = -\frac{2}{5},$$

$$i = 3, j = 3: \quad a_{33}^{(1)} = a_{33} + m_{31}a_{13} = \frac{13}{5}$$

$$b_3^{(1)} = b_3 + m_{31}b_2 = \frac{16}{5}$$

(*Note:* $b_1^{(1)} = b_1$, $a_{21}^{(1)} = a_{31}^{(1)} = 0$, $a_{11}^{(1)} = a_{11}$, $a_{12}^{(1)} = a_{12}$, $a_{13}^{(1)} = a_{13}$.)

$$\begin{pmatrix} 5 & 1 & 1 \\ 0 & \frac{4}{5} & \frac{4}{5} \\ 0 & \frac{3}{5} & \frac{13}{5} \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 7 \\ \frac{8}{5} \\ \frac{16}{5} \end{pmatrix}$$

$$A^{(1)}x = b^{(1)}$$

Step 2: $k = 2$

$i = 3$

$$m_{32} = -\frac{a_{32}^{(1)}}{a_{22}^{(1)}} = -\frac{3}{4}$$

$$i = 3, j = 3: \quad a_{33}^{(2)} = a_{33}^{(1)} + m_{32}a_{23}^{(1)} = 2$$

$$b_3^{(2)} = b_3^{(1)} + m_{32}b_2^{(1)} = 2$$

$$\begin{pmatrix} 5 & 1 & 1 \\ 0 & \frac{4}{5} & \frac{4}{5} \\ 0 & 0 & 2 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 7 \\ \frac{8}{5} \\ 2 \end{pmatrix}$$

$$A^{(2)}x = b^{(2)}$$

Note that $A^{(2)}$ is upper triangular.

Back substitution: This triangular system is easily solved using back substitution:

$$2x_3 = 2 \Rightarrow x_3 = 1$$

$$\frac{4}{5}x_2 + \frac{4}{5}x_3 = \frac{8}{5} \Rightarrow x_2 = 1$$

$$5x_1 + x_2 + x_3 = 7 \Rightarrow x_1 = 1$$

3.2 Definitions and Concepts of Stability

The examples on catastrophic cancellations and recursive computations in the last chapter had one thing in common: *The inaccuracy of the computed result in each case was due entirely to the algorithm used*, because as soon as the algorithm was changed or rearranged and applied to the problem with the same data, the computed result became very satisfactory. Thus, we are talking about two different types of algorithms for a given problem. The algorithms of the first type—giving inaccurate results—are examples of **unstable algorithms**, whereas the ones of the second type—giving satisfactory results—are **stable algorithms.**

The study of stability is very important. This is done by means of **roundoff error analysis.** There are two types: **backward error analysis** and **forward error analysis.**

In forward analysis an attempt is made to see how the computed solution obtained by the algorithm differs from the exact solution based on the same data.

DEFINITION 3.2.1 An algorithm will be called **forward stable** if the computed solution $\hat{x}$ is close to the exact solution, x, in some sense.

The roundoff error bounds obtained in Chapter 2 for various arithmetic operations are the result of forward error analyses. Thus, from Theorem 2.3.1 we immediately conclude that floating-point arithmetic operations (addition, subtraction, multiplication, and division) are forward stable.

On the other hand, backward analysis relates the error to the data of the problem rather than to the problem's solution. Thus we define backward stability as follows:

DEFINITION 3.2.2 An algorithm is called **backward stable** if it produces an exact solution to a nearby problem.

Backward error analysis, popularized in the literature by J. H. Wilkinson (1919–1986), is now widely used in matrix computations; using this analysis, the stability (or instability) of many algorithms in numerical linear algebra has been established in recent years. **In this book, by "stability" we will imply "backward stability," unless otherwise stated.**

As a simple example of backward stability, consider again the case of computing the sum of two floating-point numbers x and y. We have seen before that

$$\begin{aligned} \mathrm{fl}(x+y) &= (x+y)(1+\delta) \\ &= x(1+\delta) + y(1+\delta) \\ &= x' + y' \end{aligned}$$

Thus, the computed sum of two floating-point numbers x and y is the exact sum of another two floating-point numbers x' and y'. Because

$$|\delta| \le \mu$$

both x' and y' are close to x and y, respectively. Thus we conclude that *the operation of adding two floating-point numbers is backward stable*. Similar statements, of course, hold for other floating-point arithmetic operations. For example, the addition of n floating-point numbers and computing the inner product of two floating-point vectors are backward stable.

For yet another type of example, consider the problem for solving the linear system $Ax = b$.

DEFINITION 3.2.3 An algorithm for solving $Ax = b$ will be called **stable** if the computed solution $\hat{x}$ is such that

$$(A + E)\hat{x} = b + \delta b$$

with E and δb small.

Measuring Smallness

The "smallness" of a matrix or a vector is measured either by looking into its entries or by computing its norm.

> **Normwise versus Entrywise Errors**
>
> Although measuring errors in computations using norms is traditional in matrix computations, componentwise measure of errors is becoming increasingly important. It really does make more sense.
>
> An $n \times n$ matrix A has n^2 entries, but the norm of A is a single number. Thus the smallness or largeness of the norm of an error matrix E does not truly reflect the smallness or largeness of the individual entries of E. For example, if $E = (10, 0.00001, 1)^T$, then $\|E\| = 10.0499$. Thus, the small entry 0.00001 is not reflected in the norm measure.

Examples of Stable and Unstable Algorithms by Backward Error Analysis

Example 3.2.1 A Stable Algorithm—Solution of an Upper Triangular System by Back Substitution Consider Algorithm 3.1.3 (the back-substitution method). Suppose the algorithm is implemented using accumulation of inner product in double precision. Then it can be shown (see Chapter 11) that the computed solution $\hat{x}$ satisfies

$$(T + E)\hat{x} = b$$

where the entries of the error matrix E are quite small. In fact, if $E = (e_{ij})$ and $T = (t_{ij})$, then

$$|e_{ij}| \le c\mu|t_{ij}|, \qquad i, j = 1, \ldots, n$$

and c is a constant of order unity. Thus, *the back-substitution process for solving an upper triangular system is stable.*

Example 3.2.2 An Unstable Algorithm—Gaussian Elimination Without Pivoting

Consider the problem of solving the nonsingular linear system $Ax = b$ using Gaussian elimination (Algorithm 3.1.5).

It has been shown by Wilkinson (see Chapter 11 of this book) that, when the process does not break down, the computed solution $\hat{x}$ satisfies

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

with

$$\|E\|_\infty \le cn^3\rho\|A\|_\infty\mu + 0(\mu^2)$$

where

$$A^{(k)} = (a_{ij}^{(k)})$$

are the reduced matrices in the elimination process and ρ, known as the **growth factor**, is given by

$$\rho = \frac{\max_k \max_{i,j} |a_{ij}^{(k)}|}{\max_{i,j} |a_{ij}|}$$

More specifically, if $\alpha = \max_{i,j} |a_{i,j}|$ and $\alpha_k = \max_{i,j} |a_{i,j}^{(k)}|$, then the growth factor ρ is given by

$$\rho = \frac{\max(\alpha, \alpha_1, \ldots, \alpha_{n-1})}{\alpha}$$

For an arbitrary matrix A, ρ can be quite large, because the entries of the reduced matrices $A^{(k)}$ can grow arbitrarily. To see this, consider the simple matrix

$$A = \begin{pmatrix} 10^{-10} & 1 \\ 1 & 2 \end{pmatrix}$$

One step of Gaussian elimination using nine-decimal-digit floating-point arithmetic will yield the reduced matrix

$$A^{(1)} = \begin{pmatrix} 10^{-10} & 1 \\ 0 & 2 - 10^{10} \end{pmatrix} = \begin{pmatrix} 10^{-10} & 1 \\ 0 & -10^{10} \end{pmatrix}$$

The growth factor for this problem is then

$$\rho = \frac{\max(\alpha, \alpha_1)}{\alpha} = \frac{\max(2, 10^{10})}{2} = \frac{10^{10}}{2}$$

which is quite large. Thus, if we now solve a linear system with this reduced upper triangular matrix, we cannot expect a small error matrix E. Indeed, if we wish to solve

$$10^{-10}x_1 + x_2 = 1$$
$$x_1 + 2x_2 = 3$$

using the preceding $A^{(1)}$, the computed solution will be $x_1 = 0$, $x_2 = 1$, whereas the exact solution is $x_1 = x_2 = 1$. Note that in solving the above system using Gaussian elimination, the vector

$$b = \begin{pmatrix} 1 \\ 3 \end{pmatrix}$$

will be modified to

$$b^{(1)} = \begin{pmatrix} 1 \\ 3 - 10^{10} \end{pmatrix} = \begin{pmatrix} 1 \\ -10^{10} \end{pmatrix}$$

(in nine-digit arithmetic). This shows that *Gaussian elimination is unstable for an arbitrary linear system.*

Note: Gaussian elimination without pivoting is not unstable for all matrices. There are certain classes of matrices, such as *symmetric positive* definite matrices, for which Gaussian elimination is stable. We discuss this special system in Chapter 6 in some detail.

If an algorithm is stable for a given matrix A, then it will be good to know if the algorithm is stable for every matrix A in a given class.

DEFINITION 3.2.4 An algorithm is **stable for a class of matrices** C if for every matrix A in C, the computed solution by the algorithm is the exact solution of a nearby problem.

Thus, for the linear system problem

$$Ax = b$$

an algorithm is stable for a class of matrices C if for every $A \in C$ and for each b, it produces a computed solution $\hat{x}$ that satisfies

$$(A + E)\hat{x} = b + \delta b$$

for some E and δb, where $(A + E)$ is close to A and $b + \delta b$ is close to b.

3.3 Conditioning of the Problem and Perturbation Analysis

From the preceding discussion we should not form the opinion that if a stable algorithm is used to solve a problem, then the computed solution will be accurate.

A property of the problem called **conditioning** also contributes to the accuracy or inaccuracy of the computed result.

The conditioning of a problem is a property of the problem itself. It is concerned with how the solution of the problem will change if the input data contains some impurities. This concern arises from the fact that in practical applications, the data very often come from some experimental observations where the measurements can be subjected to disturbances (or "noise") in the data. There are other sources of error also, such as roundoff errors (discussed in Chapter 11), discretization errors, and so on. Thus, when a numerical analyst has a problem in hand to solve, he or she must frequently solve the problem not with the original data, but with data that have been perturbed. The question naturally arises: What effects do these perturbations have on the solution?

A theoretical study done by numerical analysts to investigate these effects, which is independent of the particular algorithm used to solve the problem, is called **perturbation analysis**. This study helps us detect whether a given problem is "bad" or "good" in the sense of whether small perturbations in the data will create a large or small change in the solution. Specifically we use the following standard definition.

DEFINITION 3.3.1 A problem (with respect to a given set of data) is called an **ill conditioned** or **badly conditioned** problem if a small relative error in data causes a large relative error in the solution, regardless of the method of solution. Otherwise, it is called **well conditioned**.

Suppose a problem P is to be solved with an input c. Let $P(c)$ denote the value of the problem with the input c. Let δc denote the perturbation in c. Then P is said to be ill conditioned for the input data c if the relative error in the answer,

$$\frac{|P(c + \delta c) - P(c)|}{|P(c)|}$$

is much larger than the relative error in the data,

$$\frac{|\delta c|}{|c|}$$

Note: The definition of conditioning is data-dependent. Thus, a problem that is ill conditioned for *one* set of data could be well conditioned for *another* set.

3.4 Conditioning of the Problem, Stability of the Algorithm, and Accuracy of the Solution

As stated in the previous section, the conditioning of a problem is a property of the problem itself and has nothing to do with the algorithm used to solve the problem. To a user, of course, the accuracy of the computed solution is of primary importance. However, the accuracy of a computed solution by a given algorithm is directly

connected with both the stability of the algorithm and the conditioning of the problem. *If the problem is ill conditioned, no matter how stable the algorithm is, the accuracy of the computed solution cannot be guaranteed.*

Backward Stability and Accuracy

Note that the definition of backward stability does not say that the computed solution $\hat{x}$ by a backward stable algorithm will be close to the exact solution of the original problem. However, when a stable algorithm is applied to a well-conditioned problem, the computed solution should be near the exact solution because, in this case, stability will guarantee the exact solution of a nearby problem and well-conditioning will guarantee that the solution to the original problem and that of the nearby problem are close.

The ill-conditioning of a problem contaminates the computed solution, even with the use of a stable algorithm, therefore yielding an unacceptable solution. When a computed solution is unsatisfactory, some users (who are not usually concerned with conditioning) tend to put the blame on the algorithm for the inaccuracy. This is unfair. However, if a "stable" algorithm is applied to an ill-conditioned problem, it should not introduce more error than what the data warrants.

From the previous discussion, it is quite clear now that investigating the conditioning of a problem is very important.

The Condition Number of a Problem

Numerical analysts usually try to associate a number called the **condition number** with a problem. The condition number indicates whether the problem is ill or well conditioned. More specifically, the condition number gives a bound for the relative error in the solution when a small perturbation is applied to the input data.

In numerical linear algebra, condition numbers for many (but not all) problems have been identified. Unfortunately, computing the condition number is often more involved and time consuming than solving the problem itself. For example (as we shall see in Chapter 6), for the linear system problem $Ax = b$, the condition number is

$$\text{Cond}(A) = \|A\| \, \|A^{-1}\|$$

Thus, computing the condition number in this case involves computing the inverse of A; it is more expensive to compute the inverse than to solve the system $Ax = b$. In Chapter 6 we shall discuss methods for estimating Cond(A) without explicitly computing A^{-1}.

We shall discuss conditioning of each problem in detail in the relevant chapter. Before closing this section, however, let's mention several well-known examples of ill-conditioned problems.

An Ill-Conditioned Subtraction

Consider the subtraction $c = a - b$:

$$a = 12{,}354{,}101$$
$$b = 12{,}345{,}678$$
$$c = a - b = 8423$$

Now perturb a in the sixth place:

$$\hat{a} = 12{,}354{,}001$$
$$\hat{c} = c + \delta c = \hat{a} - b = 8323$$

Thus, a perturbation in the sixth digit in the input value caused a change in the second digit in the answer. Note that the relative error in the data is

$$\frac{a - \hat{a}}{a} = 0.000008$$

whereas the relative error in the computed result is

$$\frac{c - \hat{c}}{c} = 0.01187722$$

An Ill-Conditioned Root-Finding Problem

Consider solving the simple quadratic equation

$$f(x) = x^2 - 2x + 1 = 0$$

The roots are $x = 1, 1$. Now perturb the coefficient 2 by 0.00001. The computed roots of the perturbed polynomial $\hat{f}(x) = x^2 - 2.00001x + 1$ are $x_1 = 1.0032$ and $x_2 = 0.9968$. Relative errors in x_1 and x_2 are 0.0032. The relative error in the data is 5×10^{-6}.

3.5 The Wilkinson Polynomial

The preceding example involved multiple roots. Multiple roots or roots close to each other invariably make the root-finding problem ill conditioned; however, the problem can be ill conditioned even when the roots are very well separated. Consider the following well-known example by Wilkinson (see also Forsythe, Malcolm, and Moler 1977, 18–19).

$$\begin{aligned} P(x) &= (x-1)(x-2)\cdots(x-20) \\ &= x^{20} - 210x^{19} + \cdots \end{aligned}$$

The zeros of $P(x)$ are $1, 2, \ldots, 20$ and are distinct. Now perturb the coefficient of x^{19} from -210 to $-210 + 2^{-23}$, leaving other coefficients unchanged. Wilkinson used a binary computer with $t = 30$. Therefore, this change signified a change in the 30th significant base 2 digit. The roots of the perturbed polynomial, carefully computed by Wilkinson, were found to be (reproduced from Forsythe, Malcolm, and Moler 1977, 18):

1.00000	0000	10.09526	6145	±	0.64350	0904i
2.00000	0000	11.79363	3881	±	1.65232	9728i
3.00000	0000	13.99235	8137	±	2.51883	0070i
4.00000	0000	16.73073	7466	±	2.81262	4894i
4.99999	9928	19.50243	9400	±	1.94033	0347i
6.00000	6944					
6.99969	7234					
8.00726	7603					
8.91725	0249					
20.84690	8101					

The table shows that certain zeros are more sensitive to the perturbation than are others. The following analysis, due to Wilkinson (see also Forsythe, Malcolm, and Moler 1977, 19), attempts to explain this phenomenon.

Let the perturbed polynomial be

$$P(x, \alpha) = x^{20} - \alpha x^{19} + \cdots$$

Then the (condition) number

$$\left.\frac{\partial x}{\partial \alpha}\right|_{x=i}$$

measures the sensitivity of the root $x = i, i = 1, 2, \ldots, 20$. To compute this number, differentiate the equation $P(x, \alpha) = 0$ with respect to α:

$$\frac{\partial x}{\partial \alpha} = \frac{-\partial P/\partial \alpha}{\partial P/\partial x} = \frac{x^{19}}{\sum_{i=1}^{20} \sum_{\substack{j=1 \\ j \neq i}}^{20} (x - j)}$$

The values of $\left.\frac{\partial x}{\partial \alpha}\right|_{x=i}$ for $i = 1, 2, \ldots, 20$ are listed next. (See Forsythe, Malcolm, and Moler 1977, 19.)

Root	$\partial x/\partial \alpha\|_{x=i}$	Root	$\partial x/\partial \alpha\|_{x=i}$
1	-8.2×10^{-18}	11	-4.6×10^{7}
2	8.2×10^{-11}	12	2.0×10^{8}
3	-1.6×10^{-6}	13	-6.1×10^{8}
4	2.2×10^{-3}	14	1.3×10^{9}
5	-6.1×10^{-1}	15	-2.1×10^{9}
6	5.8×10^{1}	16	2.4×10^{9}
7	-2.5×10^{3}	17	-1.9×10^{9}
8	6.0×10^{4}	18	1.0×10^{9}
9	-8.3×10^{5}	19	-3.1×10^{8}
10	7.6×10^{6}	20	4.3×10^{7}

Root-finding and Eigenvalue Computation

The preceding examples teach us a very useful lesson: *It is not a good idea to compute the eigenvalues of a matrix by explicitly finding the coefficients of the characteristic polynomial and evaluating its zeros*, since the roundoff errors in computations will invariably put some small perturbations in the computed coefficients of the characteristic polynomial, and these small perturbations in the coefficients may cause large changes in the zeros. The eigenvalues are then computed inaccurately.

3.6 An Ill-Conditioned Linear System Problem

The matrix

$$H = \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \cdots & \frac{1}{n} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \cdots & \frac{1}{n+1} \\ \vdots & \vdots & \vdots & & \vdots \\ \frac{1}{n} & \frac{1}{n+1} & \frac{1}{n+2} & \cdots & \frac{1}{2n-1} \end{pmatrix}$$

is called the **Hilbert matrix** after the celebrated mathematician David Hilbert. The linear system problem, even with a Hilbert matrix of moderate order, is extremely ill conditioned. For example, take $n = 5$ and consider solving

$$Hx = b$$

where $b \approx (2.2833, 1.4500, 1.0929, 0.8845, 0.7456)$. The exact solution is $x = (1, 1, 1, 1, 1,)^T$. Now perturb the $(5, 1)$ element of H in the fifth place to obtain 0.20001. The computed solution with this very slightly perturbed matrix is $(0.9937, 1.1252, 0.4365, 1.8765, 0.5618)^T$. Note that $\text{Cond}(H) = O(10^5)$.

For more examples of ill-conditioned linear system problems, see Chapter 6.

3.7 Examples of Ill-Conditioned Eigenvalue Problems

Example 3.7.1 Consider the 10×10 matrix:

$$A = \begin{pmatrix} 1 & 1 & & & \\ & 1 & 1 & 0 & \\ & & \ddots & \ddots & \\ & 0 & & \ddots & 1 \\ & & & & 1 \end{pmatrix}$$

The eigenvalues of A are all 1. Now perturb the $(10, 1)$ coefficient of A by a small quantity $\epsilon = 10^{-10}$. Then the eigenvalues of the perturbed matrix computed using the MATLAB software (described in the next chapter) that uses a numerically effective

eigenvalue-computation algorithm were found to be:

$$
\begin{array}{rcl}
0 & & \\
1.0184 & + & 0.0980i \\
0.9506 & + & 0.0876i \\
1.0764 & + & 0.0632i \\
0.9051 & + & 0.0350i \\
1.0999 & + & 0.00i \\
1.0764 & - & 0.0632i \\
0.9051 & - & 0.0350i \\
1.0184 & - & 0.0980i \\
0.9506 & - & 0.0876i
\end{array}
$$

(Note the change in the eigenvalues.)

Example 3.7.2 The Wilkinson Bidiagonal Matrix Again, it should not be thought that an eigenvalue problem can be ill conditioned only when the eigenvalues are multiple or are close to each other. An eigenvalue problem with well-separated eigenvalues can be very ill conditioned too. Consider the 20×20 triangular matrix (known as the Wilkinson bidiagonal matrix):

$$
A = \begin{pmatrix} 20 & 20 & & & \\ & 19 & 20 & 0 & \\ & & \ddots & \ddots & \\ & 0 & & \ddots & 20 \\ & & & & 1 \end{pmatrix}
$$

The eigenvalues of A are $1, 2, \ldots, 20$. Now perturb the $(20, 1)$ entry of A by $\epsilon = 10^{-10}$. If the eigenvalues of this slightly perturbed matrix are computed using a stable algorithm (such as the QR iteration method to be described in Chapter 8), it will be seen that some of them will change drastically; they will even become complex.

In this case also, certain eigenvalues are more ill conditioned than others. To explain this, Wilkinson computed the condition number of each of the eigenvalues. The condition number of the eigenvalue λ_i of a nondefective matrix A is defined to be (see Chapter 8):

$$
\boxed{\mathrm{Cond}(\lambda_i) = \frac{1}{|y_i^T x_i|}}
$$

where y_i and x_i are, respectively, the unit left and right eigenvectors of A corresponding to the eigenvalue λ_i (recall that x is a right eigenvector of A associated with an eigenvalue λ if $Ax = \lambda x$, $x \neq 0$; similarly, y is a left eigenvector associated with λ if $y^T A = \lambda y^T$).

In our case, the right eigenvector x_r corresponding to $\lambda_r = r$ has components (see Wilkinson 1965, 91):

$$
\left[1, \frac{20-r}{-20}, \frac{(20-r)(19-r)}{(-20)^2}, \ldots, \frac{(20-r)!}{(-20)^{20-r}}, 0, \ldots, 0\right]
$$

and the components of y_r are

$$
\left[0, 0, \ldots, 0, \frac{(r-1)!}{20^{r-1}}, \ldots, \frac{(r-1)(r-2)}{20^2}, \frac{r-1}{20}, 1\right]
$$

These vectors are not quite 2-normalized; even so, the reciprocal of their products gives us an estimate of the condition numbers.

In fact, K_r, the condition number of the eigenvalue $\lambda = r$, is

$$
|K_r| = \frac{1}{|y_r^T x_r|} = \frac{20^{19}}{(20-r)!\,(r-1)!}
$$

The number K_r is large for all values or r. The smallest K_i for the Wilkinson bidiagonal matrix are $K_1 = K_{20} \approx 4.31 \times 10^7$ and the largest ones are $K_{11} = K_{10} \approx 3.98 \times 10^{12}$.

Example 3.7.3 (Wilkinson 1965, 92)

$$
A = \begin{pmatrix}
n & (n-1) & (n-2) & \cdots & 3 & 2 & 1 \\
(n-1) & (n-1) & (n-2) & \cdots & 3 & 2 & 1 \\
 & (n-2) & (n-2) & \cdots & 3 & 2 & 1 \\
 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
 & & & & 2 & 2 & 1 \\
 & & & & & 1 & 1
\end{pmatrix}
$$

As n increases, the smallest eigenvalues become progressively ill conditioned. For example, when $n = 12$, the condition numbers of the first few eigenvalues are of order unity, whereas those of the last three are of order 10^7.

3.8 Strong, Weak, and Mild Stability

While establishing the stability of an algorithm by backward error analysis, we sometimes get much more than the preceding definition of stability calls for. For example, it can be shown that when Gaussian elimination with *partial* and *complete pivoting* (for discussions on partial and complete pivoting, see Chapter 5) is applied to solve a *nonsingular* system $Ax = b$, the computed solution $\hat{x}$ not only satisfies

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

with a small error matrix E, but $(A + E)$ is also found to be *nonsingular*. The standard definition of stability, of course, does not need that.

On the other hand, if Gaussian elimination without pivoting is applied to a symmetric positive definite system, then *the standard backward error analysis* shows that the computed solution $\hat{x}$ satisfies $(A + E)\hat{x} = b$ with a small error matrix E, but it does not show that $(A + E)$ is symmetric positive definite.

Thus, we may talk about two types of stable algorithms, one type giving not only a small error matrix but also the perturbed matrix $A + E$ belonging to the same class as the matrix A itself and the other type just giving a small error matrix without any restriction on $A + E$.

To distinguish between these two types of stability, Bunch (1987) recently introduced the concept of strong stability. Following Bunch we give the following definition.

DEFINITION 3.8.1 An algorithm for solving the linear system problem $Ax = b$ is **strongly stable** for a class of matrices C if, for each A in C the computed solution is the exact solution of a nearby problem, and the matrix $(A + E)$ also belongs to C.

Examples (Bunch)

1. The Gaussian elimination with pivoting is strongly stable on the class of nonsingular matrices.
2. The Cholesky algorithm for computing the factorization of a symmetric positive definite matrix of the form $A = HH^T$ is strongly stable on the class of symmetric positive definite matrices. (See Chapter 6 for a description of the Cholesky algorithm.)
3. The Gaussian elimination without pivoting is strongly stable on the class of nonsingular diagonally dominant matrices.

With an analogy of this definition of strong stability, Bunch also introduced the concept of **weak stability** that depends upon the conditioning of the problem.

DEFINITION 3.8.2 An algorithm is **weakly stable** for a class of matrices C if for each well-conditioned matrix in C the algorithm produces an acceptable accurate solution.

Thus, an algorithm for solving the linear system $Ax = b$ is weakly stable for a class of matrices C if for each well-conditioned matrix A in C and for each b, the computed solution $\hat{x}$ to $Ax = b$ is such that

$$\frac{\|x - \hat{x}\|}{\|x\|} \text{ is small}$$

Bunch was motivated to introduce this definition to point out that the well-known Levinson algorithm for solving linear systems involving *Toeplitz* matrices (a matrix $T = (t_{ij})$ is Toeplitz if the entries along each diagonal are the same) is weakly stable on the class of symmetric positive definite Toeplitz matrices. This very important and remarkable result was proved by Cybenko (1980). The result was important because the signal-processing community had been using the Levinson algorithm routinely for years without fully investigating the stability behavior of this important algorithm. [Bojanczyk, Brent, de Hoog, and Sweet (1994) recently proved that a related algorithm for Toeplitz matrices, known as the Bareiss algorithm, has better numerical properties than the Levinson algorithm.]

Remarks on Stability, Strong Stability, and Weak Stability:

1. If an algorithm is strongly stable, it is necessarily stable.
2. Note that *stability* implies *weak stability*. Weak stability is good enough for most users.
3. If a numerical analyst can prove that a certain algorithm is not weakly stable, then it follows that the algorithm is *not* stable because *not weakly stable* implies *not stable*.

Mild Stability. We have defined an algorithm to be backward stable if the algorithm produces a solution that is an exact solution of a nearby problem. But it might very well happen that an algorithm produces a solution that is only close to the exact solution of a nearby problem. What should we call such an algorithm? Van Dooren (1991), following de Jong (1977) has called such an algorithm a mixed stable algorithm. We believe that it is more appropriate to call such stability *mild stability*. After all, such an algorithm is stable in the mild sense.

DEFINITION 3.8.3 An algorithm is **mildly stable** if it produces a solution that is close to the exact solution of a nearby problem.

Example 3.8.1

1. The QR algorithm for the rank-deficient least-squares problems is mildly stable (see Lawson and Hanson 1974, 95 and Chapter 7 of this book).
2. The QR algorithm for the full-rank underdetermined linear system problem is mildly stable (see Lawson and Hanson 1974, 93 and Chapter 7 of this book).

3.9 Review and Summary

In this chapter we have introduced two of the most important concepts in numerical linear algebra, namely, the conditioning of the problem and stability of the algorithm, and have discussed how they affect the accuracy of the solution.

1. *Conditioning of the problem*: The conditioning of the problem is a property of the problem. A problem is said to be ill conditioned if a small change in the data causes a large change in the solution, otherwise it is well conditioned. The conditioning of a problem is data dependent. A problem can be ill conditioned with respect to one set of data but quite well conditioned with respect to another set.

   Ill-conditioning or well-conditioning of a matrix problem is generally measured by means of a number called the *condition number*. The condition number for the linear system problem $Ax = b$ is $\|A\| \, \|A^{-1}\|$.

   The well-known examples of ill-conditioned problems are the Wilkinson polynomial for the root-finding problem, the Wilkinson bidiagonal ma-

trix for the eigenvalue problem, the Hilbert matrix for the algebraic linear system problem, and so on.

2. *Stability of an algorithm*: An algorithm is said to be a *backward stable algorithm* if it computes the exact solution of a nearby problem. Some examples of stable algorithms are back substitution and forward elimination for triangular systems, Gaussian elimination with pivoting for linear systems, QR factorization using Householder and Givens transformations, and QR iteration algorithm for eigenvalue computations.

   The Gaussian elimination algorithm without row changes is unstable for arbitrary matrices.

3. *Effects of conditioning and stability on the accuracy of the solution*: The conditioning of the problem and the stability of the algorithm both have effects on accuracy of the solution computed by the algorithm.

   If a stable algorithm is applied to a well-conditioned problem, it should compute an accurate solution. On the other hand, if a stable algorithm is applied to an ill-conditioned problem, there is no guarantee that the computed solution will be accurate. The definition of backward stability does not imply that. However, if a stable algorithm is applied to an ill-conditioned problem, it should not introduce more errors than what the data warrants.

EXERCISES

1. (a) Show that the floating-point computations of the sum, product, and quotient of two numbers are backward and forward stable.
   (b) Are the floating-point computations of the inner and outer product of two vectors backward stable? Give reasons for your answer.
2. Find the growth factor of Gaussian elimination without pivoting for the following matrices.

$$\begin{pmatrix} 0.00001 & 1 \\ 1 & 1 \end{pmatrix}, \quad \begin{pmatrix} 1 & 1 \\ 0.00001 & 1 \end{pmatrix}, \quad \begin{pmatrix} 1 & -1 & -1 & -1 & -1 \\ 0 & 1 & -1 & -1 & -1 \\ 0 & 0 & 1 & -1 & -1 \\ 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 1 \\ 1 & 0.9 \end{pmatrix}, \quad \begin{pmatrix} 1 & 1 \\ 1 & 0.99 \end{pmatrix}, \quad \begin{pmatrix} 1 & 1 \\ 1 & 0.999 \end{pmatrix}, \quad \begin{pmatrix} 1.0001 & 1 \\ 1 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 0.9 & 0.81 \\ 1 & 1.9 & 3.61 \end{pmatrix}, \quad \begin{pmatrix} 1 & 1 & 1 \\ 0.9 & 0.9 & 0.9 \\ 1 & 1.9 & 3.61 \end{pmatrix}, \quad \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \end{pmatrix}$$

3. Find the condition number of each of the matrices of Exercise 2.
4. (a) Show that $\mathrm{Cond}(cA) = \mathrm{Cond}(A)$ for all nonzero scalars c.
   (b) Show that if $\|I\| \geq 1$, then $\mathrm{Cond}(A) \geq 1$.

5. Prove that

$$\frac{\|\,\mathrm{fl}(AB) - AB\|_F}{\|AB\|_F} \leq n\mu\,\mathrm{Cond}_F(B) + O(\mu^2)$$

   where A and B are matrices and B is nonsingular. ($\mathrm{Cond}_F(B) = \|B\|_F\|B^{-1}\|_F$.)
6. Are the following floating-point computations backward stable? Give reasons for your answer in each case.
   (a) $\mathrm{fl}(x(y+z))$
   (b) $\mathrm{fl}(x_1 + x_2 + \cdots + x_n)$
   (c) $\mathrm{fl}(x_1 x_2 \cdots x_n)$
   (d) $\mathrm{fl}(x^T y/c)$, where x and y are vectors and c is a scalar
   (e) $\mathrm{fl}\left(\sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}\right)$
7. Find the growth factor of Gaussian elimination for each of the following matrices and hence conclude that Gaussian elimination for linear systems with these matrices is backward stable.

   (a) $\begin{pmatrix} 10 & 1 & 1 \\ 1 & 10 & 1 \\ 1 & 1 & 10 \end{pmatrix}$

   (b) $\begin{pmatrix} 4 & 0 & 2 \\ 0 & 4 & 0 \\ 2 & 0 & 5 \end{pmatrix}$

   (c) $\begin{pmatrix} 10 & 1 & 1 \\ 1 & 15 & 5 \\ 1 & 5 & 14 \end{pmatrix}$
8. Show that Gaussian elimination without pivoting for the matrix

$$\begin{pmatrix} 10 & 1 & 1 \\ 1 & 10 & 1 \\ 1 & 1 & 10 \end{pmatrix}$$

   is strongly stable.
9. Let H be an unreduced upper Hessenberg matrix. Find a diagonal matrix D such that $D^{-1}HD$ is a normalized upper Hessenberg matrix (that is, all subdiagonal entries are 1). Show that the transforming matrix D must be ill conditioned if one or several subdiagonal entries of H are very small. Do a numerical example of order 5 to verify this.
10. Show that the roots of the following polynomials are ill conditioned:
    (a) $x^3 - 3x^2 + 3x - 1$
    (b) $(x-1)^3(x-2)$
    (c) $(x-1)(x-0.99)(x-2)$
11. State a mathematical result that shows that the matrix–vector multiplication with an ill-conditioned matrix may give rise to a large relative error in the computed result. Construct your own 2×2 example to see this.
12. (a) Using Gaussian elimination without pivoting, solve the linear systems $Ax = b$ with each matrix A in Exercise 2, and choosing b in each case such that the exact solution x has all entries equal to 1.

(b) For each of the systems in 12(a), compute the relative error

$$\frac{\|x - \hat{x}\|}{\|x\|}$$

where $\hat{x}$ is the computed solution.

(c) If the relative error for any solution is large, explain what happened.

13. Using Exercise 5, prove that the error in matrix multiplication $B_1B_2 \ldots B_NAC_1C_2 \ldots C_N$ will be magnified at most by

$$\prod_{i=1}^{N} \text{Cond}(B_i)\,\text{Cond}(C_i)$$

Suggestions for Further Reading

For computer codes of standard matrix computations, see the book *Handbook for Matrix Computations* by T. Coleman and Charles Van Loan (Philadelphia: SIAM, 1988). The concepts of stability and conditioning are thoroughly discussed in the book *An Introduction to Matrix Computations* by G. W. Stewart (New York: Academic Press, 1973, Chapter 2). Note that Stewart's definition of backward stability is slightly different than the usual definition of backward stability introduced by Wilkinson. We also strongly suggest the readers to read an illuminating paper by James R. Bunch in the area: "The Weak and Strong Stability of Algorithms in Numerical Linear Algebra," *Lin. Alg. Appl.* 88-89 (1987): 49–66. See also the paper by Bunch, Demmel, and Van Loan (1989).

Wilkinson's book *The Algebraic Eigenvalue Problem* (see Chapter 2 Further Readings) is a rich source of knowledge for results on backward stability for matrix algorithms.

An important paper discussing notions and concepts of different types of stability in general is a paper by L. S. de Jong (1977).

A recent book by Stewart and Sun (1990) is devoted to the theory of perturbations of various matrix problems and is a rich source of information in this area. For advanced readers, Rice's (1966) seminal paper and the celebrated paper of Kahan (1966) are highly recommended.

CHAPTER FOUR

Numerically Effective Algorithms and Mathematical Software

4.1 Definitions and Examples

Solving a problem on a computer involves the following major steps performed in sequence:

1. *Making a mathematical model of the problem*—that is, translating the problem into the language of mathematics. For example, mathematical models of many engineering problems are sets of ordinary and partial differential equations.
2. *Finding or developing constructive methods* for solving the mathematical model. This step usually consists of a literature search to find what methods are available for the problems.
3. *Identifying the best method from a numerical point of view* (the best one may be a combination of several others). We call this method the *numerically effective method.*
4. Finally, *implementing on the computer* the numerically effective method identified in step 3. This amounts to writing and executing a reliable and efficient computer program based on the identified numerically effective method and may also require exploitation of the target computer architecture.

The purpose of creating mathematical software is to provide a scientist or engineer with computer programs he or she can use with confidence to solve a problem. Thus, mathematical software should be of high quality.

Let's be specific about what we mean when we call software *high-quality mathematical software*. High quality mathematical software should ideally have the following features (Rice, 1981):

1. *Power and flexibility:* It can be used to solve several different variations of the original problem and the closely associated problems. For example, closely associated with the linear system $Ax = b$ problem are:

   (a) Computing the inverse of A—that is, finding an X such that $AX = I$. Although finding the inverse of A and solving $Ax = b$ are equivalent

problems, solution of a linear system using the inverse of the system matrix is *not* advisable. Computing the inverse explicitly should be avoided, unless a specific application really calls for it.

(b) Finding the determinant and rank of A.

(c) Solving $AX = B$, where B is a matrix.

Also, a matrix problem may have some special structures. It may be positive definite, banded, Toeplitz, dense, sparse, and so on. The software should state clearly what variations of the problem it can handle and whether it is special-structure oriented.

2. *Easily read and modified:* The software should be well documented. The documentation should be clear and easy to read, even for a nontechnical user, so that if some modifications are needed, they can be made easily. To quote from the cover page of Forsythe, Malcolm, and Moler (1977):

> ... it is an order of magnitude easier to write two good subroutines than to decide which one is best. In choosing among the various subroutines available for a particular problem, we placed considerable emphasis on the clarity and style of programming. If several subroutines have comparable accuracy, reliability, and efficiency, we have chosen the one that is the least difficult to read and use.

3. *Portability:* It should be able to run on different computers with few or no changes.
4. *Robustness:* It should be able to deal with an unexpected situation during execution.
5. *Based on a numerically effective algorithm:* It should be based on an algorithm that has attractive numerical properties.

We have used the expression *numerically effective* several times without qualification. This is the most important component of high-quality mathematical software. We shall call a matrix algorithm **numerically effective** if it is

1. *General-purpose:* The algorithm should work for a wide class of matrices.
2. *Reliable:* The algorithm should give warning whenever it is on the verge of breakdown due to excessive roundoff errors or not being able to meet some specified criterion of convergence. There are algorithms that produce completely wrong answers without giving warning at all. *Gaussian elimination without pivoting* (for the linear system or equivalent problem) is one such algorithm. It is *not reliable*.
3. *Stable:* Total rounding errors of the algorithm should not exceed the errors that are inherent in the original problem (see Sections 3.2 and 3.3).
4. *Efficient:* The efficiency of an algorithm is measured by the amount of computer time consumed in its implementation. Theoretically, the number of floating-point operations needed to implement the algorithm indicates its efficiency.

DEFINITION 4.1.1 A floating-point operation, or **flop**, is the amount of computer time required to execute the Fortran statement

```
A(I,J) = A(I,J) + t * A(I,J)
```


A flop involves one multiplication, one addition, and some subscript manipulations. Similarly, one division coupled with an addition or subtraction will be counted as one flop. This definition of a flop has been used in the popular software package LINPACK (this package is briefly described in Section 4.3).

A Note on the Definition of Flop Count: With the advent of supercomputing technology, there is a tendency to count an addition or subtraction as a flop as well. This definition of a flop has been adopted in the second edition of the book by Golub and Van Loan (1989). However, we have decided to stick to the original LINPACK definition. Note that if an addition (or subtraction) is counted as a flop, then the "new flop" has twice the value of the "old flop."

DEFINITION 4.1.2 A matrix algorithm involving computation with matrices of order n will be called an **efficient** algorithm if it takes *no more than* $O(n^3)$ flops. (The historical *Cramer's rule* for solving a linear system is, therefore, *not efficient*, since $O(n!)$ flops are required for its execution. See Chapter 6.)

One point is well worth mentioning here. An algorithm may be efficient but still unstable. For example, Gaussian elimination without pivoting requires about $n^3/3$ flops for an $n \times n$ matrix. Therefore, although it is efficient, it is unreliable and unstable for an arbitrary matrix.

5. *Economic in the use of storage:* Usually, about n^2 storage locations are required to store a dense matrix of order n. Therefore, if an algorithm requires storage of several matrices during its execution, a large number of storage locations will be needed even when n is moderate. Thus, it is important to give special attention to economy of storage while designing an algorithm.

By carefully rearranging an algorithm, we can greatly reduce its storage requirement (examples of this are presented later). In general, if a matrix generated during execution of the algorithm is not needed for future use, it should be overwritten by another computed element.

Notation for Overwriting and Interchange

We will use the notation

$$a \equiv b$$

to denote that b *overwrites* a. Similarly, the interchange of two computed quantities a and b is written symbolically as

$$a \leftrightarrow b$$

4.2 Flop Count and Storage Considerations for Some Basic Algorithms

Numerical linear algebra often deals with triangular matrices, which can be stored using only $n(n+1)/2$ locations rather than n^2 locations. This useful fact should be

kept in mind while designing an algorithm in numerical linear algebra, so that the extra available space can be used for something else.

Sparse matrices, in general, have lots of zero entries. A convenient scheme for storing a sparse matrix involves storing only the nonzero entries. In the following, we illustrate the flop-count and storage scheme for some simple basic matrix computations with dense matrices.

> In determining the flop count of an algorithm, we note that the numbers of multiplications and additions in a matrix algorithm are roughly the same. Thus, a count only of the number of multiplications in a matrix algorithm gives us an idea of the total flop count for that algorithm (similar comments also hold for divisions and subtractions). Also, the *counts involving zero elements can be omitted.*

Example 4.2.1 Inner Product Computation Let

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \quad \text{and} \quad y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

be two n-vectors. Then the inner product $z = x^T y = \sum_{i=1}^{n} x_i y_i$ can be computed as (Algorithm 3.1.2):

$z = 0$
For $i = 1, 2, \ldots, n$ do
  $z = z + x_i y_i$

Just 1 flop is needed for each i. Then a total of n *flops* is needed to execute the algorithm.

Example 4.2.2 Outer Product Computation The outer product xy^T is an $n \times n$ matrix Z, as shown here. The (i, j)th component of the matrix is $x_i y_j$. Because there are n^2 components and each component requires one multiplication, *the outer-product computation requires n^2 flops and n^2 storage locations.* However, very often we do not require the matrix from the outer product explicitly.

$$Z = xy^T = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} (y_1 \quad y_2 \quad \ldots \quad y_n) = \begin{pmatrix} x_1y_1 & x_1y_2 & \ldots & x_1y_n \\ x_2y_1 & x_2y_2 & \ldots & x_2y_n \\ \vdots & \vdots & & \vdots \\ x_ny_1 & x_ny_2 & \ldots & x_ny_n \end{pmatrix}$$

Example 4.2.3 Matrix–Vector Product Let $A = (a_{ij})$ be an $n \times n$ matrix and let

$$b = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

be an n-vector. Then

$$Ab = \begin{pmatrix} a_{11}b_1 + a_{12}b_2 + \cdots + a_{1n}b_n \\ a_{21}b_1 + a_{22}b_2 + \cdots + a_{2n}b_n \\ \vdots \\ a_{n1}b_1 + a_{n2}b_2 + \cdots + a_{nn}b_n \end{pmatrix}$$

Flop count: Each component of the vector Ab requires n multiplications and $(n-1)$ additions. Since there are n components, the *computation of Ab requires about n^2 flops.*

Example 4.2.4 Matrix–Matrix Product with Upper Triangular Matrices Let $U = (u_{ij})$ and $V = (v_{ij})$ be two upper triangular matrices of order n. Then the following algorithm computes the product $C = UV$. The algorithm overwrites V with the product UV.

For $i = 1, 2, \ldots, n$ do
  For $j = i, i+1, \ldots, n$ do

$$v_{ij} \equiv c_{ij} = \sum_{k=i}^{j} u_{ik} v_{kj}$$

Note that in this pseudocode, j represents the inner loop and i represents the outer loop. For each value of i from 1 to n, j takes the values i through n.

Flop count:

1. Computing c_{ij} requires $j - i + 1$ multiplications.
2. Since j runs from i to n and i runs from 1 to n, the total number of multiplications is

$$\sum_{i=1}^{n}\sum_{j=i}^{n}(j - i + 1) = \sum_{i=1}^{n}(1 + 2 + \cdots + (n - i + 1))$$

$$= \sum_{i=1}^{n} \frac{(n-i+1)(n-i+2)}{2}$$

$$\approx \frac{n^3}{6} \qquad \text{(for large } n\text{)}$$

(Recall that $1 + 2 + 3 + \cdots + r = r(r+1)/2$ and $1^2 + 2^2 + 3^2 + \cdots + r^2 = r(r+1)(2r+1)/6$.)

> **Flop Count for the Product of Two Triangular Matrices**
>
> For large n, the product of two $n \times n$ triangular matrices requires about $\frac{n^3}{6}$ flops.

Example 4.2.5 Matrix–Matrix Multiplication Let A be an $m \times n$ matrix and B be a $n \times p$ matrix. Then the following algorithm computes the product $C = AB$.

For $i = 1, 2, \ldots, m$ do
  For $j = 1, 2, \ldots, p$ do
    $$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

Flop count: There are n multiplications in computing each c_{ij}. Since j runs from 1 to p and i runs from 1 to m, the total number of multiplications is mnp. Thus, for two square matrices A and B, each of order n, this count is n^3.

> **Flop Count for the Product of Two Matrices**
> Let A be $m \times n$ and B be $n \times p$. Then computing $C = AB$ requires mnp flops. In particular, it takes n^3 flops to compute the product of two $n \times n$ square matrices.

The preceding algorithm for matrix–matrix multiplication for two $n \times n$ matrices will obviously require n^2 storage locations for each matrix. However, it can be rewritten in such a way that there will be a substantial savings in storage, as illustrated below.

The following algorithm overwrites B with the product AB, assuming that an additional column has been annexed to B (alternatively, we can use a work vector to hold values temporarily). A and B are $n \times n$.

Example 4.2.6 Matrix–Matrix Product with Economy in Storage

For $j = 1, 2, \ldots, n$ do
  1. $h_i = \sum_{k=1}^{n} a_{ik} b_{kj}$ $(i = 1, 2, \ldots, n)$
  2. $b_{ij} \equiv h_i$ $(i = 1, 2, \ldots, n)$

(h is a temporary work vector).

Example 4.2.7 Computation of $\left(I - \frac{2uu^T}{u^T u}\right) A$ In numerical linear algebra we often need to compute

$$\left(I - \frac{2uu^T}{u^T u}\right) A$$

where I is an $m \times m$ identity matrix, u is an m-vector, and A is $m \times n$. The matrix $I - 2uu^T/u^T u$ is called a *Householder matrix* (see Chapter 5). Naively, we would form the matrix $(I - 2uu^T/u^T u)$ from the vector u and then form the matrix product explicitly with A. This would require $O(m^2 n)$ flops. We show next that this matrix product can implicitly be performed with $O(mn)$ flops. *The key observation here is that we do not need to form the matrix $(I - 2uu^T/u^T u)$ explicitly.*

The following algorithm computes the product. The algorithm overwrites A with the product. Let

$$\beta = \frac{2}{u^T u}$$

Then

$$\left(I - \frac{2uu^T}{u^T u}\right) A$$

becomes $A - \beta uu^T A$.

Let

$$u = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_m \end{pmatrix}$$

Then the (i, j)th entry of $(A - \beta uu^T A)$ is equal to $a_{ij} - \beta(u_1 a_{1j} + u_2 a_{2j} + \cdots + u_m a_{mj}) u_i$. Thus, we have the following algorithm.

ALGORITHM 4.2.1 Computing $\left(I - \frac{2uu^T}{u^T u}\right) A$

Step 1: Compute $\beta = \frac{2}{u^T u}$. (m+1) flop

Step 2: For $j = 1, 2, \ldots, n$ do
  $\alpha = u_1 a_{1j} + u_2 a_{2j} + \cdots + u_m a_{mj}$ n(m-1) flop
  $\alpha \equiv \beta \cdot \alpha$
  For $i = 1, 2, \ldots, m$ do
    $a_{ij} \equiv a_{ij} - \alpha u_i$ mn flop

MATCOM Note: Algorithm 4.2.1 has been implemented in the MATCOM program PHOUSMUL. See section 4.3.8 and Appendix B to learn more about MATCOM.

Flop count:

1. There are $m + 1$ flops needed to compute β (m flops to compute the inner product and 1 flop to divide 2 by the inner product).
2. There are n α's, and each α costs $(m + 1)$ flops. Thus, we need $n(m + 1)$ flops to compute the α's.
3. There are mn a_{ij} to compute. Each a_{ij} costs just 1 flop, once the α's are computed.

The total flop count is $(m + 1) + 2mn + n$.

We now summarize this important result (which is used repeatedly in this book) in the following:

> **Flop Count for the Product $\left(I - \frac{2uu^T}{u^T u}\right) A$**
> Let A be $m \times n$ $(m \geq n)$ and u be an m-vector. Then the preceding product can be computed with only $(m + 1) + 2mn + n$ flops. In particular, if $m = n$, then it takes roughly $2n^2$ flops, compared to n^3 flops if done naively.

A Numerical Example: Let

$$u = (1, 1, 1)^T, \qquad A = \begin{pmatrix} 1 & 1 \\ 2 & 1 \\ 0 & 0 \end{pmatrix}$$

Then

$$\beta = \frac{2}{3}$$

j=1: Compute the first column of the product.

$$\alpha = u_1 a_{11} + u_2 a_{21} + u_3 a_{31} = 1 + 2 = 3$$

$$\alpha \equiv \beta\alpha = \frac{2}{3} \times 3 = 2$$

$$a_{11} \equiv a_{11} - \alpha u_1 = 1 - 2 = -1$$

$$a_{21} \equiv a_{21} - \alpha u_2 = 2 - 2 \times 1 - 2 - 2 = -0$$

$$a_{31} \equiv a_{31} - \alpha u_3 = 0 - 2 \times 1 = -2$$

j=2: Compute the second column of the product.

$$\alpha = u_1 a_{12} + u_2 a_{22} + u_3 a_{32} = 1 + 1 = 2$$

$$\alpha \equiv \beta\alpha = 2 \times \frac{2}{3} = \frac{4}{3}$$

$$a_{12} \equiv a_{12} - \alpha u_1 = 1 - \frac{4}{3} \times 1 = -\frac{1}{3}$$

$$a_{22} \equiv a_{22} - \alpha u_2 = 1 - \frac{4}{3} \times 1 = -\frac{1}{3}$$

$$a_{32} \equiv a_{32} - \alpha u_3 = 0 - \frac{4}{3} = -\frac{4}{3}$$

Thus,

$$A \equiv \left(I - \frac{2uu^T}{u^T u}\right) A$$

$$= \begin{pmatrix} -1 & -\frac{1}{3} \\ 0 & -\frac{1}{3} \\ -2 & -\frac{4}{3} \end{pmatrix}$$

Example 4.2.8 Flop Count for Algorithm 3.1.3 (Back-Substitution Process) From the pseudocodes of this algorithm, we see that it takes 1 flop to compute y_n, 2 flops to compute y_{n-1}, and so on. Thus to compute y_1 through y_n, we need $(1 + 2 + 3 + \cdots + n) = n(n+1)/2 \approx n^2/2$ flops.

> **Flop Count for the Back-Substitution Process**
> It requires roughly $n^2/2$ flops to solve an upper triangular system using the back-substitution process.

Example 4.2.9 Flop Count and Storage Considerations for Algorithm 3.1.4 (The Inverse of an Upper Triangular Matrix) Let's state the algorithm once more here:

For $k = n, n-1, \ldots, 2, 1$ do

$$s_{kk} = \frac{1}{t_{kk}}$$

For $i = k-1, k-2, \ldots, 1$

$$s_{ik} = -\frac{1}{t_{ii}} \left(\sum_{j=i+1}^{k} t_{ij} s_{jk} \right)$$

Flop count: The count is approximately

$k = 1$	1 flop
$k = 2$	3 flops
$k = 3$	6 flops
$\vdots$	
$k = n$	$\frac{n(n+1)}{2}$ flops

Total flops:

$$1 + 3 + 6 + \cdots + \frac{n(n+1)}{2} = \sum_{r=1}^{n} \frac{r(r+1)}{2} = \sum_{r=1}^{n} \frac{r^2}{2} + \sum_{r=1}^{n} \frac{r}{2} \approx \frac{n^3}{6}$$

> **Flop Count for Computing the Inverse of an Upper Triangular Matrix**
> It requires about $\frac{n^3}{6}$ flops to compute the inverse of an upper triangular matrix.

Storage Considerations. Since the inverse of an upper triangular matrix is an upper triangular matrix and it is clear from the algorithm that we can overwrite t_{ik} by s_{ik}, *the algorithm can be rewritten so that it overwrites T with the inverse S.* Thus we can rewrite the algorithm as follows.

ALGORITHM 4.2.2 Computing the Inverse of an Upper Triangular Matrix with Economical Storage

For $k = n, n-1, \ldots, 2, 1$ do

$$t_{kk} \equiv s_{kk} = \frac{1}{t_{kk}}$$

For $i = k-1, k-2, \ldots, 1$ do

$$t_{ik} \equiv s_{ik} = -\frac{1}{t_{ii}} \sum_{j=i+1}^{k} t_{ij} s_{jk}$$

Math. Seminar

MATCOM Note: Algorithm 4.2.2 has been implemented in MATCOM program INVUPTR.

Example 4.2.10 Flop Count and Storage Considerations for Algorithm 3.1.5 (Gaussian Elimination) It is shown in Chapter 5 that the Gaussian elimination algorithm takes about $n^3/3$ flops. The algorithm can overwrite A with the upper triangular matrix $A^{(n-1)}$; in fact, it can overwrite A with each $A^{(k)}$. The multipliers m_{ik} can be stored in the lower half of A as they are computed. Each $b^{(k)}$ can overwrite the vector b.

ALGORITHM 4.2.3 Gaussian Elimination Algorithm with Economical Storage

For $k = 1, 2, \ldots, n-1$ do
  For $i = k+1, \ldots, n$ do
    $a_{ik} \equiv m_{ik} = -\dfrac{a_{ik}}{a_{kk}}$
    $b_i \equiv b_i + m_{ik} b_k$
    For $j = k+1, \ldots, n$ do
      $a_{ij} \equiv a_{ij} + m_{ik} a_{kj}$

MATCOM Note: Algorithm 4.2.3 has been implemented in MATCOM program GAUSS.

4.3 Some Existing High-Quality Mathematical Software for Linear Algebra Problems

Several high-quality mathematical software packages for various types of matrix computations are in existence. These are LINPACK, EISPACK, MATLAB, NETLIB, IMSL, NAG, and the most recently released, LAPACK, and TEMPLATES.

4.3.1 LINPACK

According to its users' guide, LINPACK is "a collection of Fortran subroutines which analyze and solve various systems of simultaneous linear algebraic equations. The subroutines are designed to be completely machine independent, fully portable, and to run at near optimum efficiency in most operating environments."

Although primarily intended for linear systems, the package also contains routines for singular value decomposition (SVD) and problems associated with linear systems, such as computing the inverse, the determinant, and the linear least-squares problem. Most of the routines are for square matrices, but some handle rectangular coefficient matrices associated with overdetermined or underdetermined problems.

The routines are meant for small and dense problems of order less than a few hundred and band matrices of order less than several thousand. There are no routines for iterative methods.

4.3.2 EISPACK

EISPACK is an eigensystem package. The package is designed primarily to compute the eigenvalues and eigenvectors of a matrix; however, it contains routines for the

generalized eigenvalue problem of the form $Ax = \lambda Bx$ and for the singular-value decomposition.

The eigenvalues of an arbitrary matrix A are computed in several sequential phases. First, the matrix A is balanced. If it is nonsymmetric the balanced matrix is then reduced to an upper Hessenberg matrix by similarity (if it is symmetric, it is reduced to symmetric tridiagonal). Finally the eigenvalues of the transformed upper Hessenberg or the symmetric tridiagonal matrix are computed using the implicit QR iterations or the Sturm sequence method. There are EISPACK routines to perform all these tasks. The implicit QR iteration method and the Sturm sequence methods are described in Chapter 8.

4.3.3 LAPACK

The building blocks of numerical linear algebra algorithms have three levels of BLAS (basic linear algebra subroutines):

- *Level 1 BLAS* are for vector–vector operations. A typical Level 1 BLAS is of the form $y \equiv \alpha x + y$, where x and y are vectors and α is a scalar.
- *Level 2 BLAS* are for matrix–vector operations. A typical Level 2 BLAS is of the form $y \equiv \alpha Ax + \beta y$.
- *Level 3 BLAS* are for matrix–matrix operations. A typical Level 3 BLAS is of the form $C \equiv \alpha AB + \beta C$.

Level 1 BLAS is used in LINPACK. Unfortunately, the algorithms composed of Level 1 BLAS are not suitable for achieving high efficiency on most supercomputers of today, such as on the Cray Y-MP or C-90.

Although Level 2 BLAS can give good speed (sometimes almost peak speed) on many vector computers such as CRAY X-MP or CRAY Y-MP, they are not suitable for high efficiency on other modern supercomputers (for example, CRAY 2).

The Level 3 BLAS are ideal for most of today's supercomputers. They can perform $O(n^3)$ floating-point operations on $O(n^2)$ data.

Therefore, during the last several years, an intensive attempt has been made by numerical linear algebraists to restructure the traditional BLAS-1 based algorithms into algorithms rich in BLAS-2 and BLAS-3 operations. As a result, there now exist algorithms of these types, called **blocked algorithms**, for most matrix computations. These algorithms have been implemented in a software package called LAPACK.

According to its user's guide, "LAPACK is a transportable library of Fortran 77 subroutines for solving the most common problems in numerical linear algebra: systems of linear equations, linear least squares problems, eigenvalue problems, and singular value problems. It has been designed to be efficient on a wide range of modern high-performance computers.

"LAPACK is designed to supersede LINPACK and EISPACK, principally by restructuring the software to achieve much greater efficiency on vector processors, high-performance super-scalar workstations, and shared-memory multiprocessors. LAPACK also adds extra functionality, uses some new or improved algorithms, and integrates the two sets of algorithms into a unified package.

"The LAPACK Users' Guide gives an informal introduction to the design of the algorithms and software, summarizes the contents of the package, describes conven-

tions used in the software and its documentation, and includes complete specifications for calling the routines."

4.3.4 NETLIB

Netlib stands for *net*work *lib*rary. LINPACK, EISPACK, and LAPACK subroutines are available electronically from this library, along with many other types of softwares for matrix computations. A user can obtain software from these packages by sending electronic mail to

```
netlib@netlib.ornl.gov
```

Also, files can be transferred to a local directory by anonymous ftp:

```
ftp netlib.att.com
ftp www.netlib.org
```

To find out how to use netlib, enclose the following in the subject header or body of the electronic mail message:

```
send index
```

Information on the subroutines available in a given package can be obtained by sending e-mail:

```
send index from {library}
```

Thus, to obtain a description of subroutines for LAPACK, send the message:

```
send index from LAPACK
```

To obtain a piece or pieces of software from a package, send the message:

```
send {routines} from {library}
```

Thus, to get a subroutine called SGECO from LINPACK, send the message:

```
send SGECO from LINPACK
```

(This message will send you SGECO and other routines that call SGECO.) Xnetlib, which uses an X Window interface for direct downloading, is also available (and more convenient once installed).

Further information about netlib can be obtained by anonymous FTP to either of the following sites:

```
netlib2.cs.utk.edu
netlib.att.com
```

4.3.5 NAG

NAG stands for *N*umerical *A*lgorithm *G*roup. This group has developed a large software library (also called NAG) containing routines for most computational problems including numerical linear algebra problems, numerical differential equations problems (both ordinary and partial), optimization problems, integral equations problems, statistical problems, and so on.

4.3.6 IMSL

IMSL stands for *I*nternational *M*athematical and *S*tatistical *L*ibraries. As the title suggests, this library contains routines for almost all mathematical and statistical computations.

4.3.7 MATLAB

The name MATLAB stands for *MAT*rix *LAB*oratory. It is an interactive computing system designed for easy computations of various matrix-based scientific and engineering problems. MATLAB provides easy access to matrix software developed by the LINPACK and EISPACK software projects.

MATLAB can be used to solve a linear system and associated problems (such as inverting a matrix or computing the rank and determinant of a matrix), to compute the eigenvalues and eigenvectors of a matrix, to find the singular value decomposition of a matrix, to compute the zeros of a polynomial, to compute generalized eigenvalues and eigenvectors, and so on. MATLAB is an extremely useful and valuable package for testing algorithms for small problems and for use in the classroom. It has become an indispensable tool for teaching applied and numerical linear algebra in the classroom. A remarkable feature of MATLAB is its graphic capabilities (see more about MATLAB in Appendix A).

There is a student edition of MATLAB, which was published by Prentice Hall in 1992. It is designed for easy use in the classroom.

4.3.8 MATLAB Codes and MATCOM

MATLAB codes for selected algorithms described in this book are provided for beginning students in Appendix A.

Furthermore, an interactive MATLAB Toolkit called MATCOM, implementing all the major algorithms (to be taught in the first course), can be obtained by the readers of this book, so that students can compare different algorithms for the same problem with respect to numerical efficiency, stability, accuracy, and so on. (MATCOM software is available from MathWorks. See the information sheet at the end of this book. Macintosh and PC versions are available).

4.3.9 The ACM Library

The library provided by the Association for Computing Machinery contains routines for basic matrix–vector operations, linear systems and associated problems, nonlinear systems, zeros of polynomials, and so on. The journal *TOMS* (*ACM Transactions on Mathematical Software*) publishes these algorithms.

4.3.10 ITPACK (Iterative Software Package)

The ITPACK software package is designed to solve the linear system $Ax = b$ by various iterative techniques where A is a large sparse matrix. (The package is available from `math.utexas.edu` via anonymous ftp.)

4.3.11 The Software Package TEMPLATES (for the Solution of Linear Systems: Building Blocks for Iterative Methods)

TEMPLATES contains a large number of programs that serve as building blocks for almost all state-of-the-art iterative methods for solving large and sparse linear systems. A template is a description of a general algorithm rather than the executable object code or the source code more commonly found in a conventional software library.

The software is available from netlib. Send a request by electronic mail as follows:

```
mail netlib@anl.gov
```

Single or multiple requests (one per line) may be made on the subject line or in the body as follows:

```
Send index from linalg.
Send sftemplates.shar from linalg.
```

The first request results in a return e-mail message containing the index from the library linalg, along with a brief description of its contents. The second request results in a return e-mail message consisting of a shar file containing the single precision FORTRAN routines and a README file (Barrett, Berry, Chan et al., 1994).

4.3.12 SPARSKIT: A Basic Toolkit for Sparse Matrix Computations

SPARSKIT was developed by Youcef Saad of the University of Minnesota. SPARSKIT version 2 is now available and is located in an anonymous ftp area in ftp.cs.umn.edu (128.101.230.9). To access it you need to do the following:

- FTP to ftp.cs.umn.edu [128.101.230.9] with the login name 'anonymous' and your name as passwd.
- Then cd to /pub/sparse. You will find a file SPARSKIT2.tar.Z and a directory called SPARSKIT2. You can either retrieve the compressed tar file SPARSKIT2.tar.Z (use 'binary' before 'get' when FTP'ing) or you can retrieve specific files/modules, or all of the directory SPARSKIT2.
- If you retrieve the compressed file SPARSKIT2.tar.Z you need to type the following in order to create the subdirectory SPARSKIT2 in your directory:

```
uncompress -c SPARSKIT2.tar.Z | tar xvf -
```

4.4 Review and Summary

The purpose of this chapter has been to introduce concepts of numerically effective algorithms and the associated high-quality mathematical software.

1. Writing codes for a given algorithm can be a trivial task. However, all software written is not high-quality software. We defined *high quality mathematical software* as software that is (1) powerful and flexible, (2) easy to read and modify, (3) portable, (4) robust, and, more importantly, (5) based on a numerically effective algorithm.
2. As with software, there may exist many algorithms for a given problem. However, not all algorithms are numerically effective. We have defined a *numerically effective algorithm* as one that is (1) general purpose, (2) reliable, (3) stable, and (4) efficient in terms of both time and storage.
3. The efficiency of a matrix algorithm is measured by computer time consumed by the algorithm. A theoretical measure is the number of *flops* required to execute the algorithm. Roughly, one multiplication (or a division) together with one addition (or a subtraction) has been defined to be 1 flop.

   A matrix algorithm involving matrices of order n requiring no more than $O(n^3)$ flops was defined to be an *efficient algorithm*; stability of an algorithm was defined in Chapter 3. *An algorithm may be efficient without being stable*. Thus, an efficient algorithm may not necessarily be a numerically effective algorithm. There are algorithms that are fast but not stable.
4. Several examples (Section 4.2) were provided to show how certain basic matrix operations can be reorganized rather easily to make them both storage- and time-efficient without implementing them naively. Example 4.2.7 is the most important one in this context. Here we showed how to organize the computation of the product of an $n \times n$ matrix A with the matrix $H = I - 2uu^T/u^Tu$, known as the *Householder matrix*, so that the product can be computed with $O(n^2)$ flops rather than the $O(n^3)$ flops that would be needed if computed naively, ignoring the structure of H. This computation forms a basis for many other matrix computations described later in the book.
5. A summary of high-quality mathematical software packages such as LINPACK, EISPACK, LAPACK, MATLAB, IMSL, NAG, TEMPLATES, and SPARSKIT was provided in the final section.

EXERCISES

1. Develop an algorithm to compute the product $C = AB$ in each of the following cases. Your algorithm should take advantage of the special structure of the matrices in each case. Give flop count and storage requirements in each case.
   (a) A and B are both lower triangular matrices.
   (b) A is arbitrary and B is lower triangular.
   (c) A and B are both tridiagonal.
   (d) A is arbitrary and B is upper Hessenberg.
   (e) A is upper Hessenberg and B is tridiagonal.
   (f) A is upper Hessenberg and B is upper triangular.
2. A square matrix $A = (a_{ij})$ is said to be a band matrix of bandwidth $2k + 1$ if

   $$a_{ij} = 0 \qquad \text{whenever } |i - j| > k$$

   Develop an algorithm to compute the product $C = AB$, where A is arbitrary and B is a band matrix of bandwidth 3, taking advantage of the structure of the matrix B. Give flop count.

3. Using the ideas in Algorithm 4.2.1, develop an algorithm to compute the product $A(I + xy^T)$, where A is an $n \times n$ matrix and x and y are n-vectors. Your algorithm should require roughly $2n^2$ flops.
4. Rewrite the algorithm of Exercise 3 in the special cases when the matrix $I + xy^T$ is
   (a) An elementary matrix: $I + m_k e_k^T$, $m_k = (0, 0, \ldots, 0, m_{k+1,k}, \ldots, m_{n,k})^T$, and e_k^T is the kth row of I.
   (b) A Householder matrix: $I - \dfrac{2uu^T}{u^T u}$, where u is an n-vector.
5. Let A and B be two symmetric matrices of the same order. Develop an algorithm to compute $C = A + B$, taking advantage of symmetry for each matrix. Your algorithm should overwrite B with C. What is the flop count?
6. Let $A = (a_{ij})$ be an unreduced lower Hessenberg matrix of order n. Then, given the first row r_1, it can be shown (Datta and Datta, 1976) that the successive rows r_2 through r_n of A^k (k is a positive integer $\leq n$) can be computed recursively as follows:

$$r_{i+1} = \frac{\left(r_i B_i - \sum_{j=1}^{i-1} a_{ij} r_j\right)}{a_{i,i+1}}, \qquad i = 1, 2, \ldots, n-1, \quad \text{where } B_i = A - a_{ii} I$$

   Develop an algorithm to compute A^k. Give flop count for the algorithm.
7. Let a_r and b_r denote, respectively, the rth columns and the rth row of the matrices A and B. Then develop an algorithm to compute the product AB from the formula

$$AB = \sum_{i=1}^{n} a_i b_i$$

   Give flop count and storage requirement of the algorithm.
8. Consider the 12×12 matrix

$$A = \begin{pmatrix} 12 & 11 & 10 & \cdots & 3 & 2 & 1 \\ 11 & 11 & 10 & \cdots & 3 & 2 & 1 \\ 0 & 10 & 10 & \cdots & 3 & 2 & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & 0 & \cdots & 2 & 2 & 1 \\ 0 & 0 & 0 & \cdots & 0 & 1 & 1 \end{pmatrix}$$

   Find the eigenvalues of this matrix. Use MATLAB.

   Now perturb the (12,1) element to 10^{-9} and compute the eigenvalues of this perturbed matrix. What do you conclude about the conditioning of the eigenvalues?

MATLAB AND MATCOM PROGRAMS AND PROBLEMS

Note: You will need the program **phousmul** from MATCOM.

1. Using the MATLAB function '**rand**', create a 5×5 random matrix and then print out the following outputs:

   A(2, :), A(:, 1), A(:, 5),
   A(1, 1: 2 : 5), A([1, 5]), A(4: − 1: 1, 5: -1: 1).
2. Using the function 'for', write a MATLAB program to find the inner product and outer product of two n-vectors u and v.

   [s] = **inpro**(u, v)
   [A] = **outpro**(u, v)

   Test your program by creating two different vectors u and v using rand(4, 1).
3. Learn how to use the following MATLAB commands to create special matrices:

compan	companion matrix
diag	diagonal matrix
ones	constant matrix
zeros	zero matrix
rand	random matrix
tril	lower triangular part
hankel	Hankel matrix
toeplitz	Toeplitz matrix
hilb	Hilbert matrix
triu	upper triangular
vander	Vandermonde matrix

4. Write MATLAB programs to create the following well-known matrices:
   (a) [A] = **wilk**(n) to create the Wilkinson bidiagonal matrix $A = (a_{ij})$ of order n:

$$\begin{aligned} a_{ii} &= n - i + 1, & i &= 1, 2, \ldots, n \\ a_{i-1,i} &= n, & i &= 2, 3, \ldots, n \\ a_{ij} &= 0, & &\text{otherwise} \end{aligned}$$

   (b) [A] = **pie**(n, α) to create the Pie matrix $A = (a_{ij})$ of order n:

$$\begin{aligned} a_{ii} &= \alpha, & &\alpha \text{ a parameter near 1 or } n-1 \\ a_{ij} &= 1 & &\text{for } i \neq j \end{aligned}$$

5. Using "help" commands for "flops," "clock," "etime," and so on, learn how to measure flop count and timing for an algorithm.
6. Using MATLAB functions 'for', 'size', and 'zero', write a MATLAB program to find the product of two upper triangular matrices A and B of order $m \times n$ and $n \times p$, respectively. Test your program using

   A = **triu**(rand(4, 3))
   B = **triu**(rand(3, 3))
7. Run the MATLAB program **phousmul**(A, u) from MATCOM by creating a random matrix A of order 6×3 and a random vector u with six elements. Print the output and the number of flops and elapsed time.
8. Modify the MATLAB program **phousmul** to **housxy**(A, x, y) to compute the product $(I + xy^T)A$. Test your program by creating a 15×15 random matrix A and the vectors x and y of appropriate dimensions. Print the product and the number of flops and elapsed time.

Suggestions for Further Reading

A clear and nice description of how to develop high-quality mathematical software packages is included in the book *Matrix Computations and Mathematical Software* by John Rice (New York: McGraw-Hill, 1981). This book contains a chapter (Chapter 12) on software projects. Students interested in comparative studies of various software may find them interesting. An excellent paper by Demmel (1984) on the reliability of numerical software is worth reading.

Another recent book in the area is *Handbook of Matrix Computations* by T. Coleman and C. Van Loan (Philadelphia: SIAM, 1988). The books by Rice and Coleman and Van Loan are a must for readers interested in software-development for matrix problems.

The books by Forsythe, Malcolm, and Moler (1977) and by Hager (1988) contain some useful subroutines for matrix computations. See also the books by Johnston and by Kahaner, Moler, and Nash in Chapter 2 Further Reading.

Stewart's book (1973) is a valuable source for learning how to organize and develop algorithms for basic matrix operations in time-efficient and storage-economic ways.

Each software package has its own users' manual that describes in detail the functions of the subroutines, how to use them, and so on. We list the most important ones.

1. *LINPACK Users' Guide*, by J. Dongarra, J. Bunch, C. Moler, and G. W. Stewart, SIAM, Philadelphia, PA, 1979.
2. *EISPACK Users' Guide* can be obtained from either NESC or IMSL. *Matrix Eigensystem Routines EISPACK Guide*, by B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow, Y. Ikebe, V. C. Klema, and C. B. Moler, has been published by Springer-Verlag, Berlin, 1976, as volume 6 of *Lecture Notes in Computer Science*. A later extension, prepared by Garbow, Boyle, Dongarra, and Moler in 1977, is also available as a Springer-Verlag publication.
3. *LAPACK Users' Guide*, prepared by E. Anderson et al., is available from SIAM. SIAM's address and telephone number are: SIAM (Society for Industrial and Applied Mathematics), 3600 University City Science Center, Philadelphia, PA 19114-2688; (215) 382-9800.
4. The NAG Library and the associated users' manual can be obtained from:

   Numerical Algorithms Group, Inc.
   1400 Opus Place, Suite 200
   Downers Grove, IL 60515-5702
   Tel: (708) 971-2395

5. IMSL: The IMSL software library and documentation are available from IMSL, 2500 Parkwest Tower One, Houston, Texas 77042-3020; (713) 782-6060.
6. MATLAB: The MATLAB software and Users' Guide are available from:

   The MathWorks, Inc.
   Cochituate Place
   24 Prime Park Way
   Natick, MA 01760-1520
   TEL: (508) 653-1415; FAX: (508) 653-2997
   e-mail: info@mathworks.com

   The student edition of MATLAB has been published by Prentice-Hall, Englewood Cliffs, NJ, 07632. A $5\frac{1}{4}$-in. disk is included with the book for MS-DOS personal computers.
7. The book *TEMPLATES for the Solution of Linear Systems: Building Blocks for Iterative Methods* by R. Barrett, M. Berry, T. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst, is available from SIAM, Philadelphia, or in Postscript format over the Internet. To retrieve the postscript file you can use one of the following methods:
   - anonymous ftp to netlib2.cs.utk.edu
     cd linalg
     get templates.ps
     quit
   - send e-mail to netlib@ornl.gov and in the message type:
     `send templates.ps from linalg`

For more information on accessing mathematical software electronically, the paper "Distribution of Mathematical Software via Electronic Mail" by J. Dongarra and E. Grosse, *Communications of the ACM* 30,5 (1987): 403–7, is worth reading.

Finally, a nice survey of the blocked algorithms has been given by Demmel, Heath, and Van der Vorst: *Parallel Numerical Linear Algebra*. (Acta Numerica, vol. 2, Cambridge, MA: University Press, 1993.)

CHAPTER FIVE

Some Useful Transformations in Numerical Linear Algebra and Their Applications

The major objectives of this chapter are to introduce three fundamental tools—elementary, Householder, and Givens matrices—and show how these matrices can be used to achieve important matrix factorizations such as LU and its variants, QR, and reduction to a Hessenberg form of a matrix. Applications of the QR factorization to constructing orthonormal bases and orthogonal projections, and updating QR factorization of a matrix are also briefly mentioned.

The following background material and tools developed earlier will be required for smooth comprehension of this chapter's material: range and null space (Section 1.3.2), rank properties (Section 1.3.3), orthogonality and projections (Sections 1.3.6 and 1.3.7), special matrices (Section 1.4), basic Gaussian elimination algorithm (Algorithm 3.1.5), and stability and conditioning concepts and results (Sections 3.2–3.4).

5.1 A Computational Methodology in Numerical Linear Algebra

Most computational algorithms to be presented in this book have a common basic structure that can be described as follows:

1. The problem is first transformed to a "reduced" problem.
2. The reduced problem is then solved, exploiting the special structure exhibited by the problem.
3. Finally, the solution of the original problem is recovered from the solution of the reduced problem.

The reduced problem typically involves a "condensed" form or forms of the matrix A, such as triangular, Hessenberg (almost triangular), tridiagonal, real Schur form (quasitriangular), or bidiagonal. It is the structures of these condensed forms that are exploited in the solution of the reduced problem. For example, as we have

seen in Chapter 3, the solution of the linear system $Ax = b$ is usually obtained by first triangularizing the matrix A and then solving an equivalent triangular system. In eigenvalue computations, the matrix A is transformed to a Hessenberg form before applying the QR iterations. To compute the *singular-value decomposition*, the matrix A is first transformed to a bidiagonal matrix, and then singular values of the bidiagonal matrix are computed. These condensed forms are normally achieved through a series of transformations known as *elementary*, *Householder*, or *Givens transformations*. We study these transformations here and show how they can be applied to achieve various condensed forms.

5.2 Elementary Matrices and LU Factorization

In this section we show how to triangularize a matrix A using the classical elimination scheme known as the Gaussian elimination scheme. The tools of Gaussian elimination are elementary matrices.

DEFINITION 5.2.1 An elementary lower triangular matrix of order n is a matrix of the form

$$E = \begin{pmatrix} 1 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & & \vdots & \vdots \\ \vdots & \vdots & & 0 & 1 & \ddots & & \vdots \\ 0 & 0 & 0 & \cdots & m_{k+1,k} & \ddots & 0 & 0 \\ 0 & 0 & 0 & \cdots & \vdots & \ddots & \ddots & 0 \\ 0 & 0 & 0 & \cdots & m_{n,k} & \cdots & 0 & 1 \end{pmatrix}$$

Thus, it is an identity matrix except possibly for a few nonzero elements below the diagonal of a single column. If the nonzero elements lie in the kth column as shown above, then E has the form

$$E = I + m_k e_k^T$$

where I is the identity matrix of order n, $m_k = (0, 0, \ldots, 0, m_{k+1,k}, \ldots, m_{n,k})^T$, and e_k is the kth unit vector. Note that $e_i^T m_k = 0$, $i = 1, 2, \ldots, k$.

Elementary matrices can be very conveniently used to create zeros in a vector, as shown in the following lemma.

LEMMA 5.2.1 Given

$$a = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix}, \qquad a_1 \neq 0$$

there is an elementary matrix E such that Ea is a multiple of e_1.

Math. Seminar

Proof: Define

$$E = \begin{pmatrix} 1 & 0 & 0 & \cdots & \cdots & 0 \\ -\frac{a_2}{a_1} & 1 & 0 & \cdots & \cdots & 0 \\ \vdots & 0 & \ddots & & & 0 \\ \vdots & \vdots & \ddots & \ddots & & \vdots \\ \vdots & \vdots & & \ddots & \ddots & \vdots \\ -\frac{a_n}{a_1} & 0 & 0 & \cdots & 0 & 1 \end{pmatrix}$$

Then E is an elementary lower triangular matrix such that

$$Ea = \begin{pmatrix} a_1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

■

DEFINITION 5.2.2 The elements $m_{i1} = -a_i/a_1$, $i = 2, \ldots, n$, are called **multipliers.**

5.2.1 Gaussian Elimination without Pivoting

We described the basic Gaussian elimination scheme (Gaussian elimination without pivoting) in Chapter 3 (Algorithm 3.1.5). We will see in this section that this process yields an LU factorization of A, $A = LU$, whenever the process can be carried to completion. *The key observation is that the matrix $A^{(k)}$ is a result of premultiplication of $A^{(k-1)}$ by a suitable elementary lower triangular matrix.*

Set $A = A^{(0)}$.

Step 1: Find an elementary matrix E_1 such that $A^{(1)} = E_1 A$ has zeros below the (1, 1) entry in the first column. That is, $A^{(1)}$ has the form

$$A^{(1)} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ 0 & a_{22}^{(1)} & \cdots & a_{2n}^{(1)} \\ \vdots & \vdots & & \vdots \\ 0 & a_{n2}^{(1)} & \cdots & a_{nn}^{(1)} \end{pmatrix}$$

Note that it is sufficient only to find E_1 such that

$$E_1 \begin{pmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{n1} \end{pmatrix} = \begin{pmatrix} a_{11} \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

Then $A^{(1)} = E_1 A$ will have the preceding form and is the same as the matrix $A^{(1)}$ obtained at the end of step 1 of Algorithm 3.1.5.

Record the multipliers: $m_{21}, m_{31}, \ldots, m_{n1}$; $m_{i1} = -a_{i1}/a_{11}$, $i = 2, \ldots, n$.

Step 2: Find an elementary matrix E_2 such that $A^{(2)} = E_2 A^{(1)}$ has zeros below the (2, 2) entry in the second column. The matrix E_2 can be constructed as follows.

First, find an elementary matrix $\widehat{E}_2$ of order $(n-1)$ such that

$$\widehat{E}_2 \begin{pmatrix} a_{22}^{(1)} \\ a_{32}^{(1)} \\ \vdots \\ \vdots \\ a_{n2}^{(1)} \end{pmatrix} = \begin{pmatrix} a_{22}^{(1)} \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

Record the multipliers: $m_{32}, \ldots, m_{n2}$; $m_{i2} = -a_{i2}^{(1)}/a_{22}^{(1)}$, $i = 3, \ldots, n$. Then define

$$E_2 = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & & & \\ \vdots & & \widehat{E}_2 & \\ 0 & & & \end{pmatrix} = \left(\begin{array}{c|c} 1 & 0 \\ \hline 0 & \widehat{E}_2 \end{array}\right)$$

$A^{(2)} = E_2 A^{(1)}$ will then have zeros below the (2, 2) entry in the second column.

$$A^{(2)} = \begin{pmatrix} a_{11} & a_{12} & \cdots & \cdots & a_{1n} \\ 0 & a_{22}^{(1)} & \cdots & \cdots & a_{2n}^{(1)} \\ 0 & 0 & a_{33}^{(2)} & \cdots & a_{3n}^{(2)} \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & a_{n3}^{(2)} & \cdots & a_{nn}^{(2)} \end{pmatrix}$$

Note that premultiplication of $A^{(1)}$ by E_2 does not destroy zeros already created in $A^{(1)}$. This matrix $A^{(2)}$ is the same as the matrix $A^{(2)}$ of Algorithm 3.1.5.

Step k: In general, at the kth step, an elementary matrix E_k is found such that $A^{(k)} = E_k A^{(k-1)}$ has zeros below the (k, k) entry in the kth column. E_k is computed in two successive steps. First, an elementary matrix $\widehat{E}_k$ of order $n - k + 1$ is constructed such that

$$\widehat{E}_k \begin{pmatrix} a_{k,k}^{(k-1)} \\ a_{k+1,k}^{(k-1)} \\ \vdots \\ a_{nk}^{(k-1)} \end{pmatrix} = \begin{pmatrix} a_{kk}^{(k-1)} \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

and then E_k is defined as

$$E_k = \begin{pmatrix} I_{k-1} & & 0 \\ & \widehat{E}_k & \\ 0 & & \end{pmatrix}$$

Here I_{k-1} is the matrix of the first $(k-1)$ rows and columns of the $n \times n$ identity matrix I. The matrix $A^{(k)} = E_k A^{(k-1)}$ is the same as the matrix $A^{(k)}$ of Algorithm 3.1.5.

Record the multipliers:

$$m_{k+1,k}, \ldots, m_{ik}; \quad m_{i,k} = -\frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}}, \qquad i = k+1, \ldots, n$$

Step $n-1$: At the end of the $(n-1)$st step, the matrix $A^{(n-1)}$ is upper triangular and the same matrix $A^{(n-1)}$ of Algorithm 3.1.5.

$$A^{(n-1)} = \begin{pmatrix} a_{11} & a_{12} & \cdots & \cdots & \cdots & a_{1n} \\ 0 & a_{22}^{(1)} & \cdots & \cdots & \cdots & a_{2n}^{(1)} \\ 0 & 0 & a_{33}^{(2)} & \cdots & \cdots & a_{3n}^{(2)} \\ \vdots & \vdots & \ddots & \ddots & & \vdots \\ \vdots & \vdots & & \ddots & \ddots & \\ 0 & 0 & \cdots & \cdots & 0 & a_{nn}^{(n-1)} \end{pmatrix}$$

Obtain L and U:

$$\begin{aligned} A^{(n-1)} &= E_{n-1}A^{(n-2)} = E_{n-1}E_{n-2}A^{(n-3)} = \cdots \\ &= E_{n-1}E_{n-2}E_{n-3}\cdots E_2E_1A \end{aligned}$$

Set

$$U = A^{(n-1)}, \qquad L_1 = E_{n-1}E_{n-2}\cdots E_2E_1$$

Then from before we have

$$U = L_1A$$

Because each E_k is a unit lower triangular matrix (a lower triangular matrix having 1s along the diagonal), so is the matrix L_1; therefore, L_1^{-1} exists. (Note that the product of two triangular matrices of one type is a triangular matrix of the same type.)

Set $L = L_1^{-1}$. Then the equation $U = L_1A$ becomes

$$A = LU$$

This factorization of A is known as **LU factorization.**

DEFINITION 5.2.3 The entries $a_{11}, a_{22}^{(1)}, \ldots, a_{nn}^{(n-1)}$ are called **pivots**, and the preceding process of obtaining LU factorization is known as **Gaussian elimination without row interchanges**. It is commonly known as **Gaussian elimination without pivoting**. The process is named after the German mathematician and astronomer Karl Friedrich Gauss (1777–1855).

An Explicit Expression for L

Because $L = L_1^{-1} = E_1^{-1}E_2^{-1}\cdots E_{n-1}^{-1}$ and $E_i^{-1} = I - m_ie_i^T$, where

$$m_i = (0, 0, \ldots, 0, m_{i+1,i}, \ldots, m_{n,i})^T$$

we have

$$L = \begin{pmatrix} 1 & 0 & \cdots & \cdots & 0 \\ -m_{21} & 1 & 0 & & 0 \\ -m_{31} & -m_{32} & 1 & \ddots & 0 \\ \vdots & \vdots & \ddots & \ddots & 0 \\ -m_{n1} & -m_{n2} & \cdots & -m_{n,n-1} & 1 \end{pmatrix}$$

Thus, L can be formed without finding a matrix inverse.

Existence and Uniqueness of LU Factorization

It is very important to note that for an LU factorization to exist, the pivots must be different from zero. Thus, LU factorization may not exist even for a very simple matrix. Take

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

It can be shown (Exercise 3) that if A_r is the submatrix of A consisting of the first r rows and columns, then

$$\det(A_r) = a_{11}a_{22}^{(1)}\cdots a_{rr}^{(r-1)}$$

Thus, if $\det(A_r)$, $r = 1, 2, \ldots, n$ is nonzero, then an LU factorization always exists. Indeed, the LU factorization in this case is unique, for if

$$A = L_1U_1 = L_2U_2$$

then, since L_1 and L_2 and the matrix A are nonsingular, from

$$\det(A) = \det(L_1U_1) = \det(L_1)\cdot\det(U_1)$$

and

$$\det(A) = \det(L_2U_2) = \det(L_2)\cdot\det(U_2)$$

it follows that U_1 and U_2 are also nonsingular. Hence

$$L_2^{-1}L_1 = U_2U_1^{-1}$$

$L_2L_1^{-1}$ is the product of two unit lower triangular matrices and is, therefore, unit lower triangular; $U_2U_1^{-1}$ is the product of two upper triangular matrices and is, therefore, upper triangular. Since a unit lower triangular matrix can be equal to an upper triangular matrix only if both are the identity, we have

$$L_1 = L_2, \qquad U_1 = U_2$$

THEOREM 5.2.1 LU Theorem Let A be an $n \times n$ matrix with all nonzero leading principal minors. Then A has a unique LU factorization:

$$A = LU$$

where L is unit lower triangular and U is upper triangular.

Remark: Note that in Theorem 5.2.1, if the diagonal entries of L are not specified, then the factorization is not unique.

Example 5.2.1 Find an LU factorization of

$$A = \begin{pmatrix} 2 & 2 & 3 \\ 4 & 5 & 6 \\ 1 & 2 & 4 \end{pmatrix}$$

Step 1: Compute E_1. The multipliers are: $m_{21} = -2, m_{31} = -\frac{1}{2}$.

$$E_1 = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -\frac{1}{2} & 0 & 1 \end{pmatrix}$$

$$A^{(1)} = E_1 A = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -\frac{1}{2} & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 2 & 3 \\ 4 & 5 & 6 \\ 1 & 2 & 4 \end{pmatrix} = \begin{pmatrix} 2 & 2 & 3 \\ 0 & 1 & 0 \\ 0 & 1 & \frac{5}{2} \end{pmatrix}$$

Step 2: Compute E_2. The multiplier is $m_{32} = -1$.

$$\widehat{E}_2 = \begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix}$$

$$E_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & \widehat{E}_2 \end{pmatrix}$$

$$A^{(2)} = E_2 A^{(1)} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} 2 & 2 & 3 \\ 0 & 1 & 0 \\ 0 & 1 & \frac{5}{2} \end{pmatrix} = \begin{pmatrix} 2 & 2 & 3 \\ 0 & 1 & 0 \\ 0 & 0 & \frac{5}{2} \end{pmatrix}$$

Thus

$$U = \begin{pmatrix} 2 & 2 & 3 \\ 0 & 1 & 0 \\ 0 & 0 & \frac{5}{2} \end{pmatrix}$$

Compute L.

$$L_1 = E_2 E_1 = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ \frac{3}{2} & -1 & 1 \end{pmatrix}$$

$$L = L_1^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ -m_{21} & 1 & 0 \\ -m_{31} & -m_{32} & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ \frac{1}{2} & 1 & 1 \end{pmatrix}$$

(Note that neither L_1 nor its inverse needs to be computed explicitly.)

Forming the Matrix *L* and Other Computational Details

1. Each elementary matrix E_k is uniquely determined by $(n-k)$ multipliers

$$m_{k+1,k}, \ldots, m_{nk}$$

Thus to construct E_k, it is sufficient to save these multipliers.

2. The premultiplication of $A^{(k-1)}$ by E_k affects only rows $(k+1)$ through n. The first k rows of the product remain the same as those of $A^{(k-1)}$, and only the last $(n-k)$ rows are modified.

   Thus, if $E_k A^{(k-1)} = A^{(k)} = (a_{ij}^{(k)})$, then

   (a) $a_{ij}^{(k)} = a_{ij}^{(k-1)}$ $(i = 1, 2, \ldots, k; j = 1, 2, \ldots, n)$ (the first k rows);

   (b) $a_{ij}^{(k)} = a_{ij}^{(k-1)} + m_{ik} a_{kj}^{(k-1)}$ $(i = k+1, \ldots, n; j = k+1, \ldots, n)$ (the last $(n-k)$ rows).

   (Note that this is how the entries of $A^{(k)}$ were obtained from those of $A^{(k-1)}$ in Algorithm 3.1.5.)

   For example, let $n = 3$.

$$E_1 A = A^{(1)} = (a_{ij}^{(1)})$$

$$E_1 = \begin{pmatrix} 1 & 0 & 0 \\ m_{21} & 1 & 0 \\ m_{31} & 0 & 1 \end{pmatrix}$$

$$E_1 A = \begin{pmatrix} 1 & 0 & 0 \\ m_{21} & 1 & 0 \\ m_{31} & 0 & 1 \end{pmatrix} \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22}^{(1)} & a_{23}^{(1)} \\ 0 & a_{32}^{(1)} & a_{33}^{(1)} \end{pmatrix}$$

   where

$$a_{22}^{(1)} = a_{22} + m_{21} a_{12}$$
$$a_{23}^{(1)} = a_{23} + m_{21} a_{13}$$

   and so on.

3. As soon as $A^{(k)}$ is formed, it can overwrite A.
4. The vector $(m_{k+1,k}, \ldots, m_{nk})$ has $(n-k)$ components, and at the kth step exactly $(n-k)$ zeros are produced; an obvious scheme of storage will then be to store these $(n-k)$ elements in the positions $(k+1, k), (k+2, k), \ldots, (n, k)$ of A below the diagonal.
5. The entries of the upper triangular matrix U then can be stored in the upper half of A, including the diagonal. With this storage scheme, the matrix $A^{(n-1)}$ at the end of the $(n-1)$st step will be

$$A \equiv A^{(n-1)} = \begin{pmatrix} a_{11} & a_{12} & \cdots & \cdots & a_{1n} \\ m_{21} & a_{22} & \cdots & \cdots & a_{2n} \\ m_{31} & m_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ m_{n1} & m_{n2} & \cdots & m_{n,n-1} & a_{nn} \end{pmatrix}$$

Thus a typical kth step of Gaussian elimination for LU factorization consists of

1. Forming the multipliers and storing them in the appropriate places below diagonal, and
2. Updating the entries of the rows $(k+1)$ through n and saving them in the upper half of A.

Based on the preceding discussion, we now present the following algorithm.

ALGORITHM 5.2.1 Triangularization Using Gaussian Elimination without Pivoting Let A be an $n \times n$ matrix. The following algorithm computes triangularization of A whenever it exists. The algorithm overwrites the upper triangular part of A including the diagonal with U, and the entries of A below the diagonal are overwritten with multipliers needed to compute L.

For $k = 1, 2, \ldots, (n-1)$ do

1: Form the multipliers

$$a_{ik} \equiv m_{ik} = -\frac{a_{ik}}{a_{kk}} \quad (i = k+1, k+2, \ldots, n)$$

2: Update the entries

$$a_{ij} \equiv a_{ij} + m_{ik}a_{kj} \quad (i = k+1, \ldots, n;\ j = k+1, \ldots, n)$$

Remark: The algorithm does not give the matrix L explicitly; however, it can be formed out of the multipliers saved at each step, as shown earlier (see the explicit expression for L).

MATCOM Note: Algorithm 5.2.1 has been implemented in MATCOM program LUGSEL. L and U have been explicitly computed.

Flop count. The algorithm requires roughly $n^3/3$ flops. This can be seen as follows: At step 1, we compute $n-1$ multipliers and update $(n-1)^2$ entries of A. Each multiplier requires 1 flop and updating each entry also requires 1 flop. Thus, step 1 requires $[(n-1)^2 + (n-1)]$ flops.

Similarly, step 2 requires $[(n-2)^2 + (n-2)]$ flops, and so on. In general, step k requires $[(n-k)^2 + (n-k)]$ flops. Because there are $(n-1)$ steps, we have

$$\begin{aligned}\text{Total flops} &= \sum_{k=1}^{n-1}(n-k)^2 + \sum_{k=1}^{n-1}(n-k)\\ &= \frac{n(n-1)(2n-1)}{6} + \frac{n(n-1)}{2}\\ &\cong \left[\frac{n^3}{3} + O(n^2)\right]\end{aligned}$$

Gaussian Elimination for a Rectangular Matrix

The Gaussian elimination process for an $n \times n$ matrix A just described can easily be extended to an $m \times n$ matrix to compute its LU factorization, when it exists. The process is identical, only the number of steps in this case is $k = \min\{m-1, n\}$. The following is an illustrative example. Let

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$$

$$m = 3, \qquad n = 2$$

The number of steps is $k = \min(2, 2) = 2$.

Step 1: The multipliers are $m_{21} = -3$, $m_{31} = -5$.

$$a_{22} \equiv a_{22}^{(1)} = a_{22} + m_{21}a_{12} = -2$$
$$a_{32} \equiv a_{32}^{(1)} = a_{32} + m_{31}a_{13} = -4$$

$$A \equiv A^{(1)} = \begin{pmatrix} 1 & 2 \\ 0 & -2 \\ 0 & -4 \end{pmatrix}$$

Step 2: The multiplier is $m_{32} = -2$, $a_{32} \equiv a_{32}^{(2)} = 0$.

$$A \equiv A^{(2)} = \begin{pmatrix} 1 & 2 \\ 0 & -2 \\ 0 & 0 \end{pmatrix}$$

$$U = \begin{pmatrix} 1 & 2 \\ 0 & -2 \\ 0 & 0 \end{pmatrix}$$

Note that U in this case is *upper trapezoidal* rather than an upper triangular matrix:

$$L = \begin{pmatrix} 1 & 0 & 0 \\ -m_{21} & 1 & 0 \\ -m_{31} & -m_{32} & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 3 & 1 & 0 \\ 5 & 2 & 1 \end{pmatrix}$$

Verify that

$$LU = \begin{pmatrix} 1 & 0 & 0 \\ 3 & 1 & 0 \\ 5 & 2 & 1 \end{pmatrix}\begin{pmatrix} 1 & 2 \\ 0 & -2 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix} = A$$

Difficulties with Gaussian Elimination without Pivoting

As we have seen before, Gaussian elimination without pivoting fails if any of the pivots is zero. However, it is worse yet if any pivot becomes close to zero: In this case the method can be carried to completion, but the obtained results may be totally wrong.

Consider the following celebrated example from Forsythe and Moler (1967, 34): Let Gaussian elimination without pivoting be applied to

$$A = \begin{pmatrix} 0.0001 & 1 \\ 1 & 1 \end{pmatrix}$$

There is only one step. Multiplier $m_{21} = -1/10^{-4} = -10^4$.

$$U = A^{(1)} = \begin{pmatrix} 0.0001 & 1 \\ 0 & -10^4 \end{pmatrix}$$

$$L = \begin{pmatrix} 1 & 0 \\ 10^4 & 1 \end{pmatrix}.$$

The product of the computed L and U gives

$$LU = \begin{pmatrix} 0.0001 & 1 \\ 1 & 0 \end{pmatrix}$$

which is different from A. Who is to blame?

Note that the pivot $a_{11}^{(1)} = 0.0001$ is very close to zero. This small pivot gave a large multiplier. The large multiplier, when used to update the entries, eliminated the small entries (for example, $(1 - 10^4)$ gave -10^4). Fortunately, *we can avoid this small pivot just by row interchanges*. Consider the matrix with the first row written second and the second written first:

$$A' = \begin{pmatrix} 1 & 1 \\ 0.0001 & 1 \end{pmatrix}$$

Gaussian elimination now gives

$$U = A^{(1)} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \quad L = \begin{pmatrix} 1 & 0 \\ 0.0001 & 1 \end{pmatrix}$$

Note that the pivot in this case is $a_{11}^{(1)} = 1$. The product

$$LU = \begin{pmatrix} 1 & 1 \\ 0.0001 & 1.0001 \end{pmatrix} = A'$$

5.2.2 Gaussian Elimination with Partial Pivoting

In the preceding example, we found a factorization of the matrix A' which is a permuted version of A in the sense that the rows have been swapped. A primary purpose of factorizing a matrix A into LU is to use this factorization to solve a linear system. It is easy to see that the solution of the system $Ax = b$ and that of the system $A'x = b'$, where b' has been obtained in a manner similar to that used to generate A', are the same. Thus, if the row interchanges can help avoid a small pivot, it is certainly desirable to do so.

As this example suggests, disaster in Gaussian elimination without pivoting can perhaps be avoided by identifying a "good pivot" (a pivot as large as possible) at each step, before the process of elimination is applied. The good pivot may be located among the entries in a column or among all the entries in a submatrix of the current matrix. In the former case, since the search is only partial, the method is called **partial pivoting**; in the latter case, the method is called **complete pivoting**. *It is important to note that the purpose of pivoting is to prevent large growth in the reduced matrices, which can wipe out original data.* One way to do this is to keep multipliers less than 1 in magnitude, and this is exactly what is accomplished by pivoting. However, large multipliers do not necessarily mean instability (see our discussion of Gaussian elimination without pivoting for symmetric positive definite matrices in Chapter 6). We first describe Gaussian elimination with partial pivoting.

The process consists of $n - 1$ steps.

Step 1: Scan the first column of A to identify the largest element in magnitude in that column. Let it be $a_{r_1,1}$.

- Form a permutation matrix P_1 by interchanging the rows 1 and r_1 of the identity matrix and leaving the other rows unchanged.
- Form P_1A by interchanging the rows r_1 and 1 of A.
- Find an elementary lower triangular matrix M_1 such that $A^{(1)} = M_1P_1A$ has zeros below the (1, 1) entry on the first column.

It is sufficient to construct M_1 such that

$$M_1 \begin{pmatrix} a_{11} \\ \vdots \\ a_{n1} \end{pmatrix} = \begin{pmatrix} * \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

Note that

$$M_1 = \begin{pmatrix} 1 & 0 & 0 & \cdots & \cdots & 0 \\ m_{21} & 1 & 0 & \cdots & \cdots & 0 \\ m_{31} & 0 & 1 & \cdots & \cdots & 0 \\ \vdots & \vdots & 0 & \ddots & & \vdots \\ \vdots & \vdots & \vdots & & \ddots & 0 \\ m_{n1} & 0 & 0 & \cdots & 0 & 1 \end{pmatrix}$$

where $m_{21} = -a_{21}/a_{11}$, $m_{31} = -a_{31}/a_{11}, \ldots, m_{n1} = -a_{n1}/a_{11}$, *and* a_{ij} *refers to the* (i, j)*th entry of the permuted matrix* P_1A. Save the multipliers m_{i1}, $i = 2, \ldots, n$ and record the row interchanges.

$$A^{(1)} = \begin{pmatrix} * & * & \cdots & * \\ 0 & * & \cdots & * \\ 0 & * & \cdots & * \\ \vdots & \vdots & & \vdots \\ 0 & * & \cdots & * \end{pmatrix}$$

Step 2: Scan the second column of $A^{(1)}$ below the first row to identify the largest element in magnitude in that column. Let the element be $a_{r_2,2}^{(1)}$. Form the permutation matrix P_2 by interchanging the rows 2 and r_2 of the identity matrix and leaving the other rows unchanged. Form $P_2A^{(1)}$.

Next, find an elementary lower triangular matrix M_2 such that $A^{(2)} = M_2P_2A^{(1)}$ has zeros below the (2, 2) entry on the second column. M_2 is constructed as follows. First, construct an elementary matrix $\widehat{M}_2$ of order $(n - 1)$ such that

$$\widehat{M}_2 \begin{pmatrix} a_{22} \\ a_{32} \\ \vdots \\ a_{n2} \end{pmatrix} = \begin{pmatrix} * \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

Then define

$$M_2 = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & & & \\ \vdots & & \widehat{M}_2 & \\ 0 & & & \end{pmatrix}$$

Note that a_{ij} refers to the (i, j)th entry of the current matrix $P_2A^{(1)}$. At the end of step 2, we will have

$$A^{(2)} = M_2P_2A^{(1)} = \begin{pmatrix} * & * & \cdots & \cdots & * \\ 0 & * & \cdots & \cdots & * \\ 0 & 0 & * & \cdots & * \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & * & \cdots & * \end{pmatrix}$$

$$M_2 = \begin{pmatrix} 1 & 0 & 0 & \cdots & \cdots & 0 \\ 0 & 1 & 0 & \cdots & \cdots & 0 \\ 0 & m_{32} & 1 & \cdots & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & & \vdots \\ & & & & \ddots & \\ 0 & m_{n2} & 0 & \cdots & 0 & 1 \end{pmatrix}$$

where $m_{i2} = -a_{i2}/a_{22}$, $i = 3, 4, \ldots, n$.

Save the multipliers m_{i2} and record the row interchange.

Step k: In general, at the kth step, scan the entries of the kth column of the matrix $A^{(k-1)}$ below the row $(k-1)$ to identify the pivot $a_{r_k,k}$, form the permutation matrix P_k, and find an elementary lower triangular matrix M_k such that $A^{(k)} = M_kP_kA^{(k-1)}$ has zeros below the (k, k) entry on the kth column. M_k is constructed first by constructing $\widehat{M}_k$ of order $(n-k+1)$ such that

$$\widehat{M}_k \begin{pmatrix} a_{kk} \\ \vdots \\ \vdots \\ a_{nk} \end{pmatrix} = \begin{pmatrix} * \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

and then defining

$$M_k = \begin{pmatrix} I_{k-1} & 0 \\ 0 & \widehat{M}_k \end{pmatrix}$$

where 0 is a matrix of zeros. *The elements $a_{i,k}$ refer to the (i, k)th entries of the matrix $P_kA^{(k-1)}$.*

Step $n-1$: At the end of the $(n-1)$st step, the matrix $A^{(n-1)}$ will be an upper triangular matrix.

Form U: Set

$$A^{(n-1)} = U \tag{5.2.1}$$

Then

$$\begin{aligned} U = A^{(n-1)} &= M_{n-1}P_{n-1}A^{(n-2)} \\ &= M_{n-1}P_{n-1}M_{n-2}P_{n-2}A^{(n-3)} = \cdots \\ &= M_{n-1}P_{n-1}M_{n-2}P_{n-2}\cdots M_2P_2M_1P_1A \end{aligned}$$

Set

$$M_{n-1}P_{n-1}M_{n-2}P_{n-2}\cdots M_2P_2M_1P_1 = M \tag{5.2.2}$$

Then we have, from the preceding, the following factorization of A:

$$U = MA$$

THEOREM 5.2.2 Partial Pivoting Factorization Theorem Given an $n \times n$ nonsingular matrix A, Gaussian elimination with partial pivoting gives an upper triangular matrix U and a "permuted" lower triangular matrix M such that

$$MA = U$$

The matrices M and U are given by (5.2.2) and (5.2.1), respectively.

From (5.2.2) it is easy to see that there exists a permutation matrix P such that $PA = LU$.

Define

$$P = P_{n-1}\cdots P_2P_1 \tag{5.2.3}$$

$$L = P(M_{n-1}P_{n-1}\cdots M_1P_1)^{-1} \tag{5.2.4}$$

Then $PA = LU$.

COROLLARY 5.2.1 Partial Pivoting LU Factorization Theorem Given an $n \times n$ nonsingular matrix A, Gaussian elimination with partial pivoting yields an LU factorization of a permuted version of A:

$$PA = LU$$

P is a permutation matrix given by (5.2.3), L is a unit lower triangular matrix given by (5.2.4), and U is an upper triangular matrix given by (5.2.1).

Note: L does not need to be formed explicitly. The lower triangular entries are composed of the multipliers. (See Example 5.2.3.)

Example 5.2.2 Triangularize the matrix A below using partial pivoting; that is, find M and U such that $MA = U$. The pivot entry is encircled by $\square$.

$$A = \begin{pmatrix} 0.0001 & 1 \\ \boxed{1} & 1 \end{pmatrix}$$

Only one step is needed. The pivot entry is 1, and $r_1 = 2$.

$$P_1 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

$$P_1A = \begin{pmatrix} 1 & 1 \\ 0.0001 & 1 \end{pmatrix}$$

$$m_{21} = -\frac{-0.0001}{1} = -10^{-4}$$

$$M_1 = \begin{pmatrix} 1 & 0 \\ m_{21} & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ -10^{-4} & 1 \end{pmatrix}$$

$$M_1P_1A = \begin{pmatrix} 1 & 0 \\ -10^{-4} & 1 \end{pmatrix}\begin{pmatrix} 1 & 1 \\ 0.0001 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 0 & 0.9999 \end{pmatrix} = U$$

$$M = M_1P_1 = \begin{pmatrix} 1 & 0 \\ -10^{-4} & 1 \end{pmatrix}\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & -10^{-4} \end{pmatrix}$$

Example 5.2.3 Triangularize

$$A = \begin{pmatrix} 0 & 1 & 1 \\ \boxed{2} & 2 & 3 \\ 1 & 2 & 1 \end{pmatrix}$$

using partial pivoting. Express $A = MU$. Find also P and L such that $PA = LU$.

Step 1: The pivot entry is $a_{21} = 1$, and $r_1 = 2$.

$$P_1 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$P_1A = \begin{pmatrix} 2 & 2 & 3 \\ 0 & 1 & 1 \\ 1 & 2 & 1 \end{pmatrix}$$

$$M_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -0.5000 & 0 & 1 \end{pmatrix}$$

$$A^{(1)} = M_1P_1A = \begin{pmatrix} 2 & 2 & 3 \\ 0 & \boxed{1} & 1 \\ 0 & 1 & -0.5000 \end{pmatrix}$$

Step 2: The pivot entry is $a_{22} = 1$.

$$P_2A^{(1)} = \begin{pmatrix} 2 & 2 & 3 \\ 0 & 1 & 1 \\ 0 & 1 & -0.50000 \end{pmatrix}$$

$$P_2 = I_3 \qquad \text{(no interchange is necessary)}$$

$$\widehat{M}_2 = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$$

$$M_2 = \begin{pmatrix} I_1 & 0 \\ 0 & \widehat{M}_2 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{pmatrix}$$

$$U = A^{(2)} = M_2P_2A^{(1)} = \begin{pmatrix} 2 & 2 & 3 \\ 0 & 1 & 1 \\ 0 & 0 & -1.5000 \end{pmatrix}$$

$$M = M_2P_2M_1P_1 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ -1 & -0.5000 & 1 \end{pmatrix}$$

It is easily verified that $MA = U$.

Form L and P:

$$P = P_2P_1 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$L = P(M_2P_2M_1P_1)^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0.5000 & 1 & 1 \end{pmatrix}$$

It is easy to verify that $PA = LU$.

Forming the Matrix *M* and Other Computational Details

1. Each permutation matrix P_k can be formed just by recording the index r_k, because P_k is the permuted identity matrix in which rows k and r_k have been interchanged. However, neither the permutation matrix P_k nor the product $P_kA^{(k-1)}$ needs to be formed explicitly. This is because *the matrix $P_kA^{(k-1)}$ is just the permuted version of $A^{(k-1)}$ in which the rows r_k and k have been interchanged.*
2. Each elementary matrix M_k can be formed just by saving the $(n-k)$ multipliers. *The matrices $M_kP_kA^{(k-1)} = M_kB$ also do not need to be computed explicitly.* Note that the elements in the first k rows of the matrix M_kB are the same as the elements of the first k rows of the matrix B, and the elements in the remaining $(n-k)$ rows are given by:

$$b_{ij} + m_{ik}b_{kj} \qquad (i = k+1, \ldots, n; \quad j = k+1, \ldots, n)$$

3. The multipliers can be stored in the appropriate places of the lower triangular part of A (below the diagonal) as they are computed.
4. The final upper triangular matrix $U = A^{(n-1)}$ is stored in the upper triangular part of A.
5. The pivot indices r_k are stored in a separate single subscripted integer array.
6. A can be overwritten with each $A^{(k)}$ as soon as the latter is formed.

In view of the preceding discussion, we can now formulate the following *practical algorithm* for LU factorization with partial pivoting.

ALGORITHM 5.2.2 Triangularization Using Gaussian Elimination with Partial Pivoting
Let A be an $n \times n$ matrix. Then the following algorithm computes the triangularization of A with rows permuted, using Gaussian elimination with partial pivoting.

The upper triangular matrix U is stored in the upper triangular part of A, including the diagonal. The multipliers needed to compute the permuted triangular matrix M such that $MA = U$ are stored in the lower triangular part of A. The permutation indices r_k are stored in a separate array.

For $k = 1, 2, \ldots, n-1$ do

1: Find r_k so that $|a_{r_k,k}| = \max_{k \le i \le n} |a_{ik}|$. Save r_k. If $a_{r_k,k} = 0$, then stop. Otherwise, continue.

2: (Interchange the rows r_k and k) $a_{kj} \leftrightarrow a_{r_k,j}$ $(j = k, k+1, \ldots, n)$.

3: (Form the multipliers) $a_{ik} \equiv m_{ik} = -\dfrac{a_{ik}}{a_{kk}}$ $(i = k+1, \ldots, n)$.

4: (Update the entries) $a_{ij} \equiv a_{ij} + m_{ik}a_{kj} = a_{ij} + a_{ik}a_{kj}$ $(i = k+1, \ldots, n;\ j = k+1, \ldots, n)$.

Flop count. The algorithm requires about $n^3/3$ flops and $O(n^2)$ comparisons. (Note that the search for the pivot at step k requires $(n-k)$ comparisons.)

Note: Algorithm 5.2.2 does not give the matrices M and P explicitly. However, they can be constructed easily, as explained, from the multipliers and the permutation indices, respectively. The algorithm can easily be extended to the $m \times n$ case as shown in Section 5.2.1.

Remark: Algorithm 5.2.2 accesses the rows of A in the innermost loop and that is why it is known as the row-oriented Gaussian elimination (with partial pivoting) algorithm. It is also known as the kij algorithm; note that i and j appear in the inner loops. The column-oriented algorithm can be similarly developed. Such a column-oriented algorithm has been used in LINPACK (LINPACK routine SGEFA).

MATCOM Note: Algorithm 5.2.2 has been implemented in MATCOM program PARPIV. The matrices M and P have been explicitly computed.

Example 5.2.4 Let

$$A = \begin{pmatrix} 1 & 2 & 4 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

Step 1: $k = 1$.

1. The pivot entry is 7. $r_1 = 3$.
2. Interchange rows 3 and 1:

$$A \equiv \begin{pmatrix} 7 & 8 & 9 \\ 4 & 5 & 6 \\ 1 & 2 & 4 \end{pmatrix}$$

3. Form the multipliers:

$$a_{21} \equiv m_{21} = -\tfrac{4}{7}, \qquad a_{31} \equiv m_{31} = -\tfrac{1}{7}$$

4. Update:

$$A \equiv \begin{pmatrix} 7 & 8 & 9 \\ 0 & \frac{3}{7} & \frac{6}{7} \\ 0 & \frac{6}{7} & \frac{19}{7} \end{pmatrix}$$

Step 2: $k = 2$.

1. The pivot entry is $\frac{6}{7}$. $r_2 = 3$.
2. Interchange rows 2 and 3:

$$A \equiv \begin{pmatrix} 7 & 8 & 9 \\ 0 & \frac{6}{7} & \frac{19}{7} \\ 0 & \frac{3}{7} & \frac{6}{7} \end{pmatrix}$$

3. Form the multiplier:

$$m_{32} = -\tfrac{1}{2}$$

4. Update:

$$A \equiv \begin{pmatrix} 7 & 8 & 9 \\ 0 & \frac{6}{7} & \frac{19}{7} \\ 0 & 0 & -\frac{1}{2} \end{pmatrix}$$

Form M:

$$M = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & -\frac{1}{7} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \end{pmatrix}$$

5.2.3 Gaussian Elimination with Complete Pivoting

In Gaussian elimination with complete pivoting, at the kth step, the search for the pivots is made among all the entries of the submatrix below the first $(k-1)$ rows. Thus, if the pivot is a_{rs}, to bring this pivot to the (k, k) position, the interchange of the rows r and k has to be followed by the interchange of the columns k and s. This is equivalent to premultiplying the matrix $A^{(k-1)}$ by a permutation matrix P_k obtained by interchanging rows k and r and postmultiplying $P_kA^{(k-1)}$ by another permutation matrix Q_k obtained by interchanging the columns k and s of the identity matrix I. The ordinary Gaussian elimination is then applied to the matrix $P_kA^{(k-1)}Q_k$; that is, an elementary lower triangular matrix M_k is sought such that the matrix

$$A^{(k)} = M_kP_kA^{(k-1)}Q_k$$

has zeros on the kth column below the (k, k) entry. The matrix M_k can, of course, be computed in two smaller steps as before.

At the end of the $(n-1)$st step, the matrix $A^{(n-1)}$ is an upper triangular matrix. Set

$$A^{(n-1)} = U \tag{5.2.5}$$

Then

$$\begin{aligned} U = A^{(n-1)} &= M_{n-1}P_{n-1}A^{(n-2)}Q_{n-1} \\ &= M_{n-1}P_{n-1}M_{n-2}P_{n-2}A^{(n-3)}Q_{n-2}Q_{n-1} \\ &= \cdots = M_{n-1}P_{n-1}M_{n-2}P_{n-2}\cdots M_1P_1AQ_1Q_2\cdots Q_{n-1} \end{aligned}$$

Set

$$M_{n-1}P_{n-1}\cdots M_1P_1 = M \tag{5.2.6}$$

$$Q_1\cdots Q_{n-1} = Q \tag{5.2.7}$$

Then we have

$$U = MAQ$$

THEOREM 5.2.3 Complete Pivoting Factorization Theorem Given an $n \times n$ matrix A, Gaussian elimination with complete pivoting yields an upper triangular matrix U, a permuted lower triangular matrix M, and a permutation matrix Q such that

$$MAQ = U$$

where U, M, and Q are given by (5.2.5)–(5.2.7).

As in the case of partial pivoting, it is easy to see from (5.2.4) and (5.2.7) that the factorization $MAQ = U$ can be expressed in the form

$$PAQ = LU$$

COROLLARY 5.2.2 Complete Pivoting LU Factorization Theorem Gaussian elimination with complete pivoting yields the factorization $PAQ = LU$, where P and Q are permutation matrices given by

$$P = P_{n-1}\cdots P_1$$

$$Q = Q_1\cdots Q_{n-1}$$

and L is a unit lower triangular matrix given by

$$L = P(M_{n-1}P_{n-1}\cdots M_1P_1)^{-1}$$

Example 5.2.5 Triangularize

$$A = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 2 & \boxed{3} \\ 1 & 1 & 1 \end{pmatrix}$$

using complete pivoting.

Step 1: $k = 1$. The pivot entry is $a_{23} = 3$.

$$P_1 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$Q_1 = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \qquad P_1AQ_1 = \begin{pmatrix} 3 & 2 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}$$

$$M_1 = \begin{pmatrix} 1 & 0 & 0 \\ -\frac{1}{3} & 1 & 0 \\ -\frac{1}{3} & 0 & 1 \end{pmatrix}$$

$$A^{(1)} = M_1P_1AQ_1 = \begin{pmatrix} 3 & 2 & 1 \\ 0 & \frac{1}{3} & -\frac{1}{3} \\ 0 & \frac{1}{3} & \boxed{\frac{2}{3}} \end{pmatrix}$$

Step 2: $k = 2$. The pivot entry is $a_{33}^{(1)} = \frac{2}{3}$.

$$P_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \qquad Q_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

$$P_2A^{(1)}Q_2 = \begin{pmatrix} 3 & 1 & 2 \\ 0 & \frac{2}{3} & \frac{1}{3} \\ 0 & -\frac{1}{3} & \frac{1}{3} \end{pmatrix} \qquad \widehat{M}_2 = \begin{pmatrix} 1 & 0 \\ \frac{1}{2} & 1 \end{pmatrix}$$

$$M_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & \frac{1}{2} & 1 \end{pmatrix}$$

$$U = A^{(2)} = M_2P_2A^{(1)}Q_2 = M_2P_2(M_1P_1AQ_1)Q_2 = \begin{pmatrix} 3 & 1 & 2 \\ 0 & \frac{2}{3} & \frac{1}{3} \\ 0 & 0 & \frac{1}{2} \end{pmatrix}$$

(Using Corollary 5.2.2, find P, Q, L, and U for yourself such that $PAQ = LU$.)

Forming the Matrix *M* and other Computational Details

Remarks similar to those in the case of partial pivoting hold. The matrices P_k, Q_k, $P_kA^{(k-1)}Q_k$, M_k, and $M_kP_kA^{(k-1)}Q_k$ do not have to be formed explicitly wasting storage unnecessarily. It is enough to save the indices and the multipliers.

In view of our discussion on forming the matrices M_k and the permutation matrices P_k, we now present a practical Gaussian elimination algorithm with *complete pivoting*, which does not show the explicit formation of the matrices P_k, Q_k, M_k, M_kA, and P_kAQ_k.

ALGORITHM 5.2.3 Triangularization Using Gaussian Elimination with Complete Pivoting Given an $n \times n$ matrix, the following algorithm computes triangularization of A with rows and columns permuted, using Gaussian elimination with complete pivoting. The algorithm overwrites A with U. U is stored in the upper triangular part of A (including the diagonal) and the multipliers m_{ik} are stored in the lower triangular part. The permutation indices r_k and s_k are saved separately.

For $k = 1, 2, \ldots, n-1$ do

1: Find r_k and s_k such that $|a_{r_k,s_k}| = \max\{|a_{ij}| : i, j \geq k\}$, and save r_k and s_k. If $a_{r_k,s_k} = 0$, then stop. Otherwise, continue.

2: (Interchange the rows r_k and k) $a_{kj} \leftrightarrow a_{r_k,j}$ $(j = k, k+1, \ldots, n)$.

3: (Interchange the columns s_k and k) $a_{ik} \leftrightarrow a_{i,s_k}$ $(i = 1, 2, \ldots, n)$.

4: (Form the multipliers) $a_{ik} \equiv m_{ik} = -\dfrac{a_{ik}}{a_{kk}}$ $(i = k+1, \ldots, n)$.

5: (Update the entries of A) $a_{ij} \equiv a_{ij} + m_{ik}a_{kj} = a_{ij} + a_{ik}a_{kj}$ $(i = k+1, \ldots, n;\ j = k+1, \ldots, n)$.

Note: Algorithm 5.2.3 does not give the matrices M, P, and Q explicitly; they have to be formed, respectively, from the multipliers m_{ik} and the permutation indices r_k and s_k, as explained. As before, the algorithm can be extended to the $m \times n$ case.

Flop count. The algorithm requires $n^3/3$ flops and $n^3/3$ comparisons.

MATCOM Note: Algorithm 5.2.3 has been implemented in MATCOM program COMPIV. Matrices M, U, and Q have been computed explicitly.

Example 5.2.6 Let

$$A = \begin{pmatrix} 1 & 2 \\ 3 & \boxed{4} \end{pmatrix}$$

Just one step is needed.

$k = 1$: The pivot entry is 4.

$$r_1 = 2$$
$$s_1 = 2$$

First, the second and first rows are switched; this is followed by the switch of the second and first columns to obtain the pivot entry 4 in the (1, 1) position:

$$A \equiv \begin{pmatrix} 3 & 4 \\ 1 & 2 \end{pmatrix}$$

(after the interchange of the first and second rows).

$$A \equiv \begin{pmatrix} 4 & 3 \\ 2 & 1 \end{pmatrix}$$

(after the interchange of the first and second columns). The multiplier is $a_{21} \equiv m_{21} = -a_{21}/a_{11} = -\frac{2}{4} = -\frac{1}{2}$.

$$A \equiv \begin{pmatrix} 4 & 3 \\ 0 & -\frac{1}{2} \end{pmatrix}$$

(after updating the entries of A).

$$P_1 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad Q = Q_1 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

$$M = M_1P_1 = \begin{pmatrix} 1 & 0 \\ -\frac{1}{2} & 1 \end{pmatrix}\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & -\frac{1}{2} \end{pmatrix}$$

5.3 Stability of Gaussian Elimination

The stability of Gaussian elimination algorithms is better understood by measuring the growth of the elements in the reduced matrices $A^{(k)}$. *(Note that although pivoting keeps the multipliers bounded by unity, the elements in the reduced matrices still can grow arbitrarily.)* We remind the readers of the definition of the growth factor in this context, given in Chapter 3.

DEFINITION 5.3.1 The **growth factor** ρ is the ratio of the largest element (in magnitude) of $A, A^{(1)}, \ldots, A^{(n-1)}$ to the largest element (in magnitude) of A:

$$\rho = \frac{\max(\alpha, \alpha_1, \alpha_2, \ldots, \alpha_{n-1})}{\alpha}$$

where $\alpha = \max\limits_{i,j} |a_{ij}|$ and $\alpha_k = \max\limits_{i,j} |a_{ij}^{(k)}|$.

Example 5.3.1 Let

$$A = \begin{pmatrix} 0.0001 & 1 \\ 1 & 1 \end{pmatrix}$$

(a) Gaussian elimination without pivoting gives

$$A^{(1)} = U \cong \begin{pmatrix} 0.0001 & 1 \\ 0 & -10^4 \end{pmatrix}$$

$$\max |a_{ij}^{(1)}| = 10^4$$
$$\max |a_{ij}| = 1$$
$$\rho = \text{the growth factor} = 10^4$$

(b) Gaussian elimination with partial pivoting yields

$$A^{(1)} = U \approx \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$

$$\max |a_{ij}^{(1)}| = 1$$
$$\max |a_{ij}| = 1$$
$$\rho = \text{the growth factor} = 1$$

The question naturally arises: How large can the growth factor ρ be for an arbitrary matrix? We answer this question in the following.

Growth Factor of Gaussian Elimination for Complete Pivoting
For Gaussian elimination with complete pivoting,

$$\rho \le \{n \cdot 2^1 \cdot 3^{1/2} \cdot 4^{1/3} \cdots n^{1/n-1}\}^{1/2}.$$

This is a slowly growing function of n. Furthermore, in practice this bound is never attained. Indeed, there was an unproven conjecture by Wilkinson (1965, 213) that the growth factor for complete pivoting was bounded by n for real $n \times n$ matrices. Unfortunately, this conjecture has recently been settled negatively by Gould. Gould (1991) exhibited a 13×13 matrix for which Gaussian elimination with complete pivoting gave the growth factor $\rho = 13.0205$. In spite of Gould's recent result, *Gaussian elimination with complete pivoting is a stable algorithm.*

Growth Factor of Gaussian Elimination for Partial Pivoting
For Gaussian elimination with partial pivoting, $\rho \le 2^{n-1}$, that is,

ρ can be as big as 2^{n-1}

Unfortunately, matrices for which this bound is attained can be constructed. Consider the following example:

$$A = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & 1 \\ -1 & 1 & 0 & \cdots & 0 & 1 \\ \vdots & \ddots & \ddots & & & \vdots \\ \vdots & & \ddots & \ddots & & \vdots \\ \vdots & & & \ddots & \ddots & \\ -1 & \cdots & \cdots & \cdots & -1 & 1 \end{pmatrix}$$

That is,

$$a_{ij} = \begin{cases} 1 & \text{for } j = i, n \\ -1 & \text{for } j < i \\ 0 & \text{otherwise} \end{cases}$$

Wilkinson (1965, 212) showed that the growth factor ρ for this matrix with partial pivoting is 2^{n-1}. To see this, take the special case with $n = 4$.

$$A = \begin{pmatrix} 1 & 0 & 0 & 1 \\ -1 & 1 & 0 & 1 \\ -1 & -1 & 1 & 1 \\ -1 & -1 & -1 & 1 \end{pmatrix}$$

$$A^{(1)} = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & -1 & 1 & 2 \\ 0 & -1 & -1 & 2 \end{pmatrix}$$

$$A^{(2)} = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 4 \\ 0 & 0 & -1 & 4 \end{pmatrix}$$

$$A^{(3)} = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 4 \\ 0 & 0 & 0 & 8 \end{pmatrix}$$

Thus, the growth factor is

$$\rho = \frac{8}{1} = 2^3 = 2^{4-1}$$

Remark: Note that this is not the only matrix for which $\rho = 2^{n-1}$. Higham and Higham (1989) have identified a set of matrices for which $\rho = 2^{n-1}$. The matrix

$$B = \begin{pmatrix} 0.7248 & 0.7510 & 0.5241 & 0.7510 \\ 0.7317 & 0.1889 & 0.0227 & -0.7510 \\ 0.7298 & -0.3756 & 0.1150 & 0.7511 \\ -0.6993 & -0.7444 & 0.6647 & -0.7500 \end{pmatrix}$$

is such a matrix. Recently, Foster (1994) and Wright (1993) have given examples where Gaussian elimination with partial pivoting can give large growth factors.

Examples of the preceding type are rare. Indeed, in many practical examples, the elements of the matrices $A^{(k)}$ very often continue to decrease in size. *Thus, Gaussian elimination with partial pivoting is not unconditionally stable in theory, but in practice it can be considered a stable algorithm.*

Growth Factor and Stability of Gaussian Elimination without Pivoting
For Gaussian elimination without pivoting, *ρ can be arbitrarily large*, except for a few special cases, as we shall see later, such as symmetric positive definite matrices. Thus, *Gaussian elimination without pivoting is, in general, a completely unstable algorithm.*

5.4 Householder Transformations and Applications to QR Factorization and Hessenberg Reduction

DEFINITION 5.4.1 A matrix of the form

$$H = I - \frac{2uu^T}{u^T u}$$

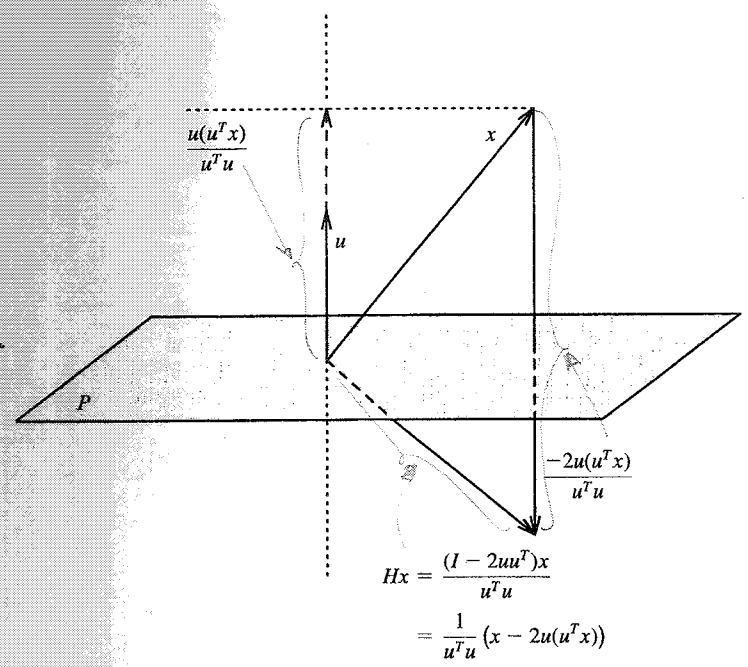


FIGURE 5.1

where u is a nonzero vector, is called a **Householder matrix** after the celebrated American numerical analyst Alston Householder (1904–1993).

A Householder matrix is also known as an *elementary reflector* or a *Householder transformation*. Figure 5.1 gives a geometric interpretation of a Householder transformation.

With this geometric interpretation, the following results become clear:

- $\|Hx\|_2 = \|x\|_2$ for every $x \in \mathbb{R}^n$. A reflection does not change the length of the vector.
- H is an orthogonal matrix. $\|Hx\|_2 = \|x\|_2$ for every x implies that H is orthogonal.
- $H^2 = I$. Hx reflects x to the other side of P, but $H^2x = H(Hx)$ reflects it back to x.
- $Hy = y$ for every $y \in P$. Vectors in P cannot be reflected away.
- H has a simple eigenvalue -1 and $(n-1)$-fold eigenvalue 1. $P = \{v \in \mathbb{R}^n : v^Tu = 0\}$ has $n-1$ linearly independent vectors $y_1, \ldots, y_{n-1}$ and $Hy_i = y_i$, $i = 1, \ldots, n-1$. So 1 is an $(n-1)$-fold eigenvalue. Also, H reflects u to $-u$; that is, $Hu = -u$. Thus, -1 is an eigenvalue of H that must be a simple eigenvalue because H can have only n eigenvalues.
- $\det(H) = (-1) \cdot 1 \cdots 1 = -1$.

Given $x, y \in \mathbb{R}^n$ with $\|x\|_2 = \|y\|_2$; if we choose u to be a unit vector parallel to $x - y$, then $H = I - 2uu^T$ reflects x to y.

The importance of Householder matrices lies in the fact that they can also be used to create zeros in a vector.

LEMMA 5.4.1 Given a nonzero vector $x \neq e_1$, there always exists a Householder matrix H such that Hx is a multiple of e_1.

Proof: Define

$$H = I - 2\frac{uu^T}{u^Tu}$$

with $u = x + \text{sign}(x_1)\|x\|_2 e_1$. Then, by direct multiplication, one can see that Hx is a multiple of e_1. (See #3 of the **Notes** below.) ■

Notes:

1. If x_1 is zero, its sign can be chosen either + or −. Any possibility of overflow or underflow in the computation of $\|x\|_2$ can be avoided by scaling the vector x. Thus the vector u should be determined from the vector $x/\max_i\{|x_i|\}$ rather than from the vector x itself.
2. Since $u = x + \text{sign}(x_1)\|x\|_2 e_1 = (x_1 + \text{sign}(x_1)\|x\|_2, x_2, \ldots, x_n)^T$, the vector u can be stored over x.
3. Also, $Hx = (-\text{sign}(x_1)\|x\|, 0, \ldots, 0)^T$.

ALGORITHM 5.4.1 Creating Zeros in a Vector with a Householder Matrix Given a nonzero vector x, the following algorithm computes a vector u and a scalar σ such that $Hx = \left(I - 2\frac{uu^T}{u^Tu}\right)x = (\sigma, 0, \ldots, 0)^T$, u is stored over x.

Step 1: $m = \max(|x_i|), i = 1, 2, \ldots, n$
Step 2: $x_i \equiv u_i = \frac{x_i}{m}, i = 1, 2, \ldots, n$
Step 3: $\sigma = \text{sign}(u_1)\sqrt{u_1^2 + u_2^2 + \cdots + u_n^2}$
Step 4: $x_1 \equiv u_1 = u_1 + \sigma$
Step 5: $\sigma = -m \cdot \sigma$

Example 5.4.1 Given

$$x = \begin{pmatrix} 0 \\ 4 \\ 1 \end{pmatrix}$$

find u such that $Hx = (\sigma, 0, 0)^T$, where $H = I - 2\frac{uu^T}{u^Tu}$.

$m = 4, \quad u_1 = 1.0308$
$u_2 = 1, \quad u_3 = .25$

So, $u = (1.0308, 1, 0.25)^T$, $\sigma = -4.1232$.

It is easily verified that

$$Hx = (-4.1232, 0, 0)^T$$

Flop count and roundoff property. Creating zeros in a vector by a Householder matrix is a cheap and numerically stable procedure. It takes only $2(n+1)$ flops and one square root to create zeros in the positions 2 through n in a vector, and it can be shown (Wilkinson 1965, 152–162) that if $\widehat{H}$ is the computed Householder matrix, then

$$\|H - \widehat{H}\| \leq 10\mu$$

Moreover, if inner product in double precision is accumulated, then

$$\mathrm{fl}(\widehat{H}x) = H(x + e)$$

where

$$|e| \leq c\mu\|x\|_2$$

c is a constant of order unity, and μ is the machine precision.

MATCOM Note: Algorithm 5.4.1 has been implemented in MATCOM program HOUSZERO.

In the following, we now show how matrix-vector and matrix-matrix products with a Householder matrix can be performed just by using the vector defining the Householder matrix (see also Algorithm 4.2.1). *The Householder matrix itself does not have to be formed in practice.*

Computing the Matrix-Vector Product with a Householder Matrix

Let x be an n-vector and H be an $n \times n$ Householder matrix defined by the vector u. Then

$$Hx = \left(I - \frac{2uu^T}{u^Tu}\right)x = x - \beta u(u^Tx)$$

where $\beta = \dfrac{2}{u^Tu}$.

ALGORITHM 5.4.2 Matrix-Vector Product with a Householder Matrix Given an n-vector u defining the Householder matrix $H = I - 2\dfrac{uu^T}{u^Tu}$, and n-vector $x = (x_1, x_2, \ldots, x_n)^T$, the following algorithm computes the matrix-vector product Hx. The algorithm overwrites x with Hx.

Step 1: Compute $\beta = \dfrac{2}{u^Tu}$
Step 2: Compute the sum $s = \sum_{i=1}^{n} u_i x_i$
Step 3: Update $\beta \equiv \beta s$
Step 4: For $i = 1, \ldots, n$ do
$x_i \equiv x_i - \beta u_i$

Remark: Note that Algorithm 5.4.2 is a special case of Algorithm 4.2.1. We repeated it here, for the sake of convenience.

Flop count. The algorithm requires only $O(n)$ flops, compared to $O(n^2)$ flops required for the usual matrix-vector multiplication.

Computing Matrix-Matrix Product with Premultiplication by a Householder Matrix

Algorithm 4.2.1 describes how to compute the product HA without explicitly forming the Householder matrix H. We also note here that this product can also be computed using Algorithm 5.4.2. Let $A = (a_1, a_2, \ldots, a_n)$. Then $HA = H(a_1, a_2, \ldots, a_n) = (Ha_1, Ha_2, \ldots, Ha_n)$. Each column Ha_i, $i = 1, \ldots, n$ can now be computed out of the vector u using Algorithm 5.4.2.

Computing Matrix-Matrix Product with Postmultiplication by a Householder Matrix

Let

$$A = \begin{pmatrix} a_1^T \\ a_2^T \\ \vdots \\ a_n^T \end{pmatrix}$$

$a_i^T H = (Ha_i)^T$

Then

because of orthogonality

$$AH = \begin{pmatrix} a_1^T H \\ a_2^T H \\ \vdots \\ a_n^T H \end{pmatrix}$$

$H^2 = I \Rightarrow H^{-1} = H^T = H$

Since $(a_i^T H)^T = H^T a_i = Ha_i$, $i = 1, \ldots, n$, each row of AH can be computed by matrix-vector multiplication with the Householder matrix H, as shown in Algorithm 4.2.1 or Algorithm 5.4.2.

Note: We emphasize here that in practice a matrix-matrix or matrix-vector product with a Householder matrix should be computed as shown above, without explicitly forming the Householder matrix H.

5.4.1 Householder Matrices and QR Factorization

THEOREM 5.4.1 Householder QR Factorization Theorem Given an $n \times n$ matrix A, there exists an orthogonal matrix Q and an upper triangular matrix R such that

$$A = QR$$

The matrix Q can be written as $Q = H_1 H_2 \cdots H_{n-1}$, where each H_i is a Householder matrix.

A factorization of A into QR is called a QR factorization of A.

As we will see later, the QR factorization plays a very significant role in numerical solutions of linear systems, least-squares problems, and eigenvalue and singular-value computations.

We now show how the QR factorization of A can be obtained using Householder matrices, which will provide a constructive proof of Theorem 5.4.1.

As in the process of LU factorization, this can be achieved in $(n-1)$ steps; however, unlike the Gaussian elimination process, the *Householder process can always be carried out to completion.*

Step 1: Construct a Householder matrix H_1 such that H_1A has zeros below the (1, 1) entry in the first column:

$$H_1A = \begin{pmatrix} * & * & \cdots & * \\ 0 & * & \cdots & * \\ 0 & * & \cdots & * \\ \vdots & \vdots & & \vdots \\ 0 & * & \cdots & * \end{pmatrix}$$

Note that it is sufficient to construct $H_1 = I - 2u_nu_n^T/(u_n^Tu_n)$ such that

$$H_1\begin{pmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{n1} \end{pmatrix} = \begin{pmatrix} * \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

for then H_1A will have the above form.

Overwrite A with $A^{(1)} = H_1A$ for use in the next step. Because $A^{(1)}$ overwrites A, $A^{(1)}$ can be written as

$$A \equiv A^{(1)} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ 0 & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ 0 & a_{n2} & \cdots & a_{nn} \end{pmatrix}$$

Step 2: Construct a Householder matrix H_2 such that $H_2A^{(1)}$ has zeros below the (2, 2) entry in the second column and the zeros already created in the first column of $A^{(1)}$ in step 1 are not destroyed:

$$A^{(2)} = H_2A^{(1)} = \begin{pmatrix} * & * & * & \cdots & * \\ 0 & * & * & \cdots & * \\ 0 & 0 & * & \cdots & * \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & * & \cdots & * \end{pmatrix}$$

H_2 can be constructed as follows. First, construct a Householder matrix $\widehat{H}_2 = I_{n-1} - 2u_{n-1}u_{n-1}^T/(u_{n-1}^Tu_{n-1})$, of order $n-1$ such that

$$\widehat{H}_2\begin{pmatrix} a_{22} \\ a_{32} \\ \vdots \\ \vdots \\ a_{n2} \end{pmatrix} = \begin{pmatrix} * \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

and then define

$$H_2 = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & & & \\ \vdots & & \widehat{H}_2 & \\ 0 & & & \end{pmatrix}$$

$A^{(2)} = H_2A^{(1)}$ will then have the form given earlier. *Overwrite A with $A^{(2)}$.*

Note: Because H_2 also has zeros below the diagonal on the first column, premultiplication of $A^{(1)}$ by H_2 preserves the zeros already created in step 1.

Step k: In general, at the kth step, first create a Householder matrix

$$\widehat{H}_k = I_{n-k+1} - \frac{2u_{n-k+1}u_{n-k+1}^T}{(u_{n-k+1}^Tu_{n-k+1})}$$

of order $n-k+1$ such that

$$\widehat{H}_k\begin{pmatrix} a_{kk} \\ \vdots \\ \vdots \\ a_{nk} \end{pmatrix} = \begin{pmatrix} * \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

and then, defining

$$H_k = \begin{pmatrix} I_{k-1} & 0 \\ 0 & \widehat{H}_k \end{pmatrix}$$

compute $A^{(k)} = H_kA^{(k-1)}$.

Overwrite A with $A^{(k)}$. The matrix $A^{(k)}$ will have zeros on the kth column below the (k, k)th entry, and the zeros already created in previous columns will not be destroyed. At the end of the $(n-1)$st step, the resulting matrix $A^{(n-1)}$ will be an upper triangular matrix R.

Because

$$A^{(k)} = H_kA^{(k-1)}, \qquad k = n-1, \ldots, 2$$

we have

$$\begin{aligned} R = A^{(n-1)} &= H_{n-1}A^{(n-2)} = H_{n-1}H_{n-2}A^{(n-3)} \\ &= \cdots = H_{n-1}H_{n-2}\cdots H_2H_1A \end{aligned} \tag{5.4.1}$$

Set

$$Q^T = H_{n-1}H_{n-2}\cdots H_2H_1 \tag{5.4.2}$$

Since each Householder matrix H_k is orthogonal, so is Q^T. Therefore, from (5.4.1), we have

$$R = Q^TA \quad \text{or} \quad A = QR \tag{5.4.3}$$

(Note that $Q = H_1^TH_2^T\cdots H_{n-1}^T$ is also orthogonal.)

Example 5.4.2 Let

$$A = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 1 & 1 \end{pmatrix}$$

Step 1: $k = 1$.
Construct H_1:

$$H_1 \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} * \\ 0 \\ 0 \end{pmatrix}$$

$$u_3 = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} + \sqrt{2} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} \sqrt{2} \\ 1 \\ 1 \end{pmatrix}$$

$$H_1 = I_3 - \frac{2u_3u_3^T}{u_3^Tu_3} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} - \begin{pmatrix} 1 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{\sqrt{2}} & \frac{1}{2} & \frac{1}{2} \end{pmatrix}$$

$$= \begin{pmatrix} 0 & -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{\sqrt{2}} & -\frac{1}{2} & \frac{1}{2} \end{pmatrix}$$

Form $A^{(1)}$:

$$A^{(1)} = H_1A = \begin{pmatrix} -\sqrt{2} & \frac{-3\sqrt{2}}{2} & 2\sqrt{2} \\ 0 & \frac{1-\sqrt{2}}{2} & \frac{2-\sqrt{2}}{2} \\ 0 & -\frac{(1+\sqrt{2})}{2} & -\frac{(2+\sqrt{2})}{2} \end{pmatrix}$$

Overwrite:

$$A \equiv A^{(1)} \equiv \begin{pmatrix} -1.4142 & -2.1213 & -2.8284 \\ 0 & -0.2071 & 0.2929 \\ 0 & -1.2071 & -1.7071 \end{pmatrix}$$

Step 2: $k = 2$.
Construct $\widehat{H}_2$:

$$\widehat{H}_2 \begin{pmatrix} -0.2071 \\ -1.2071 \end{pmatrix} = \begin{pmatrix} * \\ 0 \end{pmatrix}$$

$$u_2 = \begin{pmatrix} -0.2071 \\ -1.2071 \end{pmatrix} - 1.2247 \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} -1.4318 \\ -1.2071 \end{pmatrix}$$

$$\widehat{H}_2 = \begin{pmatrix} -0.1691 & -0.9856 \\ -0.9856 & 0.1691 \end{pmatrix}$$

Construct H_2:

$$H_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -0.1691 & -0.9856 \\ 0 & -0.9856 & 0.1691 \end{pmatrix}$$

Form $A^{(2)}$:

$$A^{(2)} = H_2A^{(1)} = H_2H_1A$$
$$= \begin{pmatrix} -1.4142 & -2.1213 & -2.8284 \\ 0 & 1.2247 & 1.6330 \\ 0 & 0 & -0.5774 \end{pmatrix} = \boldsymbol{R}$$

Form Q:

$$Q = H_1H_2 = \begin{pmatrix} 0 & 0.8165 & 0.5774 \\ -0.7071 & 0.4082 & -0.5774 \\ -0.7071 & -0.4082 & 0.5774 \end{pmatrix}$$

Forming the Matrix Q and Other Computational Details

1. Each Householder matrix H_k is uniquely determined by the vector u_{n-k+1} that defines the matrix $\widehat{H}_k$. It is therefore sufficient to save the vector u_{n-k+1}. If the matrix Q is needed explicitly, it can be computed from the Householder matrices H_1 through H_{n-1}.
2. The vector $u_{n-k+1} = (u_{kk}, \ldots, u_{nk})^T$ has $(n-k+1)$ components, whereas only $(n-k)$ zeros are produced at the kth step. Thus, one possible scheme for storage will be to store second through $(n-k+1)$ components in positions $(k+1, k), \ldots, (n, k)$ of A. The first component has to be stored separately in a one-dimensional array.
3. The matrix product $A^{(k)} = H_kA^{(k-1)}$ can be computed out of the vector u_{n-k+1} defining H_k without ever forming the matrix H_k. To illustrate, consider

$$A^{(2)} = H_2A^{(1)}$$
$$= \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & & & \\ \vdots & I - 2\frac{u_{n-1}u_{n-1}^T}{u_{n-1}^Tu_{n-1}} & & \\ 0 & & & \end{pmatrix} \begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1n}^{(1)} \\ 0 & & & \\ \vdots & A_1^{(1)} & & \\ 0 & & & \end{pmatrix}$$

We see that the first row of $A^{(2)}$ is the same as the first row of $A^{(1)}$, the entries of the first column of $A^{(2)}$ below the diagonal entry are all zero, and the other entries of $A^{(2)}$ are given by the matrix product

$$\left(I - 2\frac{u_{n-1}u_{n-1}^T}{u_{n-1}^T u_{n-1}}\right) A_1^{(1)}$$

which can be computed just by storing the vector u_{n-1} as shown in Algorithm 4.2.1 (see also Section 5.4). The general case is analogous.

In general, to compute $A^{(k)}$ from $A^{(k-1)}$:

a. Leave the first $(k-1)$ rows unchanged.
b. Update the next $(n-k+1)$ rows by forming the product

$$\left(I - 2\frac{u_{n-k+1}u_{n-k+1}^T}{u_{n-k+1}^T u_{n-k+1}}\right) A_{k-1}^{(k-1)}$$

where $A_{k-1}^{(k-1)}$ is the submatrix obtained from $A^{(k-1)}$ by removing the first $(k-1)$ rows and $(k-1)$ columns, using Algorithm 4.2.1, which shows that the product can be computed just by saving the vector u_{n-k+1}.

4. The nonzero entries of $A^{(k)}$ can be stored in the corresponding places of A, as soon as the former is formed.
5. With the above storage scheme, at the end of the $(n-1)$th step we have

$$A \equiv A^{(n-1)} = \begin{pmatrix} a_{11} & a_{12} & \cdots & \cdots & a_{1n} \\ u_{21} & a_{22} & \cdots & \cdots & a_{2n} \\ u_{31} & u_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ u_{n1} & u_{n2} & \cdots & u_{n,n-1} & a_{nn} \end{pmatrix}$$

The one-dimensional array v containing the first components of u_{n-k+1} is given by

$$v = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_{n-2} \end{pmatrix} \equiv \begin{pmatrix} u_{11} \\ u_{22} \\ \vdots \\ u_{n-1,n-1} \end{pmatrix}$$

Thus, all of the information needed to compute Q is contained in the lower triangular part of the matrix A below the diagonal and in the single-dimensional array v containing the first components of the vectors defining the Householder matrices H_1 through H_{n-1}. The matrix Q, if needed, can be computed out of the vectors defining the matrices $H_1, H_2, \ldots, H_{n-1}$.

ALGORITHM 5.4.3 Householder QR Factorization Given an $n \times n$ matrix A, the following algorithm creates the vectors $u_{n-k+1} = (u_{kk}, \ldots, u_{nk})^T$, $k = 1, \ldots, n-1$ defining the Householder matrices H_1 through H_{n-1}, and an upper triangular matrix R such that $A = QR$ with $Q = H_1 H_2 \ldots H_{n-1}$. The components $u_{k+1,k}$ through u_{nk} are stored in the positions $(k+1, k)$ through (n, k) of A. The first components u_{kk} are stored in a one-dimensional array $v = (v_1, v_2, \ldots, v_n)^T$. The nonzero entries of $A^{(k)} = H_k A^{(k-1)}$, $k = 1, \ldots, n-1$ overwrite those of A; that is, R overwrites the triangular part of A.

For $k = 1, 2, \ldots, n-1$ do

1: Find the vector $u_{n-k+1} = (u_{kk}, \ldots, u_{nk})^T$ defining the Householder matrix $\widehat{H}_k$ and a scalar σ such that

$$\widehat{H}_k \begin{pmatrix} a_{kk} \\ \vdots \\ a_{nk} \end{pmatrix} = \begin{pmatrix} \sigma \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

(use Algorithm 5.4.1).

2: Overwrite a_{kk} by σ:

$$a_{kk} \equiv \sigma$$

3: Store the vector u_{n-k+1} as follows:

$$a_{ik} \equiv u_{ik}, \quad i = k+1, \ldots, n$$
$$v_k \equiv u_{kk}.$$

4: Compute

$$\beta = \frac{2}{u_{n-k+1}^T u_{n-k+1}}$$

5: Update the entries of the submatrix of A containing rows k through n and columns $k+1$ through n. For $j = k+1, \ldots, n$ do

1. $s = \beta \sum_{i=k}^{n} u_{ik} a_{ij}$
2. $a_{ij} \equiv a_{ij} - s u_{ik}$ $(i = k, k+1, \ldots, n)$

Example 5.4.3 Find the QR factorization of

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 5 & 7 \\ 3 & 8 & 9 \end{pmatrix}$$

using the Householder method (Algorithm 5.4.3).

$k = 1$: Using Algorithm 5.4.1, we find vector $u_3 = (u_{11}, u_{21}, u_{31})^T$ such that

$$\widehat{H}_1 \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} = \left(I - \frac{2u_3 u_3^T}{u_3^T u_3}\right) \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} = \begin{pmatrix} \sigma \\ 0 \\ 0 \end{pmatrix}$$

$$u_3 = (1.5806, 0.6667, 1)^T$$
$$\sigma = -3.7417$$

We now update the entries of A:

$$A \equiv H_1 A = \begin{pmatrix} -3.7417 & -9.6214 & -11.7595 \\ 0 & 0.0982 & 0.7745 \\ 0 & 0.6473 & -0.3382 \end{pmatrix}$$

Note that we needed to update only a_{12}, a_{13}; a_{22}, a_{23}; and a_{32}, a_{33} because $a_{11} \equiv \sigma = -3.7417$, $a_{21} \equiv 0$, and $a_{31} \equiv 0$. Also note that $\widehat{H}_1 = H_1$.

$k = 2$: Using Algorithm 5.4.1, we find vector $u_2 = (u_{22}, u_{32})^T$ such that

$$\widehat{H}_2 \begin{pmatrix} 0.0982 \\ 0.6473 \end{pmatrix} = \begin{pmatrix} \sigma \\ 0 \end{pmatrix}$$

$$u_2 = (1.1631, 1)^T, \quad \sigma = -0.6547$$

We now update entries a_{23} and a_{33} only. Note that $a_{22} \equiv \sigma = -0.6547$ and $a_{32} = 0$; the rest of the entries are unchanged.

$$a_{23} = 0.2182, \quad a_{33} = -0.8165$$

Thus,

$$A \equiv H_2 H_1 A = R = \begin{pmatrix} -3.7417 & -9.6214 & -11.7595 \\ 0 & -0.6547 & 0.2182 \\ 0 & 0 & -0.8165 \end{pmatrix}$$

Forming Q: Q can be formed out of vectors u_3 and u_2.

$$Q^T = H_2 H_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & I - \dfrac{2u_2 u_2^T}{u_2^T u_3} & \\ 0 & & \end{pmatrix} \left(I - \frac{2u_3 u_3^T}{u_3^T u_3} \right)$$

$$= \begin{pmatrix} -0.2673 & 0.8729 & 0.4082 \\ -0.5345 & 0.2182 & -0.8165 \\ -0.8018 & -0.4364 & 0.4082 \end{pmatrix}$$

(Verify now that $Q^T A = R$.)

Flop count. The algorithm requires approximately $\frac{2}{3}n^3$ flops just to compute the triangular matrix R. This can be seen as follows.

The construction of $\widehat{H}_k$ (and therefore of H_k) requires about $2(n-k)$ flops, whereas that of $A^{(k)}$ from $A^{(k)} = H_k A$ (taking advantage of the special structure of H_k) requires roughly $2(n-k)^2$ flops. Thus,

$$\begin{aligned} \text{Total number of flops} &= 2\sum_{k=1}^{n-1} [(n-k)^2 + (n-k)] \\ &= 2[(n-1)^2 + (n-2)^2 + \cdots + 1^2] + 2[(n-1) \\ &\quad + (n-2) + \cdots + 1] \\ &= 2\frac{n(n-1)(2n-1)}{6} + 2 \cdot \frac{n(n-1)}{2} \\ &\approx \frac{2n^3}{3} \end{aligned}$$

Note: The preceding count does not take into account the explicit construction of Q. Q is available only in factored form. It should be noted that in a majority of practical applications, it is sufficient to have Q in this factored form; in many applications, Q is not needed at all. If Q is needed explicitly, another $\frac{2}{3}n^3$ flops will be required (Exercise 22).

Roundoff property. In the presence of roundoff errors the algorithm computes QR decomposition of a slightly perturbed matrix. Specifically, it can be shown (Wilkinson 1965, 236) that if $\widehat{R}$ denotes the computed R, then there exists an orthogonal $\widehat{Q}$ such that

$$A + E = \widehat{Q}\widehat{R}$$

The error matrix E satisfies

$$\|E\|_F \le \phi(n)\mu\|A\|_F$$

where $\phi(n)$ is a slowly growing function of n and μ is the machine precision. If the inner products are accumulated in double precision, then it can be shown (Golub and Wilkinson 1966) that $\phi(n) = 12 \cdot 5n$. *The algorithm is thus stable.*

MATCOM Note: Algorithm 5.4.3 has been implemented in MATCOM program HOUSQR. Q and R have been explicitly computed.

5.4.2 Householder QR Factorization of a Nonsquare Matrix

In many applications (such as in least-squares problems), the QR factorization of an $m \times n$ matrix A is needed. The preceding Householder method can be applied to obtain QR factorization of such an A as well. The process consists of $s = \min\{n, m-1\}$ steps; the Householder matrices $H_1, H_2, \ldots, H_s$ are constructed successively so that

$$H_s H_{s-1} \cdots H_2 H_1 A = Q^T A = \begin{cases} \begin{pmatrix} R_1 \\ 0 \end{pmatrix}, & \text{if } m > n \\ (R_1, S), & \text{if } m \le n \end{cases}$$

Flop count and roundoff property. The Householder method in this case requires

1. $n^2(m - \frac{n}{3})$ flops if $m \ge n$;
2. $m^2(n - \frac{m}{3})$ flops if $m \le n$.

The roundoff property is the same as in the previous case. *The QR factorization of a rectangular matrix using Householder transformations is stable.*

MATCOM Note: The MATCOM program HOUSQRN implements the method for Householder QR factorization of a nonsquare matrix.

Example 5.4.4 Let

$$A = \begin{pmatrix} 1 & 1 \\ 0.0001 & 0 \\ 0 & 0.0001 \end{pmatrix}$$

$$s = \min(2, 2) = 2$$

Step 1: Form H_1.

$$u_2 = \begin{pmatrix} 1 \\ 0.0001 \\ 0 \end{pmatrix} + \sqrt{1 + (0.0001)^2} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 2 \\ 0.0001 \\ 0 \end{pmatrix}$$

$$H_1 = I - \frac{2u_2u_2^T}{u_2^Tu_2} = \begin{pmatrix} -1 & -0.0001 & 0 \\ -0.0001 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$A^{(1)} = H_1A = \begin{pmatrix} -1 & -1 \\ 0 & -0.0001 \\ 0 & 0.0001 \end{pmatrix}$$

(In practice, H_1A should be implicitly formed from u_2.)

Step 2: Form H_2.

$$u_1 = \begin{pmatrix} -0.0001 \\ 0.0001 \end{pmatrix} - \sqrt{(-0.0001)^2 + (0.0001)^2} \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$= 10^{-4} \begin{pmatrix} -2.4141 \\ .1000 \end{pmatrix}$$

$$\widehat{H}_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} - 2\frac{u_1u_1^T}{u_1^Tu_1} = \begin{pmatrix} -0.7071 & 0.7071 \\ 0.7071 & 0.7071 \end{pmatrix}$$

$$H_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -0.7071 & 0.7071 \\ 0 & 0.7071 & 0.7071 \end{pmatrix}$$

Form R:

$$H_2A^{(1)} = \begin{pmatrix} -1 & -1 \\ 0 & 0.0001 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} R_1 \\ 0 \end{pmatrix}$$

(In practice, $H_2A^{(1)}$ should be implicitly formed from u_1.)

Form Q and R_1.

$$Q = H_1H_2 = \begin{pmatrix} -1 & 0.0001 & -0.0001 \\ -0.0001 & -0.7071 & 0.7071 \\ 0 & 0.7071 & 0.7071 \end{pmatrix}$$

$$R_1 = \begin{pmatrix} -1 & -1 \\ 0 & 0.0001 \end{pmatrix}$$

5.4.3 Householder Matrices and Reduction to Hessenberg Form

THEOREM 5.4.2 Hessenberg Reduction Theorem An arbitrary $n \times n$ matrix can always be transformed to an upper Hessenberg matrix H_u by orthogonal similarity:

$$PAP^T = H_u$$

As noted before, reduction to Hessenberg form is very important in eigenvalue computations. The matrix A is routinely transformed to a Hessenberg matrix before the process of eigenvalue computations (known as the QR iterations) starts. Hessenberg forms are also useful tools in many other applications, such as in control theory and signal processing.

The idea of orthogonal factorization using Householder matrices described in the previous section can easily be extended to obtain P and H_u.

The matrix P is constructed as the product of $(n-2)$ Householder matrices P_1 through P_{n-2}. P_1 is constructed to create zeros in the first column of A below the entry (2, 1), P_2 is determined to create zeros below the entry (3, 2) of the second column of the matrix $P_1AP_1^T$, and so on.

The process consists of $(n-2)$ *steps.* (Note that an $n \times n$ Hessenberg matrix contains at least $(n-2)(n-1)/2$ zeros.)

Step 1: Find a Householder matrix $\widehat{P}_1$ of order $n-1$ such that

$$\widehat{P}_1 \begin{pmatrix} a_{21} \\ a_{31} \\ \vdots \\ a_{n1} \end{pmatrix} = \begin{pmatrix} * \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

Define

$$P_1 = \begin{pmatrix} I_1 & 0 \\ 0 & \widehat{P}_1 \end{pmatrix}$$

and compute

$$A^{(1)} = P_1AP_1^T$$

Overwrite A with $A^{(1)}$. Then

$$A \equiv A^{(1)} = \begin{pmatrix} * & * & \cdots & * \\ * & * & \cdots & * \\ 0 & * & \cdots & * \\ \vdots & \vdots & & \vdots \\ 0 & * & \cdots & * \end{pmatrix}$$

Step 2: Find a Householder matrix $\widehat{P}_2$ of order $(n-2)$ such that

$$\widehat{P}_2 \begin{pmatrix} a_{32} \\ \vdots \\ \vdots \\ a_{n2} \end{pmatrix} = \begin{pmatrix} * \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

Define

$$P_2 = \begin{pmatrix} I_2 & 0 \\ 0 & \widehat{P}_2 \end{pmatrix}$$

and compute $A^{(2)} = P_2A^{(1)}P_2^T$.

Overwrite A with $A^{(2)}$. Then

$$A \equiv A^{(2)} = \begin{pmatrix} * & * & * & \cdots & * \\ * & * & * & \cdots & * \\ 0 & * & * & \cdots & * \\ 0 & 0 & * & \cdots & * \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & * & \cdots & * \end{pmatrix}$$

The general step k now can easily be written down.

At the end of $(n-2)$ steps, the matrix $A^{(n-2)}$ is an upper Hessenberg matrix H_u. That H_u is orthogonally similar to A is seen as follows:

$$\begin{aligned} H_u = A^{(n-2)} &= P_{n-2}A^{(n-3)}P_{n-2}^T \\ &= P_{n-2}(P_{n-3}A^{(n-4)}P_{n-3}^T)P_{n-2}^T \\ &\vdots \\ &= (P_{n-2}P_{n-3}\cdots P_1)A(P_1^TP_2^T\cdots P_{n-3}^TP_{n-2}^T) \end{aligned} \tag{5.4.4}$$

Set

$$P = P_{n-2}P_{n-3}\cdots P_1 \tag{5.4.5}$$

We then have $H_u = PAP^T$. Since each Householder matrix P_i is orthogonal, the matrix P, which is the product of $(n-2)$ Householder matrices, is also orthogonal.

Note: It is important to note that since P_k has the form

$$\begin{pmatrix} I_k & 0 \\ 0 & \widehat{P}_k \end{pmatrix}$$

postmultiplication of P_kA by P_k^T does not destroy the zeros already created in P_kA. For example, let $n = 4$ and $k = 1$. Then

$$P_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \end{pmatrix}$$

$$P_1A = \begin{pmatrix} * & * & * & * \\ * & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \end{pmatrix}$$

and

$$P_1AP_1^T = \begin{pmatrix} * & * & * & * \\ * & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \end{pmatrix} = \begin{pmatrix} * & * & * & * \\ * & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \end{pmatrix}$$

Example 5.4.5 Let

$$A = \begin{pmatrix} 0 & 1 & 2 \\ 1 & 2 & 3 \\ 1 & 1 & 1 \end{pmatrix}$$

Because $n = 3$, we have just one step to perform. Form $\widehat{P}_1$:

$$\widehat{P}_1\begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} * \\ 0 \end{pmatrix}$$

$$u_2 = \begin{pmatrix} 1 \\ 1 \end{pmatrix} + \sqrt{2}e_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix} + \sqrt{2}\begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1+\sqrt{2} \\ 1 \end{pmatrix}$$

$$\widehat{P}_1 = I_2 - \frac{2u_2u_2^T}{u_2^Tu_2} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} - 0.2929\begin{pmatrix} 5.8284 & 2.4142 \\ 2.4142 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} -0.7071 & -0.7071 \\ -0.7071 & 0.7071 \end{pmatrix}$$

Form P_1:

$$P_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & & \\ 0 & & \widehat{P}_1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -0.7071 & -0.7071 \\ 0 & -0.7071 & 0.7071 \end{pmatrix}$$

$$A \equiv A^{(1)} = P_1AP_1^T = \begin{pmatrix} 0 & -2.1213 & 0.7071 \\ -1.4142 & 3.5000 & -0.5000 \\ 0 & 1.5000 & -0.5000 \end{pmatrix} = H_u$$

All computations are done using four-digit arithmetic.

Forming the Matrix P and Other Computational Details

1. Each Householder matrix P_k is uniquely determined by the vector u_{n-k} that defines the matrix $\widehat{P}_k$. It is, therefore, sufficient to save the vector u_{n-k}. If the matrix P is needed explicitly, it can be computed from the Householder matrices P_1 through P_{n-2}.
2. The vector $u_{n-k} = (u_{k+1,k}\ldots u_{n,k})^T$ has $n-k$ components, whereas the number of zeros produced at the kth step is $(n-k-1)$. The components 2 through $(n-k)$ of u_{n-k} can be stored in the positions $(k+2, k)$ through (n, k) of A, and the first component is stored separately.
3. The matrix $A^{(k)} = P_kA^{(k-1)}P_k^T$ can be computed without ever forming the matrix P_k explicitly. To see this, consider

$$A^{(k)} = P_kA^{(k-1)}P_k^T = \begin{pmatrix} I_k & 0 \\ 0 & \widehat{P}_k \end{pmatrix}\begin{pmatrix} B & & C \\ 0 & b & D \end{pmatrix}\begin{pmatrix} I_k & 0 \\ 0 & \widehat{P}_k^T \end{pmatrix}$$

Here, B is $k \times k$ Hessenberg, $\widehat{P}_k$ is of order $(n-k)$ Householder matrix, and b is a vector. The vector $\widehat{P}_kb$ is a multiple of e_1. The matrix $\widehat{P}_kD(\widehat{P}_k)^T$ can be computed out of the vector defining $\widehat{P}_k$, as explained earlier (Section 5.4).

4. The nonzero entries of $A^{(k)}$ can be stored over A as soon as the former is formed. Thus, at the end of $(n-2)$nd step, we have

$$A \equiv A^{(n-2)} = \begin{pmatrix} a_{11} & a_{12} & \cdots & \cdots & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & \cdots & \cdots & a_{2n} \\ u_{31} & a_{32} & a_{33} & \cdots & \cdots & a_{3n} \\ \vdots & u_{42} & \ddots & \ddots & \cdots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \cdots \\ u_{n1} & u_{n2} & \cdots & u_{n,n-2} & a_{n,n-1} & a_{nn} \end{pmatrix}$$

The one-dimensional array v containing the first components of the vectors u_{n-k} is given by

$$v = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_{n-2} \end{pmatrix} \equiv \begin{pmatrix} u_{21} \\ u_{32} \\ u_{43} \\ \vdots \\ u_{n-1,n-2} \end{pmatrix}$$

Thus, all the information needed to compute P is contained in the lower triangular part of A below the sub-diagonal and in the single-dimensional array $v = (v_1, v_2, \ldots, v_n)^T$ containing the first entries of the vectors defining P_1 through P_{n-2}.

ALGORITHM 5.4.4 Householder Hessenberg Reduction Given an $n \times n$ matrix A, the following algorithm determines the vectors $u_{n-k} = (u_{k+1,k}, \ldots, u_{nk})^T$, $k = 1, \ldots, n-1$ defining the Householder matrices $\widehat{P}_k$, $k = 1, 2, \ldots, n-2$ (and, therefore, P_k) such that with $P = P_{n-2}P_{n-3}\ldots P_2P_1$, $H_u = PAP^T$ is an upper Hessenberg matrix. The components $u_{k+2,k}$ through $u_{n,k}$ of the vector u_{n-k} are stored in corresponding positions $(k+2, k)$ through (n, k) of A. The first components $u_{k+1,k}$ are stored in a separate array $v = (v_1, v_2, \ldots, v_n)^T$. The nonzero entries of $A^{(k)} = P_kAP_k^T$ overwrite those of A.

For $k = 1, 2, \ldots, n-2$ do

1: Determine the vector $u_{n-k} = (u_{k+1,k}, \ldots, u_{n,k})^T$ defining the Householder matrix

$$\widehat{P}_k = I_{n-k} - 2\frac{u_{n-k}u_{n-k}^T}{(u_{n-k}^T u_{n-k})}$$

of order $(n-k)$ and a scalar σ such that

$$\widehat{P}_k \begin{pmatrix} a_{k+1,k} \\ \vdots \\ \vdots \\ a_{nk} \end{pmatrix} = \begin{pmatrix} \sigma \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

(use Algorithm 5.4.1).

2: Overwrite $a_{k+1,k}$ by σ

$$a_{k+1,k} \equiv \sigma$$

3: Store the vector $u_{n-k} = (u_{k+1,k}, \ldots, u_{n,k})^T$ as follows:

$$v_k \equiv u_{k+1,k}$$
$$a_{k+i,k} \equiv u_{k+i,k}, \quad i = 2, \ldots, n-k$$

4: Compute

$$\beta = \frac{2}{u_{n-k}^T u_{n-k}}$$

5: Update the entries of A in columns $k+1$ through n and rows $k+1$ through n by premultiplication, and then in columns $k+1$ through n and rows 1 to n by postmultiplication.

Premultiplication:
For $j = k+1, \ldots, n$ do
1. $s = \beta \sum_{i=k+1}^{n} u_{ik}a_{ij}$
2. $a_{ij} \equiv a_{ij} - su_{ik} (i = k+1, \ldots, n)$

Postmultiplication:
For $i = 1, 2, \ldots, n$ do
1. $s = \beta \sum_{j=k+1}^{n} a_{ij}u_{jk}$
2. $a_{ij} \equiv a_{ij} - su_{jk} (j = k+1, \ldots, n)$

Example 5.4.6 Transform

$$A = \begin{pmatrix} 1 & 2 & 5 \\ 3 & 7 & 9 \\ 2 & 5 & 3 \end{pmatrix}$$

to an upper Hessenberg matrix H_u using Algorithm 5.4.4, and find P.

There is only one step to be done. First, we find the vector $u_2 = (u_{21}, u_{31})^T$ such that

$$\widehat{P}_1 \begin{pmatrix} 3 \\ 2 \end{pmatrix} = \left(I - \frac{2u_2u_2^T}{u_2^T u_2}\right)\begin{pmatrix} 3 \\ 2 \end{pmatrix} = \begin{pmatrix} \sigma \\ 0 \end{pmatrix}$$

$$u_2 = (2.2019, 0.6667)^T$$
$$\sigma = -3.6056$$

We now update the entries of A, noting that $a_{21} \equiv \sigma = -3.6056$, that $a_{31} = 0$, and that a_{11} remains unchanged.

$$a_{12} \equiv -4.4376, \qquad a_{13} \equiv 3.0509,$$
$$a_{22} \equiv 12.2308, \qquad a_{23} \equiv -2.8462,$$
$$a_{32} \equiv 1.1538, \qquad a_{33} \equiv -2.2308$$

Thus,

$$A \equiv H_u = P_1AP_1^T = \begin{pmatrix} 1.000 & -4.4376 & 3.0509 \\ -3.6056 & 12.2308 & -2.8462 \\ 0 & 1.1538 & -2.2308 \end{pmatrix}$$

Form P_1:

$$\widehat{P}_1 = I - \frac{2u_2u_2^T}{u_2^Tu_2} = \begin{pmatrix} -0.8321 & -0.5547 \\ -0.5547 & 0.8321 \end{pmatrix}$$

Thus,

$$P_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -0.8321 & -0.5547 \\ 0 & -0.5547 & 0.8321 \end{pmatrix}$$

Flop count. The algorithm requires $\frac{5}{3}n^3$ flops to compute H_u. This count does not include the explicit computation of P. P can be stored in factored form. If P is computed explicitly, another $\frac{2}{3}n^3$ flops will be required. However, when n is large, the storage required to form P is prohibitive.

Roundoff property. The algorithm is stable. It can be shown, (Wilkinson 1965, 351) that the computed H_u is orthogonally similar to a nearby matrix $A + E$, where

$$\|E\|_F \le cn^2\mu\,\|A\|_F$$

Here c is a constant of order unity.

If the inner products are accumulated in double precision whenever possible in the algorithm, then the term n^2 in the preceding bound can be replaced by n, so in this case

$$\|E\|_F \le cn\mu\,\|A\|_F$$

which is very desirable.

MATCOM Note: Algorithm 5.4.4 has been implemented in MATCOM program HOUSHESS.The matrix P has been explicitly computed.

Tridiagonal Reduction

If the matrix A is symmetric, then from

$$PAP^T = H_u$$

it follows immediately that the upper Hessenberg matrix H_u is also symmetric and, therefore, is tridiagonal. Thus, if the algorithm is applied to a symmetric matrix A, the resulting matrix H_u will be a symmetric tridiagonal matrix T. Furthermore, we obviously can take advantage of the symmetry of A to modify the algorithm. For example, a significant savings can be made in storage by taking advantage of the symmetry of each $A^{(k)}$.

The symmetric algorithm requires only $\frac{2}{3}n^3$ flops to compute T compared to $\frac{5}{3}n^3$ flops needed to compute H_u. The roundoff property is essentially the same as the nonsymmetric algorithm. *The algorithm is stable.*

Example 5.4.7 Let

$$A = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

Since $n = 3$, we have just one step to perform. Form $\widehat{P}_1$:

$$\widehat{P}_1\begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} * \\ 0 \end{pmatrix}$$

$$u_2 = \begin{pmatrix} 1 \\ 1 \end{pmatrix} + \sqrt{2}e_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix} + \sqrt{2}\begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1+\sqrt{2} \\ 1 \end{pmatrix}$$

$$\widehat{P}_1 = I_2 - \frac{2u_2u_2^T}{u_2^Tu_2} \equiv \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} - .2929\begin{pmatrix} 5.8284 & 2.4142 \\ 2.4142 & 1.0000 \end{pmatrix}$$

$$= \begin{pmatrix} -0.7071 & -0.7071 \\ -0.7071 & 0.7071 \end{pmatrix}$$

Form P_1:

$$P_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & & \\ 0 & & \widehat{P}_1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -0.7071 & -0.7071 \\ 0 & -0.7071 & 0.7071 \end{pmatrix}$$

Thus,

$$H_u = P_1AP_1^T = \begin{pmatrix} 0 & -1.4142 & 0 \\ -1.4142 & 2.5000 & 0.5000 \\ 0 & 0.5000 & 0.5000 \end{pmatrix}$$

(Note that H_u is symmetric tridiagonal.)

5.5 Givens Matrices and Applications to QR Factorization and Hessenberg Reduction

DEFINITION 5.5.1 A matrix of the form

$$J(i,j,c,s) = \begin{array}{c} \\ \\ \\ \\ \\ \\ \\ \\ \end{array}\begin{array}{c} \begin{array}{cccccccc} & & & & i\text{th} & j\text{th} & \text{columns} & \\ & & & & \downarrow & \downarrow & & \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ 0 & 1 & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ \vdots & \vdots & \vdots & & & & \cdots & \vdots \\ 0 & 0 & 0 & \cdots & c & s & \cdots & 0 \\ \vdots & \vdots & \vdots & & & & \cdots & \vdots \\ 0 & 0 & 0 & \cdots & -s & c & \cdots & 0 \\ \vdots & \vdots & \vdots & & & & \cdots & \vdots \\ 0 & 0 & 0 & \cdots & \cdots & 0 & \cdots & 1 \end{pmatrix} \end{array} \begin{array}{l} \\ \\ \\ \leftarrow i\text{th} \\ \\ \leftarrow j\text{th} \\ \\ \\ \end{array} \text{rows}$$

where $c^2 + s^2 = 1$, is called a **Givens matrix**, after the American numerical analyst Wallace Givens (1910–1993).

Since we can choose $c = \cos\theta$ and $s = \sin\theta$ for some θ, a Givens matrix can be conveniently denoted by $J(i, j, \theta)$. Geometrically, the matrix $J(i, j, \theta)$ rotates a pair of coordinate axes (ith unit vector as its x axis and the jth unit vector as its y axis) through the given angle θ in the (i, j) plane. That is why the Givens matrix $J(i, j, \theta)$

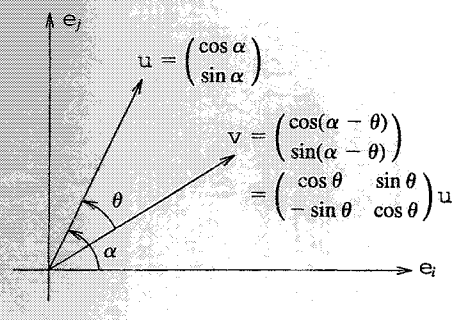


FIGURE 5.2

is commonly known as a *Givens rotation* or *plane rotation in the* (i, j) *plane*. This is illustrated in Figure 5.2.

Thus, when an n-vector

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

is premultiplied by the Givens rotation $J(i, j, \theta)$, only the ith and jth components of x are affected; the other components remain unchanged.

Note that because $c^2 + s^2 = 1$, $J(i, j, \theta) \cdot J(i, j, \theta)^T = I$; thus *the rotation* $J(i, j, \theta)$ *is orthogonal.*

If $x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ is a 2-vector, then it is a matter of simple verification that, with

$$c = \frac{x_1}{\sqrt{x_1^2 + x_2^2}}, \qquad s = \frac{x_2}{\sqrt{x_1^2 + x_2^2}}$$

the Givens rotation

$$J(1, 2, \theta) = \begin{pmatrix} c & s \\ -s & c \end{pmatrix} \quad \text{is such that} \quad J(1, 2, \theta)x = \begin{pmatrix} * \\ 0 \end{pmatrix}$$

The preceding formula for computing c and s might cause some *underflow* or *overflow*. However, the following simple rearrangement of the formula might prevent that possibility.

ALGORITHM 5.5.1 Computing the Givens Parameters The following algorithm computes the Givens parameters c and s of the Givens matrix $J(1, 2, \theta)$ such that given a nonzero 2-vector x, the vector $J(1, 2, \theta)x$ has a zero in the second place.

1. If $|x_2| \geq |x_1|$

$$t = \frac{x_1}{x_2}, \qquad s = \frac{1}{\sqrt{1 + t^2}}, \qquad c = st$$

2. If $|x_2| < |x_1|$

$$t = \frac{x_2}{x_1}, \qquad c = \frac{1}{\sqrt{1 + t^2}}, \qquad s = ct$$

Flop count. The algorithm requires about four flops and one square root.

MATCOM Note: Algorithm 5.5.1 has been implemented in MATCOM program GIVZERO.

Example 5.5.1 Let

$$x = \begin{pmatrix} 1 \\ \frac{1}{2} \end{pmatrix}$$

Since $|x_1| > |x_2|$, we use $t = \frac{1}{2}$, $c = 1/\sqrt{1 + \frac{1}{4}} = 2/\sqrt{5}$, and $s = 1/\sqrt{5}$.

$$\begin{pmatrix} c & s \\ -s & c \end{pmatrix} x = \begin{pmatrix} \frac{2}{\sqrt{5}} & \frac{1}{\sqrt{5}} \\ -\frac{1}{\sqrt{5}} & \frac{2}{\sqrt{5}} \end{pmatrix} \begin{pmatrix} 1 \\ \frac{1}{2} \end{pmatrix} \doteq \begin{pmatrix} \frac{\sqrt{5}}{2} \\ 0 \end{pmatrix}$$

We now show how the premultiplication of a matrix A by a Givens matrix can be performed without computing the Givens matrix explicitly. Similar remarks hold for postmultiplication.

ALGORITHM 5.5.2 Computing Premultiplication by a Givens Matrix Given an $m \times n$ matrix A, Givens parameters c and s, and the indices i and $j (1 \leq i \leq j \leq m)$, the following algorithm overwrites A with $J(i, j, \theta)A$.

For $k = 1, \ldots, n$ do

$$a \equiv a_{ik}$$
$$b \equiv a_{jk}$$
$$a_{ik} \equiv ac + bs$$
$$a_{jk} \equiv -as + bc$$

Flop count. The algorithm requires about $4n$ flops.

MATCOM Note: Algorithm 5.5.2 has been implemented in MATCOM program PGIVMUL.

Zeroing Specified Entries in a Vector

Givens rotations are especially useful in creating zeros in a specified position in a vector. Thus, if

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_i \\ \vdots \\ x_k \\ \vdots \\ x_n \end{pmatrix}$$

and if we desire to zero x_k only, we can construct the rotation $J(i, k, \theta)$ $(i < k)$ such that $J(i, k, \theta)x$ will have zero in the kth position.

To construct $J(i, k, \theta)$, first construct a 2×2 Givens rotation $\begin{pmatrix} c & s \\ -s & c \end{pmatrix}$ such that

$$\begin{pmatrix} c & s \\ -s & c \end{pmatrix}\begin{pmatrix} x_i \\ x_k \end{pmatrix} = \begin{pmatrix} * \\ 0 \end{pmatrix}$$

and then form the matrix $J(i, k, \theta)$ by inserting c in the positions (i, i) and (k, k), inserting s and $-s$, respectively, in the positions (i, k) and (k, i), and filling the rest of the matrix with entries of the identity matrix.

Example 5.5.2 Let

$$x = \begin{pmatrix} 1 \\ -1 \\ 3 \end{pmatrix}$$

Suppose we want to create a zero in the third position, that is, $k = 3$. Choose $i = 2$.

Step 1: Form a 2×2 rotation such that

$$\begin{pmatrix} c & s \\ -s & c \end{pmatrix}\begin{pmatrix} -1 \\ 3 \end{pmatrix} = \begin{pmatrix} * \\ 0 \end{pmatrix}, \qquad c = \frac{-1}{\sqrt{10}}, \qquad s = \frac{3}{\sqrt{10}}$$

Step 2: Form

$$J(2, 3, \theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \frac{-1}{\sqrt{10}} & \frac{3}{\sqrt{10}} \\ 0 & \frac{-3}{\sqrt{10}} & \frac{-1}{\sqrt{10}} \end{pmatrix}$$

Then

$$J(2, 3, \theta)x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \frac{-1}{\sqrt{10}} & \frac{3}{\sqrt{10}} \\ 0 & \frac{-3}{\sqrt{10}} & \frac{-1}{\sqrt{10}} \end{pmatrix}\begin{pmatrix} 1 \\ -1 \\ 3 \end{pmatrix} = \begin{pmatrix} 1 \\ \sqrt{10} \\ 0 \end{pmatrix}$$

Creating Zeros in a Vector Except Possibly in the First Place

Given an n-vector x, if we desire to zero all the entries of x except possibly the first one, we can construct $J(1, 2, \theta)$, $x^{(1)} = J(1, 2, \theta)x$, $x^{(2)} = J(1, 3, \theta)x^{(1)}$, $x^{(3)} = J(1, 4, \theta)x^{(2)}$, and so on, so that with

$$P = J(1, n, \theta) \cdots J(1, 3, \theta)\, J(1, 2, \theta)$$

we will have Px a multiple of e_1. Since each rotation is orthogonal, so is P.

Example 5.5.3 Let

$$x = \begin{pmatrix} 1 \\ -1 \\ 2 \end{pmatrix}$$

$$J(1, 2, \theta) = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$x^{(1)} = J(1, 2, \theta)x = \begin{pmatrix} \sqrt{2} \\ 0 \\ 2 \end{pmatrix}$$

$$J(1, 3, \theta) = \begin{pmatrix} \sqrt{\frac{2}{6}} & 0 & \frac{2}{\sqrt{6}} \\ 0 & 1 & 0 \\ \frac{-2}{\sqrt{6}} & 0 & \sqrt{\frac{2}{6}} \end{pmatrix}$$

Then

$$J(1, 3, \theta)x^{(1)} = \begin{pmatrix} \sqrt{6} \\ 0 \\ 0 \end{pmatrix}$$

$$P = J(1, 3, \theta)J(1, 2, \theta) = \begin{pmatrix} \frac{1}{\sqrt{6}} & \frac{-1}{\sqrt{6}} & \frac{2}{\sqrt{6}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ -\sqrt{\frac{2}{6}} & \sqrt{\frac{2}{6}} & \sqrt{\frac{2}{6}} \end{pmatrix}$$

$$Px = \begin{pmatrix} \sqrt{6} \\ 0 \\ 0 \end{pmatrix}$$

Flop count and roundoff property. Creating zeros in a vector using Givens rotations is about *twice as expensive* as using Householder matrices. To be precise, the process requires only $1\frac{1}{2}$ times flops as Householder's.

The process is as stable as the Householder method.

Creating Zeros in Specified Positions of a Matrix

The idea of creating zeros in specified positions of a vector can be trivially extended to create zeros in specified positions of a matrix as well. Thus, if we wish to create a zero in the (j, i) $[(j > i)]$ position of a matrix A, one way to do this is to construct the rotation $J(i, j, \theta)$ affecting the ith and jth rows only, such that $J(i, j, \theta)A$ will have a zero in the (j, i) position. The procedure then is as follows.

ALGORITHM 5.5.3 Creating Zeros in a Specified Position of a Matrix Using Givens Rotations Given an $n \times n$ matrix A, the following algorithm overwrites A by $J(i, j, \theta)A$ such that the latter has a zero in the (j, i) position.

Step 1: Find $c = \cos(\theta)$, $s = \sin(\theta)$ such that

$$\begin{pmatrix} c & s \\ -s & c \end{pmatrix}\begin{pmatrix} a_{ii} \\ a_{ji} \end{pmatrix} = \begin{pmatrix} * \\ 0 \end{pmatrix}$$

Step 2: Overwrite A with $J(i, j, \theta)A$ by forming the product implicitly as shown in Algorithm 5.5.2

Remark: Note that there are other ways to do this as well. For example, we can form $J(k, j, \theta)$ affecting the kth and jth rows, such that $J(k, j, \theta)A$ will have a zero in the (j, k) position.

MATCOM Note: Algorithm 5.5.3 has been implemented in MATCOM program GIVSZERO.

Example 5.5.4 Let

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 3 & 4 \\ 4 & 5 & 6 \end{pmatrix}$$

Create a zero in the (2, 1) position of A using $J(1, 2, \theta)$.

Step 1: Find c and s such that

$$\begin{pmatrix} c & s \\ -s & c \end{pmatrix}\begin{pmatrix} 1 \\ 3 \end{pmatrix} = \begin{pmatrix} * \\ 0 \end{pmatrix}$$

$$c = \frac{1}{\sqrt{10}}, \qquad s = \frac{3}{\sqrt{10}}$$

Step 2: Form $J(1, 2, \theta)$.

$$J(1, 2, \theta) = \begin{pmatrix} \frac{1}{\sqrt{10}} & \frac{3}{\sqrt{10}} & 0 \\ \frac{-3}{\sqrt{10}} & \frac{1}{\sqrt{10}} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$J(1, 2, \theta)A = \begin{pmatrix} \frac{10}{\sqrt{10}} & \frac{11}{\sqrt{10}} & \frac{15}{\sqrt{10}} \\ 0 & -\frac{3}{\sqrt{10}} & \frac{-5}{\sqrt{10}} \\ 4 & 5 & 6 \end{pmatrix}$$

Example 5.5.5 Let

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 4 & 5 & 6 \end{pmatrix}$$

Create a zero in the (3, 1) position using $J(2, 3, \theta)$.

Step 1: Find c and s such that

$$\begin{pmatrix} c & s \\ -s & c \end{pmatrix}\begin{pmatrix} 2 \\ 4 \end{pmatrix} = \begin{pmatrix} * \\ 0 \end{pmatrix}$$

$$c = \frac{2}{\sqrt{20}}, \qquad s = \frac{4}{\sqrt{20}}$$

Step 2: Form

$$J(2, 3, \theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \frac{2}{\sqrt{20}} & \frac{4}{\sqrt{20}} \\ 0 & \frac{-4}{\sqrt{20}} & \frac{2}{\sqrt{20}} \end{pmatrix}$$

$$A^{(1)} = J(2, 3, \theta)A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \frac{2}{\sqrt{20}} & \frac{4}{\sqrt{20}} \\ 0 & \frac{-4}{\sqrt{20}} & \frac{2}{\sqrt{20}} \end{pmatrix}\begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 4 & 5 & 6 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 2 & 3 \\ \sqrt{20} & \frac{26}{\sqrt{20}} & \frac{32}{\sqrt{20}} \\ 0 & \frac{-2}{\sqrt{20}} & \frac{-4}{\sqrt{20}} \end{pmatrix}$$

5.5.1 Givens Rotations and QR Factorization

It is clear from the foregoing discussion that, like Householder matrices, Givens rotations can also be applied to find QR factorization of a matrix. The Givens method, however, is almost twice as expensive as the Householder method. In spite of this, QR factorization using Givens rotations seems to be particularly useful in QR iterations for eigenvalue computations and in the solution of linear systems with structured matrices, such as Toeplitz. Givens rotations are also emerging as important tools in parallel computations of many important linear algebra problems.

Here we present the algorithm for QR factorization of an $m \times n$ matrix A using Givens' method.

The basic idea is just like Householder's: Compute orthogonal matrices $Q_1, Q_2, \ldots, Q_k$, using Givens rotations such that $A^{(1)} = Q_1 A$ has zeros below the (1, 1) entry in the first column, $A^{(2)} = Q_2 A^{(1)}$ has zeros below the (2, 2) entry in the second column, and so on. Each Q_i is generated as a product of Givens rotations. One way to form $\{Q_i\}$ is:

$$Q_1 = J(1, m, \theta)J(1, m-1, \theta)\cdots J(1, 2, \theta)$$
$$Q_2 = J(2, m, \theta)J(2, m-1, \theta)\cdots J(2, 3, \theta)$$

and so on.

Let $s = \min(n, m-1)$. Then

$$R = A^{(s)} = Q_s A^{(s-1)} = Q_s Q_{s-1} A^{(s-2)} = \cdots$$
$$= Q_s Q_{s-1} \cdots Q_2 Q_1 A = Q^T A$$

We now have $A = QR$ with $Q^T = Q_s Q_{s-1} \cdots Q_2 Q_1$.

THEOREM 5.5.1 Givens QR Factorization Theorem Given an $m \times n$ matrix A, there exist $s = \min(n, m-1)$ orthogonal matrices $Q_1, Q_2, \ldots, Q_s$ defined by $Q_i = J(i, m, \theta) J(i, m-1, \theta) \ldots J(i, i+1, \theta)$ such that if

$$Q = Q_1^T Q_2^T, \ldots, Q_s^T$$

we have

$$A = QR$$

where R is a matrix of the same order as A with zeros below the diagonal.

Forming the Matrix Q and Other Computational Details

In practice there is no need to form the $n \times n$ matrices $J(k, \ell, \theta)$ and $J(k, \ell, \theta)A$ explicitly. Note that $J(k, \ell, \theta)$ can be determined by knowing c and s only, and $J(k, \ell, \theta)A$ replaces the kth and ℓth rows of A by their linear combinations. Specifically, the kth row of $J(k, \ell, \theta)A$ is c times the kth row of A plus s times the ℓth row. Similarly, the ℓth row of $J(k, \ell, \theta)A$ is $-s$ times the kth row of A plus c times the ℓth row. If the orthogonal matrix Q is needed explicitly, then it can be computed from the product

$$Q = Q_1^T Q_2^T \cdots Q_s^T$$

where each Q_i is the product of $m - i$ Givens rotations:

$$Q_i = J(i, m, \theta) J(i, m-1, \theta) \cdots J(i, i+1, \theta)$$

ALGORITHM 5.5.4 QR Factorization Using Givens Rotations

The following algorithm computes the Givens rotations defining an orthogonal matrix Q and a matrix R as in Theorem 5.5.1. The algorithm overwrites A with R.

For $k = 1, 2, \ldots, \min\{n, m-1\}$ do

For $\ell = k+1, \ldots, m$ do

1: Find c and s such that

$$\begin{pmatrix} c & s \\ -s & c \end{pmatrix} \begin{pmatrix} a_{kk} \\ a_{\ell k} \end{pmatrix} = \begin{pmatrix} * \\ 0 \end{pmatrix}$$

2: Save the indices k and ℓ and the numbers c and s

3: Overwrite A with $J(k, \ell, \theta)A$ (using Algorithm 5.5.2).

Flop count. The algorithm requires $2n^2(m - n/3)$ flops. This count, of course, does not include computation of Q. Thus, *the algorithm is almost twice as expensive as the Householder algorithm for QR factorization.* We have assumed here $m \geq n$.

Roundoff property. The algorithm is quite stable. It can be shown that for $m = n$ the computed $\widehat{Q}$ and $\widehat{R}$ satisfy

$$\widehat{R} = \widehat{Q}^T(A + E)$$

where

$$\|E\|_F \leq 6\mu n^{3/2}(1 + 6\mu)^{2n-4}\|A\|_F$$

(Wilkinson 1965, 240).

MATCOM Note: Algorithm 5.5.4 has been implemented in MATCOM program GIVQR. Q and R have been explicitly computed.

5.5.2 Uniqueness in QR Factorization

We have seen that QR factorization of a matrix A always exists and that this factorization may be obtained in different ways. We might, therefore, wonder if this factorization is unique. In the following we will see that *if A is nonsingular, then the factorization is essentially unique*. Moreover, if the diagonal entries of the upper triangular matrices of the factorizations obtained by two different methods are positive, then these two QR factorizations of A are exactly the same.

To see this, let

$$A = Q_1 R_1 = Q_2 R_2 \tag{5.5.1}$$

Because A is nonsingular, then R_1 and R_2 are also nonsingular. Note that

$$\det(A) = \det(Q_1 R_1) = \det(Q_1)\det(R_1)$$

and

$$\det(A) = \det(Q_2 R_2) = \det(Q_2)\det(R_2)$$

Since Q_1 and Q_2 are orthogonal, their determinants are ± 1. Thus $\det R_1$ and $\det R_2$ are different from zero. From (5.5.1), we have

$$Q_2^T Q_1 = R_2 R_1^{-1} = V$$

so that $V^T V = Q_1^T Q_2 Q_2^T Q_1 = I$. Now, V is upper triangular (because R_1^{-1} and R_2 are both upper triangular), so equating elements on both sides, we see that V must be a diagonal matrix with ± 1 as diagonal elements. Thus,

$$R_2 R_1^{-1} = V = \operatorname{diag}(d_1, d_2, \ldots, d_n), \quad \text{with } d_i = \pm 1, i = 1, 2, \ldots, n$$

This means that R_2 and R_1 are the same except for the signs of their rows. Similarly, from

$$Q_2^T Q_1 = V$$

we see that Q_1 and Q_2 are the same except for the signs of their columns.

If the diagonal entries of R_1 and R_2 are positive, then V is the identity matrix, so that

$$R_2 = R_1$$

and

$$Q_2 = Q_1$$

This result can be easily generalized to the case where A is $m \times n$ and has linearly independent columns.

THEOREM 5.5.2 QR Uniqueness Theorem Let A be $m \times n (m \geq n)$ and have linearly independent columns. Then there exists a *unique* $m \times n$ matrix Q with orthonormal columns and a *unique* $n \times n$ upper triangular matrix R with positive diagonal entries such that

$$A = QR$$

For a proof, see Stewart (1973), 214.

Example 5.5.6 We find R of QR factorization of

$$A = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 1 & 1 \end{pmatrix}$$

using Givens rotations and verify the uniqueness of this matrix.

Step 1: Find c and s such that

$$\begin{pmatrix} c & s \\ -s & c \end{pmatrix}\begin{pmatrix} a_{11} \\ a_{21} \end{pmatrix} = \begin{pmatrix} * \\ 0 \end{pmatrix}$$

$$a_{11} = 0, \qquad a_{21} = 1$$

$$c = 0, \qquad s = 1$$

$$J(1,2,\theta) = \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$A \equiv J(1,2,\theta)A = \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 0 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 1 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 2 & 3 \\ 0 & -1 & -1 \\ 1 & 1 & 1 \end{pmatrix}$$

Find c and s such that

$$\begin{pmatrix} c & s \\ -s & c \end{pmatrix}\begin{pmatrix} a_{11} \\ a_{31} \end{pmatrix} = \begin{pmatrix} * \\ 0 \end{pmatrix}$$

$$a_{11} = 1, \qquad c = \frac{1}{\sqrt{2}}, \qquad s = \frac{1}{\sqrt{2}}$$

$$J(1,3,\theta) = \begin{pmatrix} \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\ 0 & 1 & 0 \\ \frac{-1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \end{pmatrix}$$

$$A \equiv J(1,3,\theta)A = \begin{pmatrix} \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\ 0 & 1 & 0 \\ \left(\frac{-1}{\sqrt{2}}\right) & 0 & \frac{1}{\sqrt{2}} \end{pmatrix}\begin{pmatrix} 1 & 2 & 3 \\ 0 & -1 & -1 \\ 1 & 1 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} \sqrt{2} & \frac{3}{\sqrt{2}} & 2\sqrt{2} \\ 0 & -1 & -1 \\ 0 & \frac{-1}{\sqrt{2}} & -\sqrt{2} \end{pmatrix}$$

Step 2: Find c and s such that

$$\begin{pmatrix} c & s \\ -s & c \end{pmatrix}\begin{pmatrix} a_{22} \\ a_{32} \end{pmatrix} = \begin{pmatrix} * \\ 0 \end{pmatrix}$$

$$a_{22} = -1, \qquad a_{32} = -\frac{1}{\sqrt{2}}, \qquad c = -\frac{\sqrt{2}}{\sqrt{3}}, \qquad s = -\frac{1}{\sqrt{3}}$$

$$J(2,3,\theta)A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \frac{-\sqrt{2}}{\sqrt{3}} & -\frac{1}{\sqrt{3}} \\ 0 & \frac{1}{\sqrt{3}} & -\frac{\sqrt{2}}{\sqrt{3}} \end{pmatrix}\begin{pmatrix} \sqrt{2} & \frac{3}{\sqrt{2}} & 2\sqrt{2} \\ 0 & -1 & -1 \\ 0 & -\frac{1}{\sqrt{2}} & -\sqrt{2} \end{pmatrix}$$

$$= \begin{pmatrix} \sqrt{2} & \frac{3}{\sqrt{2}} & 2\sqrt{2} \\ 0 & \frac{\sqrt{3}}{\sqrt{2}} & \frac{2\sqrt{2}}{\sqrt{3}} \\ 0 & 0 & \frac{1}{\sqrt{3}} \end{pmatrix}$$

$$= \begin{pmatrix} 1.4142 & 2.1213 & 2.8284 \\ 0 & 1.2247 & 1.6330 \\ 0 & 0 & 0.5774 \end{pmatrix} = R$$

(using four-digit computations).

Remark: Note that the upper triangular matrix obtained here is *essentially* the same as the one given by the Householder method earlier (Example 5.4.2), differing from it only in the signs of the first and third rows.

5.5.3 Givens Rotations and Reduction to Hessenberg Form

The Givens matrices can also be employed to transform an arbitrary $n \times n$ matrix A to an upper Hessenberg matrix H_u by orthogonal similarity: $PAP^T = H_u$. However, to do this, Givens rotations must be constructed in a certain special manner. For example, in the first step, Givens rotations $J(2, 3, \theta), J(2, 4, \theta), \ldots, J(2, n, \theta)$ are successively computed so that with $P_1 = J(2, n, \theta) \cdots J(2, 4, \theta)J(2, 3, \theta)$,

$$P_1AP_1^T = A^{(1)} = \begin{pmatrix} * & * & \cdots & * \\ * & * & \cdots & * \\ 0 & * & \cdots & * \\ \vdots & \vdots & & \vdots \\ 0 & * & \cdots & * \end{pmatrix}$$

In the second step, Givens rotations $J(3, 4, \theta), J(3, 5, \theta), \ldots, J(3, n, \theta)$ are successively computed so that with $P_2 = J(3, n, \theta)J(3, n-1, \theta) \cdots J(3, 4, \theta)$,

$$P_2A^{(1)}P_2^T = A^{(2)} = \begin{pmatrix} * & * & * & \cdots & * \\ * & * & * & \cdots & * \\ 0 & * & * & \cdots & * \\ 0 & 0 & * & \cdots & * \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & * & \cdots & * \end{pmatrix}$$

and so on. At the end of $(n-2)$nd step, the matrix $A^{(n-2)}$ is the upper Hessenberg matrix H_u. The transforming matrix P is given by

$$P = P_{n-2}P_{n-3}\cdots P_2P_1$$

where $P_i = J(i+1, n, \theta)\,J(i+1, n-1, \theta)\cdots J(i+1, i+2, \theta)$.

Forming the Matrix *P* and Other Computational Details

There is no need to form $J(p+1, q, \theta)$ and $J(p+1, q, \theta)A$ explicitly, since they are completely determined by p, q and c and s (see the section at the end of the QR factorization algorithm using Givens rotations). If P is needed explicitly, it can be formed from

$$P = P_{n-2}\cdots P_2P_1$$

where $P_i = J(i+1, n, \theta)\,J(i+1, n-1, \theta)\cdots J(i+1, i+2, \theta)$.

ALGORITHM 5.5.5 Givens Hessenberg Reduction Given an $n \times n$ matrix A, the following algorithm overwrites A with $PAP^T = H_u$, where H_u is an upper Hessenberg matrix.

For $p = 1, 2, \ldots, n-2$ do

For $q = p+2, \ldots, n$ do

1: Find $c = \cos(\theta)$ and $s = \sin(\theta)$ such that

$$\begin{pmatrix} c & s \\ -s & c \end{pmatrix}\begin{pmatrix} a_{p+1,p} \\ a_{q,p} \end{pmatrix} = \begin{pmatrix} * \\ 0 \end{pmatrix}$$

2: Save c and s and the indices p and q

3: Overwrite A with $J(p+1, q, \theta)\,AJ(p+1, q, \theta)^T$

Flop count. The algorithm requires about $\frac{10}{3}n^3$ flops to compute H_u, compared with $\frac{5}{3}n^3$ required by the Householder method. Thus, *the Givens reduction to Hessenberg form is about twice as expensive as the Householder reduction.* If the matrix A is symmetric, then the algorithm requires about $\frac{4}{3}n^3$ flops to transform A to a symmetric tridiagonal matrix T; again, this is twice as much as required by the Householder method to do the same job.

Roundoff property. The roundoff property is essentially the same as the Householder method. *The method is numerically stable.*

MATCOM Note: Algorithm 5.5.5 has been implemented in MATCOM program GIVHESS. The matrix P has been explicitly computed.

Example 5.5.7

$$A = \begin{pmatrix} 0 & 1 & 2 \\ 1 & 2 & 3 \\ 1 & 1 & 1 \end{pmatrix}$$

Find c and s such that

$$\begin{pmatrix} c & s \\ -s & c \end{pmatrix}\begin{pmatrix} a_{21} \\ a_{31} \end{pmatrix} = \begin{pmatrix} * \\ 0 \end{pmatrix}$$

$$a_{21} = a_{31} = 1, \qquad c = \frac{1}{\sqrt{2}}, \qquad s = \frac{1}{\sqrt{2}}$$

$$J(2, 3, \theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 0 & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix}, \quad A \equiv J(2, 3, \theta)\,AJ(2, 3, \theta)^T$$

$$= \begin{pmatrix} 0 & 2.1213 & 0.7171 \\ 1.4142 & 3.5000 & 0.5000 \\ 0 & -1.5000 & -0.5000 \end{pmatrix}$$

$$= \text{upper Hessenberg}$$

Observation: Note that the Upper Hessenberg matrix obtained here is essentially the same as that obtained by Householder's method (Example 5.4.5); the codiagonal entries differ only in sign.

5.5.4 Uniqueness in Hessenberg Reduction

The preceding example and the observation made therein brings up the question of uniqueness in Hessenberg reduction. To this end, we state a simplified version of what is known as the *implicit Q theorem*. For a complete statement and proof, see Golub and Van Loan (1989, 367–368).

THEOREM 5.5.3 Implicit Q Theorem Let P and Q be orthogonal matrices such that $P^TAP = H_1$ and $Q^TAQ = H_2$ are two *unreduced* upper Hessenberg matrices. Suppose that P and Q have the same first columns. Then H_1 and H_2 are essentially the same in the sense that $H_2 = D^{-1}H_1D$, where

$$D = \text{diag}(\pm 1, \ldots, \pm 1)$$

Example 5.5.8 Consider the matrix

$$A = \begin{pmatrix} 0 & 1 & 2 \\ 1 & 2 & 3 \\ 1 & 1 & 1 \end{pmatrix}$$

once more. The Householder method (Example 5.4.5) gave

$$H_1 = P_1AP_1^T = \begin{pmatrix} 0 & -2.1213 & 0.7171 \\ -1.4142 & 3.5000 & -0.5000 \\ 0 & 1.5000 & -0.5000 \end{pmatrix}$$

The Givens method (Example 5.5.7) gave

$$H_2 = J(2, 3, \theta)AJ(2, 3, \theta)^T = \begin{pmatrix} 0 & 2.1213 & 0.7171 \\ 1.4142 & 3.5000 & 0.5000 \\ 0 & -1.5000 & -0.5000 \end{pmatrix}$$

In the notation of Theorem 5.5.3 we have

$$P = P_1^T, \qquad Q^T = J(2, 3, \theta)$$

Both P and Q have the same first columns, namely, the first column of the identity. We verify that

$$H_2 = D^{-1}H_1D$$

and

$$D = \text{diag}(1, -1, 1)$$

5.6 Orthonormal Bases and Orthogonal Projections

Let A be $m \times n$, where $m \geq n$. Consider the QR factorization of A:

$$Q^TA = \begin{pmatrix} R_1 \\ 0 \end{pmatrix}$$

Assume that A has full rank n. Partition $Q = (Q_1, Q_2)$, where Q_1 has n columns. Then the columns of Q_1 form an orthonormal basis for $R(A)$. Similarly, the columns of Q_2 form an orthonormal basis for the orthogonal complement of $R(A)$. Thus, the matrix

$$P_A = Q_1Q_1^T$$

is the *orthogonal projection onto* $R(A)$ and the matrix $P_A^\perp = Q_2Q_2^T$ is the *projection onto the orthogonal complement of* $R(A)$. Because the orthogonal complement of $R(A)$ is denoted by $R(A)^\perp = N(A^T)$, we shall denote $P_A^\perp$ by P_N.

Example 5.6.1 Let

$$A = \begin{pmatrix} 1 & 1 \\ 0.0001 & 0 \\ 0 & 0.0001 \end{pmatrix}$$

Using the results of Example 5.4.4, we have

$$Q = \begin{pmatrix} -1 & 0.0001 & -0.0001 \\ -0.0001 & -0.7071 & 0.7071 \\ 0 & 0.7071 & 0.7071 \end{pmatrix}$$

$$Q_1 = \begin{pmatrix} -1 & 0.0001 \\ -0.0001 & -0.7071 \\ 0 & 0.7071 \end{pmatrix}, \qquad Q_2 = \begin{pmatrix} -0.0001 \\ 0.7071 \\ 0.7071 \end{pmatrix}$$

$$P_A = Q_1Q_1^T = \begin{pmatrix} 1.000 & 0.0000 & 0.0000 \\ 0.0000 & 0.5000 & -0.5000 \\ 0.0000 & -0.5000 & 0.5000 \end{pmatrix}$$

$$P_N = P_A^\perp = Q_2Q_2^T = \begin{pmatrix} -0.0001 \\ 0.7071 \\ 0.7071 \end{pmatrix}\begin{pmatrix} -0.0001 \\ 0.7071 \\ 0.7071 \end{pmatrix}^T$$

$$= \begin{pmatrix} 0.0000 & -0.0000 & -0.0000 \\ -0.0000 & 0.5000 & 0.5000 \\ -0.0000 & 0.5000 & 0.5000 \end{pmatrix}$$

Projection of a Vector

Given an m-vector b, the vector b_R, the projection of b onto $R(A)$, is given by

$$b_R = P_Ab$$

Similarly, $b_\perp$, the projection of b onto the orthogonal complement of $R(A)$, is given by

$$b_\perp = P_A^\perp b = P_Nb$$

where we denote $P_A^\perp$ by P_N.

Note that $b = b_R + b_\perp$. Again, because the orthogonal complement of $R(A)$ is $N(A^T)$, the null space of A^T, we denote $b_\perp$ by b_N for notational convenience.

Note: Because $b_N = b - b_R$, it is tempting to compute b_N just by subtracting b_R from b once b_R has been computed. This is *not advisable*, because in the computation of b_N from $b_N = b - b_R$, cancellation can take place when $b_R \approx b$.

Example 5.6.2 Let

$$A = \begin{pmatrix} 1 & 2 \\ 0 & 1 \\ 1 & 0 \end{pmatrix}, \qquad b = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

Then

$$Q = \begin{pmatrix} -0.7071 & -0.5774 & -0.4082 \\ 0 & -0.5774 & 0.8165 \\ -0.7071 & 0.5774 & 0.4082 \end{pmatrix}$$

$$Q_1 = \begin{pmatrix} -0.7071 & -0.5744 \\ 0 & -0.5774 \\ -0.7071 & 0.5774 \end{pmatrix}$$

$$P_A = Q_1Q_1^T = \begin{pmatrix} 0.8333 & 0.3333 & 0.1667 \\ 0.3333 & 0.3333 & -0.3333 \\ 0.1667 & -0.3333 & 0.8333 \end{pmatrix}$$

$$b_R = P_Ab = \begin{pmatrix} 1.3333 \\ 0.3333 \\ 0.6667 \end{pmatrix}$$

$$P_N = \begin{pmatrix} 0.1667 & -0.3333 & -0.1667 \\ -0.3333 & 0.6667 & 0.3333 \\ -0.1667 & 0.3333 & 0.1667 \end{pmatrix}$$

$$b_N = P_Nb = \begin{pmatrix} -0.3333 \\ 0.6667 \\ 0.3333 \end{pmatrix}$$

Example 5.6.3 Let

$$A = \begin{pmatrix} 1 & 1 \\ 0.0001 & 0 \\ 0 & 0.0001 \end{pmatrix}, \qquad b = \begin{pmatrix} 2 \\ 0.0001 \\ 0.0001 \end{pmatrix}$$

Then

$$Q = \begin{pmatrix} -1 & -0.0001 & -0.0001 \\ -0.0001 & -0.7071 & 0.7071 \\ 0 & 0.7071 & 0.7071 \end{pmatrix}$$

$$P_A = Q_1 Q_1^T = \begin{pmatrix} 1.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.5000 & -0.5000 \\ 0.0000 & -5.0000 & 0.5000 \end{pmatrix}$$

$$b_R = P_A * b = \begin{pmatrix} 2 \\ 0.0001 \\ 0.0001 \end{pmatrix}$$

$$b_N = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

Projection of a Matrix Onto the Range of Another Matrix

Let $B = (b_1, \ldots, b_n)$ be an $m \times n$ matrix. We can then think of projecting each column of B onto $R(A)$ and onto its orthogonal complement. Thus, the matrix

$$B_R = P_A(b_1, \ldots, b_n) = P_A B$$

is the projection of B onto $R(A)$. Similarly, the matrix

$$B_N = P_N B$$

is the projection of B onto the orthogonal complement of $R(A)$.

Example 5.6.4 Let

$$A = \begin{pmatrix} 1 & 2 \\ 2 & 3 \\ 4 & 5 \end{pmatrix}, \qquad B = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \end{pmatrix}$$

Then $A = QR$ gives

$$Q = \begin{pmatrix} -0.2182 & -0.8165 & -0.5345 \\ -0.4364 & -0.4082 & 0.8018 \\ -0.8729 & 0.4082 & -0.2673 \end{pmatrix}$$

The orthonormal basis for $R(A)$ is

$$Q_1 = \begin{pmatrix} -0.2182 & -0.8165 \\ -0.4364 & -0.4082 \\ -0.8729 & 0.4082 \end{pmatrix}$$

$$P_A = Q_1 Q_1^T = \begin{pmatrix} 0.7143 & 0.4286 & -0.1429 \\ 0.4286 & 0.3571 & 0.2143 \\ -0.1429 & 0.2143 & 0.9286 \end{pmatrix}$$

The orthogonal projection of B onto $R(A)$ is

$$B_R = P_A B = \begin{pmatrix} 1.1429 & 2.1429 & 3.1429 \\ 1.7857 & 2.7857 & 3.7857 \\ 2.0714 & 4.0714 & 5.0714 \end{pmatrix}$$

5.7 QR Factorization with Column Pivoting

If an $m \times n$ $(m \geq n)$ matrix A has rank $r < n$, the QR factorization may not be employed to produce an orthonormal basis of $R(A)$.

To see this, just consider the following simple 2×2 example (Björck, 1994b):

$$A = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} c & -s \\ s & c \end{pmatrix} \begin{pmatrix} 0 & s \\ 0 & c \end{pmatrix} = QR$$

If c and s are chosen such that $c^2 + s^2 = 1$, $\text{rank}(A) = 1 < 2$, and the columns of Q do not form an orthonormal basis of $R(A)$ nor of its complement.

Fortunately, however, the process of QR factorization (for example, the Householder method) can be modified to yield an orthonormal basis. The idea here is to generate a permutation matrix P such that

$$AP = QR$$

where

$$R = \begin{pmatrix} R_{11} & R_{12} \\ 0 & 0 \end{pmatrix}$$

Here R_{11} is $r \times r$ upper triangular, r is the rank of A, and Q is orthogonal. The first r columns of Q will then form an orthonormal basis of $R(A)$. The following theorem guarantees the existence of such a factorization.

THEOREM 5.7.1 QR Column Pivoting Theorem Let A be an $m \times n$ matrix with $\text{rank}(A) = r$. Then there exists an $n \times n$ permutation matrix P and an $m \times m$ orthogonal matrix Q such that

$$Q^T AP = \begin{pmatrix} R_{11} & R_{12} \\ 0 & 0 \end{pmatrix}$$

where R_{11} is an $r \times r$ upper triangular matrix with nonzero diagonal entries.

Proof: Since $\text{rank}(A) = r$, there exists a permutation matrix P such that

$$AP = (A_1, A_2)$$

where A_1 is $m \times r$ and has linearly independent columns. Consider now the QR factorization of A_1:

$$Q^T A_1 = \begin{pmatrix} R_{11} \\ 0 \end{pmatrix}$$

where R_{11} is $r \times r$ upper triangular and has nonzero diagonal entries. Then

$$Q^T AP = (Q^T A_1, Q^T A_2) = \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix}$$

Since $\text{rank}(Q^T AP) = \text{rank}(A) = r$, and $\text{rank}(R_{11}) = r$, we must have $R_{22} = 0$. ■

Creating the Permutation Matrix P

There are several ways to create this permutation matrix P. We present here the outline

The permutation matrix P is formed as the product of r permutation matrices P_1 through P_r. These permutation matrices are applied to A one by one, before forming Householder matrices to create zeros in appropriate columns. More specifically, the following is done:

Step 1: Find the column of A having the *maximum norm*. Permute the columns of A so that the column of maximum norm becomes the first column.

This is equivalent to forming a permutation matrix P_1 such that the matrix AP_1 has the first column having the maximum norm. Form a Householder matrix H_1 so that

$$A^{(1)} = H_1 A P_1$$

has zeros in the first column below the (1, 1) entry.

Step 2: Find the column of the maximum norm of the submatrix $\hat{A}^{(1)}$ obtained from $A^{(1)}$ by deleting the first row and the first column. Permute the columns of $A^{(1)}$ so that the column of $\hat{A}^{(1)}$ of maximum norm becomes the first column of $\hat{A}^{(1)}$. This is equivalent to constructing a permutation matrix $\hat{P}_2$ such that the first column of $\hat{A}^{(1)}\hat{P}_2$ has the maximum norm and then postmultiplying $A^{(1)}$ by a permutation matrix P_2 defined by:

$$P_2 = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & & & \\ \vdots & & \hat{P}_2 & \\ 0 & & & \end{pmatrix}$$

Now construct a Householder matrix H_2 so that

$$A^{(2)} = H_2 A^{(1)} P_2 = H_2 H_1 A P_1 P_2$$

has zeros in the second column of A_2 below the (2, 2) entry. As before, H_2 can be constructed in two steps as in Section 5.4.1:

- The kth step can now easily be written down.
- The process is continued until the entries below the diagonal of the current matrix all become zero.

Suppose r steps are needed. Then at the end of the rth step, we have

$$\begin{aligned} A \equiv A^{(r)} &= H_r \cdots H_1 A P_1 \cdots P_r \\ &= Q^T A P = R = \begin{pmatrix} R_{11} & R_{12} \\ 0 & 0 \end{pmatrix} \end{aligned}$$

Flop count and storage consideration. The preceding method requires $2mnr - r^2(m+n) + 2r^3/3$ flops. The matrix Q, as in the Householder factorization, is stored in factored form. The matrix Q can be stored in factored form in the subdiagonal part of A, and R can overwrite A.

Example 5.7.1 Let

$$A = \begin{pmatrix} 0 & 0 \\ \frac{1}{2} & 1 \\ \frac{1}{2} & 1 \end{pmatrix}_{3\times 2} = (a_1, a_2)$$

a_2 has the largest norm. Then

$$P_1 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

$$AP_1 = \begin{pmatrix} 0 & 0 \\ 1 & \frac{1}{2} \\ 1 & \frac{1}{2} \end{pmatrix}$$

$$H_1 = \begin{pmatrix} 0 & \frac{-1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \\ \frac{-1}{\sqrt{2}} & \frac{1}{2} & \frac{-1}{2} \\ \frac{-1}{\sqrt{2}} & \frac{-1}{2} & \frac{1}{2} \end{pmatrix}$$

$$A^{(1)} = H_1 A P_1 = \begin{pmatrix} 0 & \frac{-1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \\ \frac{-1}{\sqrt{2}} & \frac{1}{2} & \frac{-1}{2} \\ \frac{-1}{\sqrt{2}} & \frac{-1}{2} & \frac{1}{2} \end{pmatrix} \begin{pmatrix} 0 & 0 \\ 1 & \frac{1}{2} \\ 1 & \frac{1}{2} \end{pmatrix} = \begin{pmatrix} -\sqrt{2} & \frac{-1}{\sqrt{2}} \\ 0 & 0 \\ 0 & 0 \end{pmatrix} = R$$

$$R = \begin{pmatrix} R_1 & R_2 \\ 0 & 0 \end{pmatrix}$$

Thus, for this example,

$$Q = H_1^T = H_1$$

$$P = P_1 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

The matrix A has rank 1, because $R_1 = \sqrt{2}$ is 1×1. The column vector

$$\begin{pmatrix} 0 \\ \frac{-1}{\sqrt{2}} \\ \frac{-1}{\sqrt{2}} \end{pmatrix}$$

forms an orthonormal basis of $R(A)$.

Complete Orthogonal Factorization

It is easy to see that the submatrix (R_{11}, R_{12}) can further be reduced by using orthogonal transformations, yielding

$$\begin{pmatrix} T & 0 \\ 0 & 0 \end{pmatrix}$$

THEOREM 5.7.2 Complete Orthogonalization Theorem Given $A_{m\times n}$ with $\text{rank}(A) = r$, there exist orthogonal matrices $Q_{m\times m}$ and $W_{n\times n}$ such that

$$Q^T A W = \begin{pmatrix} T & 0 \\ 0 & 0 \end{pmatrix}$$

where T is an $r \times r$ upper triangular matrix with positive diagonal entries.

Proof: The proof is left as an exercise. ■

This decomposition of A is called the **complete orthogonal decomposition**.

Rank-Revealing QR

The preceding process, known as the *QR factorization with column pivoting*, was proposed by Golub (1965). The factorization is known as the *rank-revealing QR factorization*, since in exact arithmetic it reveals the rank of the matrix A, which is the order of the nonsingular upper triangular matrix R_{11}. However, in the presence of rounding errors, we will actually have

$$R = \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix}$$

and if R_{22} is "small" in some measure (for example, $\|R_{22}\|$ is of $O(\mu)$, where μ is the machine precision), then the reduction will be terminated. Thus, from the preceding discussion, we note that, given an $m \times n$ matrix A ($m \geq n$), if there exists a permutation matrix P such that

$$Q^T AP = R = \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix}$$

where R_{11} is $r \times r$ and R_{22} is small in some measure, then we will say that A has numerical rank r. (For more on numerical rank, see Section 10.5.5.)

Unfortunately, the converse is not true. A celebrated counterexample due to Kahan (1966) shows that a matrix can be nearly rank-deficient without having $\|R_{22}\|$ small at all.

Consider

$$A = \text{diag}(1, s, \ldots, s^{n-1}) \begin{pmatrix} 1 & -c & -c & \cdots & -c \\ & 1 & -c & \cdots & -c \\ & & 1 & & \vdots \\ & & & \ddots & \vdots \\ & & & & 1 \end{pmatrix}$$

with $c^2 + s^2 = 1$, $c, s > 0$. For $n = 100$, $c = 0.2$, it can be shown that A is nearly singular (the smallest singular value is $O(10^{-8})$). On the other hand, $r_{nn} = s^{n-1} = 0.133$, which is not small, so R cannot be nearly singular.

The question whether at any stage R_{22} becomes really small for any matrix has been investigated by Chan (1987) and more recently by Hong and Pan (1992).

5.8 Modifying a QR Factorization

Suppose the QR factorization of an $m \times k$ matrix $A = (a_1, \ldots, a_k)$, $m \geq k$, has been obtained. A vector a_{k+1} is now appended to obtain a new matrix:

$$A' = (a_1, \ldots, a_k, a_{k+1})$$

It is natural to wonder how the QR factorization of A' can be obtained from the given QR factorization of A without finding it from scratch.

The problem is called the *updating QR factorization problem*. The downdating QR factorization is similarly defined. The updating and downdating QR factorization arise in a variety of practical applications, such as signal and image processing.

We present a simple algorithm using Householder matrices to solve the updating problem (Lawson and Hanson, 1974, 174–179).

ALGORITHM 5.8.1 Updating QR Factorization Using Householder Matrices The following algorithm computes the QR factorization of $A' = (a_1, \ldots, a_k, a_{k+1})$ given the Householder QR factorization of $A = (a_1, \ldots, a_k)$.

Step 1: Compute $b_{k+1} = H_k \cdots H_1 a_{k+1}$, where H_1 through H_k are Householder matrices such that

$$Q^T A = H_k \cdots H_1 A = \begin{pmatrix} R_1 \\ 0 \end{pmatrix} = R$$

Step 2: Compute a Householder matrix H_{k+1} so that $H_{k+1} b_{k+1} = r_{k+1}$ is zero in entries $k+2, \ldots, m$

Step 3: Form $R' = \left[\begin{pmatrix} R_1 \\ 0 \end{pmatrix}, r_{k+1} \right]$

Step 4: Form $Q' = H_{k+1} \cdots H_1$

THEOREM 5.8.1 R' and Q' defined above are such that $(Q')^T A' = R'$.

Example 5.8.1 Let

$$A = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

Then

$$H_1 = Q_1^T = \begin{pmatrix} -0.2673 & -0.5345 & -0.8018 \\ -0.5345 & 0.7745 & -0.3382 \\ -0.8018 & -0.3382 & 0.4927 \end{pmatrix}$$

$$R = \begin{pmatrix} -3.7417 \\ 0 \\ 0 \end{pmatrix}$$

$$A' = \begin{pmatrix} 1 & 1 \\ 2 & 4 \\ 3 & 5 \end{pmatrix}$$

Step 1:

$$H_1 = \begin{pmatrix} -0.2673 & -0.5345 & -0.8018 \\ -0.5345 & 0.7745 & -0.3382 \\ -0.8018 & -0.3382 & 0.4927 \end{pmatrix}$$

$$R = \begin{pmatrix} -3.7417 \\ 0 \\ 0 \end{pmatrix}$$

$$b_2 = H_1 a_2 = \begin{pmatrix} -6.4143 \\ 0.8227 \\ 0.3091 \end{pmatrix}$$

Step 2:

$$H_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -0.9426 & -0.3339 \\ 0 & -0.3339 & 0.9426 \end{pmatrix}, \qquad r_2 = \begin{pmatrix} -6.4143 \\ -0.9258 \\ 0 \end{pmatrix}$$

Step 3:

$$R' = (R, r_2) = \begin{pmatrix} -3.7417 & -6.4143 \\ 0 & -0.9258 \\ 0 & 0 \end{pmatrix}$$

$$Q' = H_2 H_1 = \begin{pmatrix} -0.2673 & -0.5345 & -0.8018 \\ 0.7715 & -0.6172 & 0.1543 \\ -0.5773 & -0.5774 & 0.5774 \end{pmatrix}$$

Verification: $(Q')^T R' = \begin{pmatrix} 1 & 1 \\ 2 & 4 \\ 3 & 5 \end{pmatrix} = A'$

5.9 Summary and Table of Comparisons

1. Three important matrices are the elementary, Householder, and Givens matrices.
   (a) *An elementary lower triangular matrix* is an $n \times n$ lower triangular matrix E of the form $E = I + m_k e_k^T$, where $m_k = (0, 0, \ldots, 0, m_{k+1,k}, \ldots, m_{n,k})^T$. If E is as given here, then $E^{-1} = I - m_k e_k^T$.
   (b) *A Householder matrix* is an $n \times n$ matrix $H = I - 2uu^T/u^T u$, where u is an n-vector. A Householder matrix is symmetric and orthogonal.
   (c) *A Givens matrix* $J(i, j, c, s)$ is an identity matrix except for the (i, i), (i, j), (j, i), and (j, j) entries, which are, respectively, c, s, $-s$, and c, where $c^2 + s^2 = 1$. A Givens matrix is an orthogonal matrix.
2. Two important matrix factorizations are LU and QR.
   (a) A factorization of A in the form $A = LU$, where L is unit lower triangular and U is upper triangular, is called an *LU factorization* of A. An LU factorization of matrix A does not always exist. If the leading principal minors of A are all different from zero, then the LU factorization of A exists and is unique (Theorem 5.2.1).

      The LU factorization of a matrix A, when it exists, is achieved using elementary lower triangular matrices. The process is called *Gaussian elimination without row interchanges.*

      The process is efficient, requiring only $n^3/3$ flops, but it is unstable for arbitrary matrices. Its use is not recommended in practice unless A is symmetric positive definite or column diagonally dominant. For decomposition of A into LU in a stable way, row interchanges (Gaussian elimination with partial pivoting) or both row and column interchanges (Gaussian elimination with complete pivoting) to identify an appropriate pivot will be needed. Gaussian elimination with partial and complete pivoting yield factorizations $MA = U$ and $MAQ = U$, respectively.

   (b) Every matrix A can always be written in the form $A = QR$, where Q is orthogonal and R is upper triangular. This is called the *QR factorization* of A. The QR factorization of a matrix A is essentially unique if A is nonsingular (Theorem 5.5.2). The QR factorization can be achieved using either Householder or Givens matrices. Both methods have guaranteed numerical stability. The Householder method is more efficient than the Givens method ($2n^3/3$ flops versus $4n^3/3$ flops (approximately)), but the Givens matrices are emerging as useful tools in parallel matrix computations and for computations with structured matrices. The Gram-Schmidt and modified Gram-Schmidt processes for a different type of QR factorization are described in Chapter 7.
3. Hessenberg reduction: The Hessenberg form of a matrix is a very useful condensed form. We will see its use throughout the whole book. An arbitrary matrix A can always be transformed to an upper Hessenberg matrix by orthogonal similarity: Given an $n \times n$ matrix A, there always exists an orthogonal matrix P such that $PAP^T = H_u$, an upper Hessenberg matrix (Theorem 5.4.2).

   This reduction can be achieved using elementary, Householder, or Givens matrices. We have described methods based on Householder and Givens matrices (Algorithms 5.4.4 and 5.5.5). Both methods have guaranteed stability, but again, the Householder method is more efficient than the Givens method.

   For the aspect of uniqueness in Hessenberg reduction, see the statement of the implicit Q theorem (Theorem 5.5.3). This theorem basically says that if a matrix A is transformed by orthogonal similarity to two different unreduced upper Hessenberg matrices H_1 and H_2 using two transforming matrices P and Q, then H_1 and H_2 are essentially the same, provided that P and Q have the same first columns.
4. Orthogonal bases and orthogonal projections: If $Q^T A = \begin{pmatrix} R_1 \\ 0 \end{pmatrix}$ is the QR factorization of an $m \times n$ matrix ($m \geq n$), then the columns of Q_1 form an orthonormal basis for $R(A)$ and the columns of Q_2 form an orthonormal basis for the orthogonal complement of $R(A)$, where $Q = (Q_1, Q_2)$ and Q_1 has n columns. A is assumed to have full rank.

   If we let $B = (b_1, \ldots, b_n)$ be an $m \times n$ matrix, then the matrix $B_R = P_A(b_1, \ldots, b_n) = P_A B$, where $P_A = Q_1 Q_1^T$, is the orthogonal projection of B onto $R(A)$. Similarly, the matrix $B_N = P_N B$ is the projection of B onto the orthogonal complement of $R(A)$, where $P_N = P_A^{\perp} = Q_2 Q_2^T$.
5. QR factorization with column pivoting, rank-revealing QR, and modifying a QR factorization were introduced. If A is rank-deficient, then the ordinary QR factorization may not produce an orthonormal basis of $R(A)$. In such a case, the QR factorization process needs to be modified. A modification called the QR factorization with column pivoting was discussed in this chapter. Such a factorization always exists (Theorem 5.7.1). A process to achieve such a factorization using Householder matrices due to Golub was described briefly in Section 5.7.

   The QR factorization with column pivoting, in exact arithmetic, reveals the rank of A. That is why it is called rank-revealing QR factorization.

TABLE 5.1 Table of Comparisons

Problem	Method	Flop Count (approximate)	Stability
LU factorization	Gaussian elimination without row interchange	$\frac{mn^2}{2} - \frac{n^3}{6}$	Unstable
Factorization: $MA = U$	Gaussian elimination with partial pivoting	$\frac{mn^2}{2} - \frac{n^3}{6}$	Stable in practice
Factorization: $MAQ = U$	Gaussian elimination with complete pivoting	$\frac{mn^2}{2} - \frac{n^3}{6}$	Stable
QR factorization	Householder	$n^2(m - \frac{n}{3})$	Stable
QR factorization	Givens	$2n^2(m - \frac{n}{3})$	Stable
Hessenberg reduction of an $n \times n$ matrix	Householder	$\frac{5n^3}{3}$	Stable
Hessenberg reduction of an $n \times n$ matrix	Givens	$\frac{10n^3}{3}$	Stable
QR factorization with column pivoting	Householder	$2mnr - r^2(m+n) + \frac{2r^3}{3}$, $r = \text{rank}(A)$	Stable

However, in the presence of rounding errors, such a rank-determination procedure is complicated and not reliable. Finally, we presented a simple algorithm to modify the QR factorization of a matrix (updating QR).

6. Table 5.1 summarizes efficiency and stability properties of some of these major computations. We assume that A is $m \times n$ $(m \geq n)$.
7. Gaussian elimination without pivoting is unstable; Gaussian elimination with partial pivoting is stable in practice; Gaussian elimination with complete pivoting is stable. From Table 5.1, we see that the processes of QR factorization and of reduction to a Hessenberg matrix using Householder transformations are more efficient. The methods are numerically stable. However, as we remarked earlier, Givens transformations are useful in matrix computations with structured matrices, and they are emerging as important tools for parallel matrix algorithms. Also, it is worth noting that Gaussian elimination can be used to transform an arbitrary matrix to an upper Hessenberg matrix by similarity. For details, see Wilkinson (1965, 353–5).

EXERCISES

Note: Use MATLAB whenever appropriate and necessary.

Sections 5.2 and 5.3

1. (a) Show that an elementary lower triangular matrix has the form

$$E = I + m_k e_k^T$$

where $m_k = (0, 0, \ldots, 0, m_{k+1,k}, \ldots, m_{n,k})^T$.

(b) Show that the inverse of E in (a) is given by

$$E^{-1} = I - m_k e_k^T$$

2. (a) Given

$$a = \begin{pmatrix} 0.00001 \\ 1 \end{pmatrix}$$

using five-digit arithmetic, find an elementary matrix E such that Ea is a multiple of e_1.

(b) Using your computations in (a), find the LU factorization of

$$A = \begin{pmatrix} 0.00001 & 1 \\ 1 & 2 \end{pmatrix}$$

(c) Let $\hat{L}$ and $\hat{U}$ be the computed L and U in part (b). Find

$$\frac{\|A - \hat{L}\hat{U}\|_F}{\|A\|_F}$$

where $\|\cdot\|_F$ is the Frobenius norm.

3. Show that the pivots $a_{11}, a_{22}^{(1)}, \ldots, a_{nn}^{(n-1)}$ are nonzero iff the leading principal minors of A are nonzero. *Hint:* Show that

$$\det(A_r) = a_{11} a_{22}^{(1)} \ldots a_{rr}^{(r-1)}$$

4. Let A be a symmetric positive definite matrix. At the end of the first step of the LU factorization of A, we have

$$A^{(1)} = \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ 0 & & & \\ 0 & & & \\ \vdots & & A' & \\ 0 & & & \end{pmatrix}$$

Prove that A' is also symmetric and positive definite. Hence show that LU factorization of a symmetric positive definite matrix using Gaussian elimination without pivoting always exists and is unique.

5. (a) Repeat Exercise 4 when A is a diagonally dominant matrix; that is, show that LU factorization of a diagonally dominant matrix always exists and is unique.

(b) Using (a), show that a diagonally dominant matrix is nonsingular.

6. Assuming that LU factorization of A exists, prove that

(a) A can be written in the form

$$A = LDU_1$$

where D is diagonal and L and U_1 are unit lower and upper triangular matrices, respectively.

(b) If A is symmetric, then

$$A = LDL^T$$

(c) If A is symmetric and positive definite, then

$$A = HH^T$$

where H is a lower triangular matrix with positive diagonal entries. (This is known as the *Cholesky decomposition*.)

7. Assuming that LU factorization of A exists, develop an algorithm to compute U by rows and L by columns directly from the equation

$$A = LU$$

This is known as *Doolittle reduction*.

8. Develop an algorithm to compute the factorization

$$A = LU$$

where U is unit upper triangular and L is lower triangular. This is known as *Crout reduction*. *Hint*: Derive the algorithm from the equation $A = LU$.

9. Compare the Doolittle and Crout reductions with Gaussian elimination with respect to flop count, storage requirements, and possibility of accumulating inner products in double precision.

10. A matrix G of the form

$$G = I - ge_k^T$$

is called a *Gauss–Jordan* matrix. Show that, given a vector x with the property that $e_k^T x \neq 0$, there exists a Gauss-Jordan matrix G such that

$$Gx \text{ is a multiple of } e_k$$

(a) Develop an algorithm to construct Gauss-Jordan matrices $G_1, G_2, \ldots, G_n$ successively such that

$$(G_n G_{n-1}, \ldots, G_2 G_1)A \text{ is a diagonal matrix}$$

This is known as *Gauss-Jordan reduction*.

(b) Derive conditions under which Gauss-Jordan reduction can be carried to completion.

(c) Give the flop count for the algorithm and compare it with those of Gaussian elimination, Crout reduction, and Doolittle reductions.

11. Given

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 5 & 4 \\ 3 & 4 & 5 \end{pmatrix}$$

find the following.

(a) LU factorization of A using Gaussian elimination and Doolittle reduction.

(b) LU factorization of A using Crout reduction (note that U is unit upper triangular and L is lower triangular).

12. Apply the Gauss–Jordan reduction to A of Exercise 11.

13. (a) Let A be $m \times n$ and let $r = \min\{m-1, n\}$. Develop an algorithm to construct elementary matrices $E_1, \ldots, E_r$ such that

$$E_r E_{r-1} \cdots E_2 E_1 A$$

is an upper trapezoidal matrix U. The algorithm should overwrite A with U.

(b) Show that the algorithm requires about $r^3/3$ flops.

(c) Apply the algorithm to

$$A = \begin{pmatrix} 1 & 2 \\ 4 & 5 \\ 6 & 7 \end{pmatrix}$$

14. Given a tridiagonal matrix A with nonzero off-diagonal entries, write down a set of simple conditions on the entries of A that guarantees that Gaussian elimination can be carried to completion.

15. Assuming that LU decomposition exists, from

$$A = LU$$

show that, when A is tridiagonal, L and U are both bidiagonals. Develop a scheme for computing L and U in this case and apply your scheme to find LU factorization of

$$A = \begin{pmatrix} 4 & 1 & 0 & 0 \\ 1 & 4 & 1 & 0 \\ 0 & 1 & 4 & 1 \\ 0 & 0 & 1 & 4 \end{pmatrix}$$

16. Prove that the matrix L in each of the factorizations $PA = LU$ and $PAQ = LU$, obtained by using Gaussian elimination with partial and complete pivoting, respectively, is unit lower triangular.

17. Given

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 2 & 3 & 4 & 5 \end{pmatrix}$$

Find a permutation matrix P, a unit lower triangular matrix L, and an upper triangular matrix U such that $PA = LU$.

18. For each of the following matrices, find

(a) Permutation matrices P_1 and P_2 and elementary matrices M_1 and M_2 such that $MA = M_2 P_2 M_1 P_1 A$ is an upper triangular matrix.

(b) Permutation matrices P_1, P_2, Q_1, Q_2 and elementary matrices M_1 and M_2 such that $MAQ = M_2(P_2(M_1 P_1 A Q_1)Q_2)$ is an upper triangular matrix.

(i) $A = \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \end{pmatrix}$

(ii) $A = \begin{pmatrix} 100 & 99 & 98 \\ 98 & 55 & 11 \\ 0 & 1 & 1 \end{pmatrix}$

(iii) $A = \begin{pmatrix} 1 & 0 & 1 \\ -1 & 1 & 1 \\ -1 & -1 & 1 \end{pmatrix}$

(iv) $A = \begin{pmatrix} 0.0003 & 1.566 & 1.234 \\ 1.5660 & 2.000 & 1.018 \\ 1.2340 & 1.018 & -3.000 \end{pmatrix}$

(v) $A = \begin{pmatrix} 1 & -1 & 0 \\ -1 & 2 & 0 \\ 0 & -1 & 2 \end{pmatrix}$

(c) Express each factorization in the form $PAQ = LU$ (note that for Gaussian elimination without and with partial pivoting, $Q = I$).

(d) Compute the growth factor in each case.

Sections 5.4–5.6

19. Let x be an n-vector. Give an algorithm to compute a Householder matrix

$$H = I - 2\frac{uu^T}{u^Tu}$$

such that Hx has zeros in the positions $(r+1)$ through n; $r < n$. How many flops will be required to implement this algorithm? Given $x = (1, 2, 3)^T$, apply your algorithm to construct H such that Hx has a zero in the third position.

20. Let H be an unreduced upper Hessenberg matrix of order n. Develop an algorithm to triangularize H using

(a) Gaussian elimination

(b) Householder transformations

(c) Givens rotations

Compute the flop count in each case and compare.

21. Let

$$H = \begin{pmatrix} 10 & 1 & 1 & 1 \\ 2 & 10 & 1 & 1 \\ 0 & 1 & 10 & 1 \\ 0 & 0 & 1 & 10 \end{pmatrix}$$

Triangularize H using

(a) Gaussian elimination

(b) Householder method

(c) Givens method

22. Let $\hat{H}_k = I - 2uu^T/u^Tu$, where u is a $(n-k+1)$ vector. Define

$$H_k = \begin{pmatrix} I_{k-1} & 0 \\ 0 & \hat{H}_k \end{pmatrix}$$

How many flops will be required to compute H_kA, where A is arbitrary and $n \times n$? Your count should take into account the special structure of the matrix $\hat{H}_k$.

Using this result, show that the Householder method requires about $\frac{2}{3}n^3$ flops to obtain R and another $\frac{2}{3}n^3$ flops to obtain Q in the QR factorization of A.

23. Show that it requires $n^2(m - n/3)$ flops to compute R in the QR factorization of an $m \times n$ matrix A $(m \geq n)$ using Householder's method.

24. Given

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$$

(a) Find Householder matrices H_1 and H_2 such that

$$H_2H_1A = \begin{pmatrix} R_1 \\ 0 \end{pmatrix}$$

where R_1 is 2×2 upper triangular.

(b) Find orthonormal bases for $R(A)$ and for the orthogonal complement of $R(A)$.

(c) Find orthogonal projections of A onto $R(A)$ and onto its orthogonal complement; that is, find P_A and P_N.

25. Given

$$b = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

and A as in Exercise 24, find b_R and b_N.

26. Given

$$B = \begin{pmatrix} 1 & 3 \\ 2 & 4 \\ 3 & 5 \end{pmatrix}$$

and A as in Exercise 11, find B_R and B_N.

27. Let H be an $n \times n$ upper Hessenberg matrix and let

$$H = QR$$

where Q is orthogonal and R is upper triangular obtained by Givens rotations. Prove that Q is also upper Hessenberg.

28. Develop an algorithm to compute AH, where A is $m \times n$ arbitrary and H is a Householder matrix. How many flops will the algorithm require? Your algorithm should exploit the structure of H.

29. Develop an algorithm to compute AJ, where A is $m \times n$ and J is a Givens rotation. (Your algorithm should exploit the structure of the matrix J.) How many flops are required?

30. Show that the flop count to compute R in the QR factorization of an $m \times n$ matrix A $(m \geq n)$ using Givens rotations is about $2n^2(m - n/3)$.

31. Give an algorithm to compute

$$Q = H_1 H_2 \cdots H_n$$

where H_1 through H_n are Householder matrices in the QR factorization of an $m \times n$ matrix $A; m > n$. Show that the algorithm can be implemented with about $2(m^2 n - mn^2 + n^3/3)$ flops.

32. Show that the orthogonal matrix

$$Q = Q_1^T Q_2^T \cdots Q_{n-1}^T$$

where each Q_i is the product of $(m - i)$ Givens rotations, as in Section 5.5.1, can be computed with $2n(m^2 - n^2/3)$ flops.

33. Let

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

Find QR factorization of A using

(a) the Householder method
(b) the Givens method

Compare the results.

34. Apply both the Householder and the Givens methods of reduction to the matrix

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 2 & 3 & 4 & 5 \end{pmatrix}$$

to reduce it to a Hessenberg matrix by similarity. Compare the results.

35. (a) Show that it requires $\frac{5}{3}n^3$ flops to compute the upper Hessenberg matrix H_u using the Householder method of reduction. *Hint:* $\sum_{k=1}^{n-2} 2(n-k)^2 + \sum_{k=1}^{n-2} 2n(n-k) \approx \frac{2}{3}n^3 + n^3 = 5n^3/3$.

(b) Show that if the transforming matrix P is required explicitly, another $\frac{2}{3}n^3$ flops will be needed.

(c) Work out the corresponding flop counts for reduction to Hessenberg form using Givens rotations.

(d) If A is symmetric, then show that the corresponding count in (a) is $2n^3/3$.

36. Given an unreduced upper Hessenberg matrix H, show that the matrix X defined by $X = (e_1, He_1, \ldots, H^{n-1}e_1)$ is nonsingular and is such that $X^{-1}HX$ is a companion matrix in upper Hessenberg form.

(a) What are the possible numerical difficulties with these computations?

(b) Transform

$$H = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 \times 10^{-5} & 4 & 4 & 4 \\ 0 & 1 \times 10^{-3} & 1 & 2 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

to a companion form.

37. (a) Given the pair (A, b), where A is $n \times n$ and b is a column vector, develop an algorithm to compute an orthogonal matrix P such that PAP^T = upper Hessenberg H_u and Pb is a multiple of the vector e_1.

(b) Show that H_u is unreduced and b is a nonzero multiple of e_1 if and only if rank $(b, Ab, \ldots, A^{n-1}b) = n$.

(c) Apply your algorithm in (a) to

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \\ 2 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}, \qquad b = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix}$$

Sections 5.7 and 5.8

38. Given

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

find a permutation matrix P, an orthogonal matrix Q, and an upper triangular matrix R using the Householder method such that

$$AP = QR$$

The permutation matrix P is to be chosen according to the criteria given in the book.

39. Give a proof of the complete orthogonalization theorem (Theorem 5.7.2).

40. Work out an algorithm to modify the QR factorization of a matrix A from which a column has been removed.

MATLAB AND MATCOM PROGRAMS AND PROBLEMS

Note: You will need the programs **phousmul**, **housqr**, **compiv**, **givqr**, and **givhess** from MATCOM.

1. (a) Write a MATLAB program called **elm**(v) that creates an elementary lower triangular matrix E so that Ev is a multiple of e_1, where v is an n-vector.

(b) Write a MATLAB program **elmul**(A, E) that computes the product EA, where E is an elementary lower triangular matrix and A is an arbitrary matrix. Your program should be able to take advantage of the structure of the matrix E.

(c) Using **elm** and **elmul**, write a MATLAB program **elmlu** that finds the LU factorization of a matrix, when it exists:

$$[L, U] = \textbf{elmlu}(A)$$

Using your program **elmlu**, find L and U for the following matrices:

$$A = \begin{pmatrix} 0.00001 & 1 \\ 1 & 1 \end{pmatrix}, \qquad A = \begin{pmatrix} 1 & 1 \\ 0.00001 & 1 \end{pmatrix}$$

$$A = \begin{pmatrix} 10 & 1 & 1 \\ 1 & 10 & 1 \\ 1 & 1 & 10 \end{pmatrix}, \qquad A = \text{diag}(1, 2, 3)$$

$A = 5 \times 5$ *Hilbert matrix*

Now compute (i) the product LU and (ii) $\|A - LU\|_F$ in each case and print your results.

2. Modify your program **elmlu** to incorporate partial pivoting:

$$[M, U] = \textbf{elmlupp}(A)$$

Test your program with each of the matrices of Problem 1. Compare your results with those obtained by MATLAB program:

$$[L, U] = \textbf{lu}(A)$$

3. Write a MATLAB program called **parpiv** to compute M and U such that $MA = U$, using partial pivoting:

$$[M, U] = \textbf{parpiv}(A)$$

(This program should implement Algorithm 5.2.2)

Print M, U, $\|MA - U\|_F$ and $\|MA - U\|_2$ for each of the matrices A of Problem 1.

4. Using the program **compiv** from MATCOM, print M, Q, and U, and

$$\|MAQ - U\|_2, \text{ and } \|MAQ - U\|_F$$

for each of the matrices of Problem 1.

5. (a) Write a MATLAB program called **houszero** that creates a Householder matrix H such that Ha is a multiple of e_1, where a is an $n-$vector.

(b) Using **houszero** and **phousmul** (from MATCOM), write a MATLAB program **housqr** that finds QR factorization of A:

$$[Q, R] = \textbf{housqr}(A)$$

(This program should implement Algorithm 5.4.3.)

(c) Using your program **housqr**, find Q and R such that

$$A = QR$$

for each of the matrices in Problem 1. Now compute

(i) $\|I - Q^T Q\|_F$

(ii) $\|A - QR\|_F$

and compare the results with those obtained using the MATLAB program

$$[Q, R] = qr(A)$$

(d) Repeat (c) with the program **givqr** from MATCOM, which computes QR factorization using Givens rotations, in place of **housqr**.

6. Run the program **givqr**(A) from MATCOM with each of the matrices in Problem 1. Then using $[Q, R] = qr(A)$ from MATLAB on those matrices again, verify the uniqueness of QR factorization for each A.

7. Using **givhess**(A) from MATCOM and the MATLAB function **hess**(A) on each of the matrices from Problem 1, verify the implicit QR theorems: Theorem 5.5.3 (uniqueness of Hessenberg reduction).

8. Using the results of Problem 5, find an orthonormal basis for $R(A)$, an orthonormal basis for the orthogonal complement of $R(A)$, the orthogonal projection onto $R(A)$, and the projection onto the orthogonal complement of $R(A)$ for each of the matrices of Problem 1.

9. Incorporate *maximum norm column pivoting* in **housqr** to write a MATLAB program **housqrp** that computes the QR factorization with column pivoting of a matrix A. Test your program with each of the matrices in Problem 1:

$$[Q, R, P] = \textbf{housqrp}(A)$$

Compare your results with those obtained by using the MATLAB program

$$[Q, R, P] = qr(A)$$

Note: Some of the programs you have been asked to write, such as **parpiv**, **houszero**, or **housqr**, are in MATCOM or in Appendix B. But it is a good idea to write your own programs.

Suggestions for Further Reading

The topics covered in this chapter are standard and can be found in any numerical linear algebra text. The books by Golub and Van Loan (1983, 1989) and that by G. W. Stewart (1973) are rich sources of further knowledge in this area. We have used them frequently in this chapter.

The book *Matrix Computations* in particular contains a detailed algorithm for QR factorization with column pivoting using Householder transformations.

The book by Lawson and Hanson (1974) contains in-depth discussion of triangularization using Householder and Givens transformations and QR factorization with column pivoting (Chapters 10 and 15).

The details of error analysis of the Householder and the Givens methods for QR factorization and reduction to Hessenberg forms are contained in Wilkinson (1965).

For error analyses of QR factorization using Givens transformations and variants of Givens transformations, see Gentleman (1975).

A nice discussion on orthogonal projections is given in the book *Numerical Linear Algebra and Optimization* by Philip E. Gill, Walter Murray, and Margaret H. Wright (Reading, Mass.: Addison Wesley, 1991).

A recent paper of Chandrasekaran and Ipsen (1994c) contains a full perspective of current research on rank-revealing QR factorization.

For more on updating and downdating problems, see Daniel, Gragg, Kaufman, and Stewart (1976), Gill, Golub, Murray, and Saunders (1974), Bojanczyk, Brent, Van Dooren, and De Hoog (1987), and Nazareth (1989).

Curious readers may wish to see the original papers of Givens (1958) and Householder (1958) on orthogonal (unitary) triangularizations of a matrix.

The biographies of Wallace Givens, Jr., Alston Householder, and James H. Wilkinson, may be found in SIAM News Letter, July, 1993, SIAM News Letter, October, 1993, and SIAM Calendar 1994, respectively.

CHAPTER SIX

Numerical Solutions of Linear Systems

6.1 Introduction

The major objectives of this chapter are to study numerical methods for solving linear systems problems. This study also includes some basic theoretical results on the existence and uniqueness of solutions, discussions of practical applications giving rise to linear systems, and perturbation analysis of the linear systems problems.

The problem of solving the linear system

$$Ax = b$$

arises in a wide variety of applications. As a matter of fact, it might be said that numerical solutions of almost all practical engineering and applied science problems routinely require solution of a linear system problem. (See Section 6.3.)

We discuss methods only for nonsingular linear systems in this chapter. The case where the matrix A is not square or the system has more than one solution is treated in Chapter 7.

A method called *Cramer's rule*, taught in elementary undergraduate linear algebra courses, is of significance from a theoretical point of view.

Cramer's Rule

Let A be a nonsingular matrix of order n and b be an n-vector. The solution x of the system $Ax = b$ is given by

$$x_i = \frac{\det(A_i)}{\det(A)}, \qquad i = 1, \ldots, n$$

where A_i is a matrix obtained by replacing the ith column to A by the vector b and $x = (x_1, x_2, \ldots, x_n)^T$.

Remarks: Cramer's rule is not at all practical from a computational viewpoint. For example, solving a linear system with 20 equations and 20 unknowns by Cramer's rule, using the usual definition of determinant, would require more than a million years even on a fast computer (Forsythe, Malcom, and Moler 1977, 30). For an $n \times n$ system, it will require about $O(n!)$ flops.

Two types of methods are normally used for numerical computations:

1. Direct methods
2. Iterative methods

The direct methods consist of a finite number of steps that all must be performed for any given method before the solution is obtained. On the other hand, iterative methods are based on computing a sequence of approximations to the solution x, and a user can stop whenever a certain desired accuracy is obtained or a certain number of iterations are completed. The iterative methods are used primarily for large and sparse systems. Both direct and iterative methods are studied in this chapter.

In Section 6.2 we state the basic theoretical results (without proofs) on the existence and uniqueness of solutions for linear systems. In Section 6.3 we discuss several engineering applications giving rise to linear systems problems, and Section 6.4 describes direct methods for solving linear systems.

As seen in Chapter 3, the sensitivity of the solution of a linear system problem due to perturbations of the data of the problem is measured by the condition number of the matrix A. In this chapter, we will include details of sensitivity analysis and the effect of the condition number on the accuracy of the computed solution (Sections 6.6 and 6.7).

In many instances, especially in the case of ill-conditioned problems, one needs to refine the computed solution cheaply using the factorization already obtained for computing the original solution. The process is known as iterative refinement technique. We discuss this technique in Section 6.9 following our discussions on componentwise perturbation (Section 6.8). In Section 6.10 we discuss iterative methods: the Jacobi, Gauss-Seidel, SOR, conjugate gradient, and GMRES methods.

Computing the determinant and the inverse of a matrix are seldom needed in practice. Most problems apparently involving inverses can be reformulated in terms of solutions of linear systems problems. Computing the inverse of a matrix is about three times as expensive as solving a linear system. However, there are statistical and engineering applications that call for explicit computation of the inverses, because the entries of the inverse matrix have some physical significance. We, therefore, include a brief discussion on inverse and determinant computations (Section 6.5).

The following major tools and concepts developed earlier will be needed for smooth learning and comprehension of this chapter's material: special matrices (Section 1.4), concepts and results on vector and matrix norms (Section 1.7), convergence of matrix sequence and convergent matrices (Section 1.7.3), LU factorizations (Sections 5.2.1, 5.2.2, 5.2.3), the concept of growth factor (Section 5.3), QR factorization of a matrix (Sections 5.4.1 and 5.5.1), concepts of conditioning and stability (Sections 3.2, 3.3, and 3.4), and some basic knowledge of ordinary and partial differential equations.

6.2 Basic Results on the Existence, Uniqueness, and Invariance of Solutions

Consider the nonhomogeneous system of m equations in n unknowns:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{12}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_m \end{aligned}$$

In matrix form, the system is written as

$$Ax = b$$

where

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}, \quad x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}, \quad b = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}, \quad b \neq 0$$

Given an $m \times n$ matrix A and an m-vector b, if there exists a vector x satisfying $Ax = b$, then we say that the system is **consistent**. Otherwise, it is **inconsistent**. It is natural to ask if a given system $Ax = b$ is consistent and, if it is consistent, how many solutions there are, when the solution is unique, and so on. To this end, we state the following theorem.

THEOREM 6.2.1 Existence and Uniqueness Theorem for a Nonhomogeneous System

1. The system $Ax = b$ is consistent if and only if $b \in R(A)$—in other words, $\text{rank}(A) = \text{rank}(A, b)$.
2. If the system is consistent and the columns of A are linearly independent, then the solution is unique.
3. If the system is consistent and the columns of A are linearly dependent, then the system has an infinite number of solutions.

Homogeneous Systems

If the vector $b = 0$, then the system $Ax = 0$ is called a **homogeneous system**. A homogeneous system always has a solution, namely, $x = 0$. This is the *trivial solution.*

THEOREM 6.2.2 Existence Theorem for a Homogeneous System

1. The system $Ax = 0$ has a nontrivial solution if and only if the columns of A are linearly dependent. If $Ax = 0$ has a nontrivial solution, it has infinitely many solutions.
2. If $m = n$, then $Ax = 0$ has a nontrivial solution if and only if A is singular.

THEOREM 6.2.3 Solution Invariance Theorem A solution of a consistent system

$$Ax = b$$

remains unchanged under any of the following operations:

1. Any two equations are interchanged.
2. An equation is multiplied by a nonzero constant.
3. A nonzero multiple of one equation is added to another equation.

Two systems obtained from one another by applying any of the above operations are called **equivalent systems**. Theorem 6.2.3 then says that *two equivalent systems have the same set of solutions.*

6.3 Some Applications Giving Rise to Linear Systems Problems

It is probably not an overstatement that linear systems problems arise in almost all practical applications. We give examples here from electrical, mechanical, chemical, and civil engineering. We start with a simple problem—an electric circuit.

6.3.1 An Electric Circuit Problem

Consider the diagram of an electrical circuit shown in Figure 6.1. We would like to determine the amount of current between the nodes A_1, A_2, A_3, A_4, A_5, and A_6. Kirchhoff's law on current tells us that *the algebraic sum of all currents entering a node must be zero.* Applying this law at node A_2, we have

$$I_1 - I_2 + I_4 = 0 \tag{6.3.1}$$

At node A_5,

$$I_2 - I_3 - I_4 = 0 \tag{6.3.2}$$

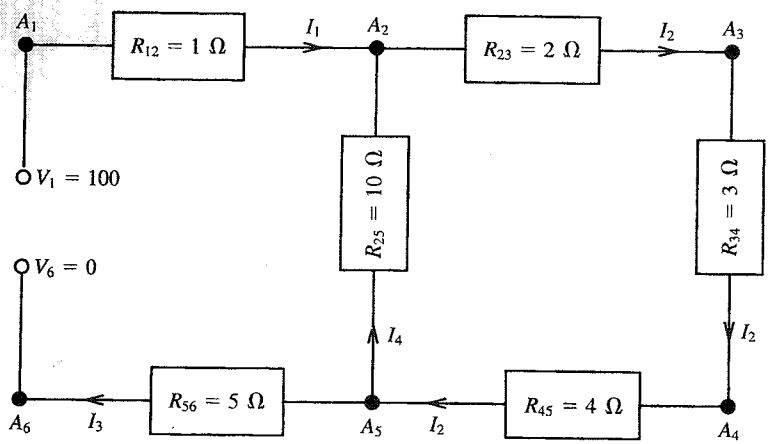


FIGURE 6.1

At node A_3,

$$I_2 - I_2 = 0 \tag{6.3.3}$$

At node A_4,

$$I_2 - I_2 = 0 \tag{6.3.4}$$

Now consider the voltage drop around each closed loop of the circuit, $A_1A_2A_3A_4A_5A_6A_1$, $A_1A_2A_5A_6A_1$, and $A_2A_3A_4A_5A_2$. Kirchhoff's law on voltage tells us that *the net voltage drop around each closed loop is zero.* Thus at the loop $A_1A_2A_3A_4A_5A_6A_1$, substituting the values of resistances and voltages, we have

$$I_1 + 9I_2 + 5I_3 = 100 \tag{6.3.5}$$

Similarly, at $A_1A_2A_5A_6A_1$ and $A_2A_3A_4A_5A_2$, we have, respectively

$$I_1 - 10I_4 + 5I_3 = 100 \tag{6.3.6}$$

$$9I_2 + 10I_4 = 0 \tag{6.3.7}$$

Note that (6.3.6) + (6.3.7) = (6.3.5). Thus we have four equations in four unknowns:

$$I_1 - I_2 + I_4 = 0 \tag{6.3.8}$$

$$I_2 - I_3 - I_4 = 0 \tag{6.3.9}$$

$$I_1 - 10I_4 + 5I_3 = 100 \tag{6.3.10}$$

$$9I_2 + 10I_4 = 0 \tag{6.3.11}$$

Equations (6.3.8)–(6.3.11) can be written as

$$\begin{pmatrix} 1 & -1 & 0 & 1 \\ 0 & 1 & -1 & -1 \\ 1 & 0 & 5 & -10 \\ 0 & 9 & 0 & 10 \end{pmatrix} \begin{pmatrix} I_1 \\ I_2 \\ I_3 \\ I_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 100 \\ 0 \end{pmatrix}$$

the solution of which yields the current between the nodes.

6.3.2 Analysis of a Processing Plant Consisting of Interconnected Reactors

Many mathematical models are based on conservation laws such as conservation of mass, conservation of momentum, and conservation of energy.

As an example, consider a chemical processing plant consisting of six interconnected chemical reactors (Figure 6.2), with different mass flow rates of a component of a mixture into and out of the reactors. We are interested in knowing the concentration of the mixture at different reactors. The example here is similar to that given in Chapra and Canale (1988, pp. 295–8). Application of conservation of mass to all these reactors results in a linear system of equations, as follows, consisting of five equations in five unknowns. The solution of the system will tell us the concentration of the component at each of these reactors.

Steady-state, completely mixed reactor. Consider first a reactor with two flows coming in and one flow going out, as shown in Figure 6.3. Application of the steady-

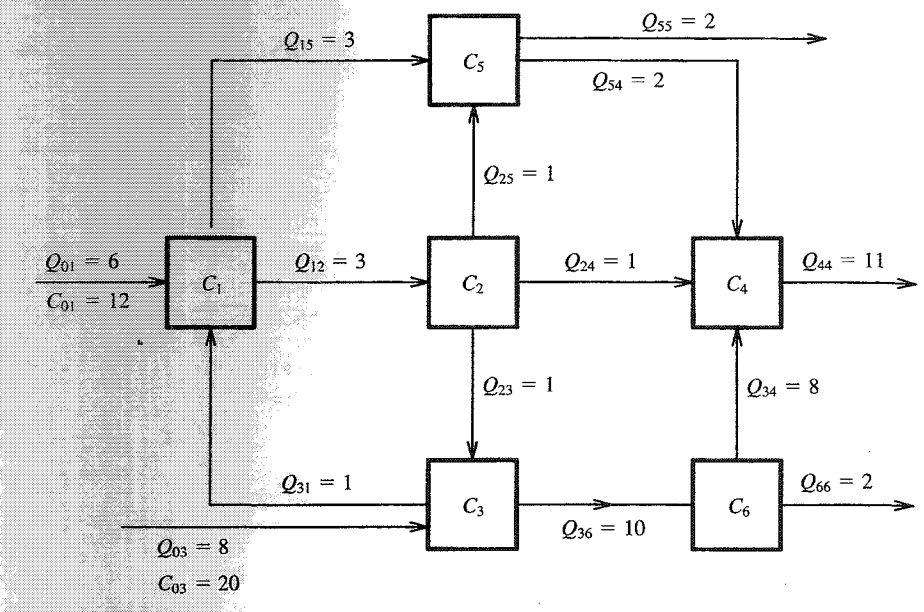


FIGURE 6.2

state conservation of mass for a component in the mixture gives

$$m_1 + m_2 = m_3 \tag{6.3.12}$$

Noting that

$$m_i = Q_i \cdot C_i$$

where

m_i = mass flow rate of the component at the inlet and outlet sections i, $i = 1, 2, 3$ in mg/min

Q_i = volumetric flow rate at the section i, $i = 1, 2, 3$ in /m^3/min

C_i = density or concentration at the section i, $i = 1, 2, 3$ in mg/m^3

we get, from (6.3.12),

$$Q_1C_1 + Q_2C_2 = Q_3C_3 \tag{6.3.13}$$

For given inlet flow rates and concentrations, the outlet concentration C_3 can be found from equation (6.3.13). Under steady state operation, this outlet concentration also

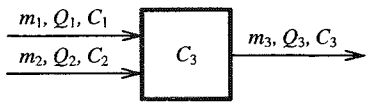


FIGURE 6.3

represents the homogeneous concentration inside the reactor. Such information is necessary for designing the reactor to produce mixtures of a specified concentration. For details, see Chapra and Canale (1988).

Referring now to Figure 6.2, where we consider the plant consisting of six reactors, we have the following equations (a derivation of each of these equations is similar to that of (6.3.13)).

For reactor 1:

$$6C_1 - C_3 = 72 \tag{6.3.14}$$

For reactor 2:

$$3C_1 - 3C_2 = 0 \tag{6.3.15}$$

For reactor 3:

$$-C_2 + 11C_3 = 160 \tag{6.3.16}$$

For reactor 4:

$$C_2 - 11C_4 + 2C_5 + 8C_6 = 0 \tag{6.3.17}$$

For reactor 5:

$$3C_1 + C_2 - 4C_5 = 0 \tag{6.3.18}$$

For reactor 6:

$$10C_3 - 10C_6 = 0 \tag{6.3.19}$$

Equations (6.3.14)–(6.3.19) can be rewritten in matrix form as

$$\begin{pmatrix} 6 & 0 & -1 & 0 & 0 & 0 \\ 3 & -3 & 0 & 0 & 0 & 0 \\ 0 & -1 & 11 & 0 & 0 & 0 \\ 0 & 1 & 0 & -11 & 2 & 8 \\ 3 & 1 & 0 & 0 & -4 & 0 \\ 0 & 0 & 10 & 0 & 0 & -10 \end{pmatrix} \begin{pmatrix} C_1 \\ C_2 \\ C_3 \\ C_4 \\ C_5 \\ C_6 \end{pmatrix} = \begin{pmatrix} 72 \\ 0 \\ 160 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{6.3.20}$$

or

$$AC = D$$

The ith coordinate of the unknown vector C represents the mixture concentration at reactor i of the plant. The solution of the system gives the required concentrations.

6.3.3 Linear Systems Arising from Ordinary Differential Equations (Finite Difference Scheme)

A Case Study on a Spring-Mass Problem

Consider a system of three masses suspended vertically by a series of springs, as shown in Figure 6.4, where k_1, k_2, and k_3 are the spring constants, and x_1, x_2, and x_3 are the displacements of each spring from its equilibrium position.

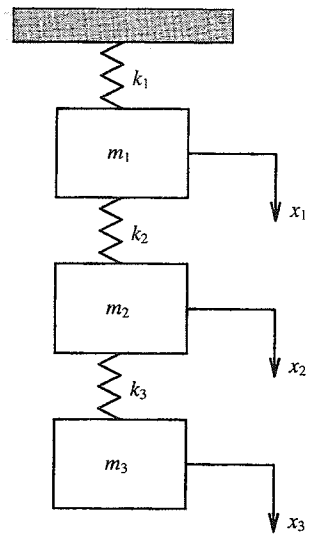


FIGURE 6.4

The free-body diagram for these masses can be represented as shown in Figure 6.5.

Referring to Figure 6.5, the equations of motion, by *Newton's second law*, are:

$$m_1\frac{d^2x_1}{dt^2} = k_2(x_2 - x_1) + m_1g - k_1x_1$$

$$m_2\frac{d^2x_2}{dt^2} = k_3(x_3 - x_2) + m_2g - k_2(x_2 - x_1)$$

$$m_3\frac{d^2x_3}{dt^2} = m_3g - k_3(x_3 - x_2)$$

Suppose we are interested in knowing the displacement of these springs when the system eventually returns to the steady state—that is, when the system comes to rest.

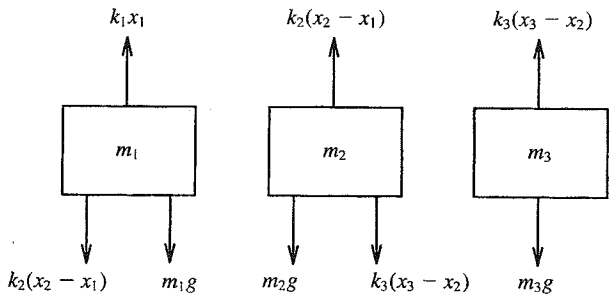


FIGURE 6.5

By setting the second-order derivatives to zero, we obtain

$$k_1x_1 + k_2(x_1 - x_2) = m_1g$$
$$k_2(x_2 - x_1) + k_3(x_2 - x_3) = m_2g$$
$$k_3(x_3 - x_2) = m_3g$$

This system of equations in three unknowns, x_1, x_2, and x_3, can be rewritten in matrix form as

$$\begin{pmatrix} k_1 + k_2 & -k_2 & 0 \\ -k_2 & k_2 + k_3 & -k_3 \\ 0 & -k_3 & k_3 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} m_1g \\ m_2g \\ m_3g \end{pmatrix}$$

or

$$Kx = w$$

The matrix

$$K = \begin{pmatrix} k_1 + k_2 & -k_2 & 0 \\ -k_2 & k_2 + k_3 & -k_3 \\ 0 & -k_3 & k_3 \end{pmatrix}$$

is called the *stiffness matrix*. The solution of the system gives the required displacements.

6.3.4 Linear Systems Arising from Partial Differential Equations: A Case Study on Temperature Distribution

Many engineering problems are modeled by partial differential equations. Numerical approaches to these equations typically require pointwise or piecewise discretization. This process of discretization in turn gives rise to linear systems of many interesting types. We shall illustrate this with a problem in heat transfer theory.

A major objective in a heat transfer problem is to determine the temperature distribution $T(x, y, z, t)$ in a medium resulting from imposed boundary conditions on the surface of the medium. Once this temperature distribution is known, the heat flux or heat transfer rate per unit area at any point in the medium can be computed from Fourier's law, which is expressed as

$$q''_{x_i} = -K\frac{\partial T}{\partial x_i}$$

where x_i is either x, y, or z; q''_{xi} is the heat flux or heat transfer rate per unit area in the x_i direction; $\partial T/\partial x_i$ is the temperature gradient in the x_i direction; and the positive constant K is called the thermal conductivity of the material.

The heat diffusion equation or heat equation that governs this temperature distribution is obtained by applying conservation of energy over an infinitesimally small differential element, from which we obtain the relation in Cartesian coordinates as

$$\frac{\partial}{\partial x}\left(K\frac{\partial T}{\partial x}\right) + \frac{\partial}{\partial y}\left(K\frac{\partial T}{\partial y}\right) + \frac{\partial}{\partial z}\left(K\frac{\partial T}{\partial z}\right) + \dot{q} = \rho C_p \qquad (6.3.21)$$

where ρ is the density, C_p is the specific heat, and $\dot{q}$ is the energy generated per unit volume.

It is often possible to work with a simplified form of equation (6.3.21). For example, if the thermal conductivity, K, is a constant, the heat equation is

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2} + \frac{\dot{q}}{K} = \frac{1}{\alpha}\frac{\partial T}{\partial t} \tag{6.3.22}$$

where $\alpha = K/(\rho C_p)$ is a thermophysical property known as the *thermal diffusivity*.

Under steady-state conditions, there can be no changes of energy storage; that is, the unsteady state term $\partial T/\partial t$ can be dropped, and equation (6.3.21) reduces to the three dimensional *Poisson's equation*:

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2} + \frac{\dot{q}}{K} = 0 \tag{6.3.23}$$

If the heat transfer is two-dimensional (for example, in the x and y directions) and there is no energy generation, then the heat equation reduces to *Laplace's equation*:

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0 \tag{6.3.24}$$

If the heat transfer is unsteady and one-dimensional without energy generation, then the heat equation reduces to

$$\frac{\partial^2 T}{\partial x^2} = \frac{1}{\alpha}\frac{\partial T}{\partial t} \tag{6.3.25}$$

Very often there are practical situations where the geometry or boundary conditions are such that an analytical solution cannot be obtained. In such cases, the best alternatives are *finite difference* or *finite element methods*, which are well suited for computers.

Finite Difference Scheme

A well-known scheme for solving a partial differential equation is to use **finite differences**. The idea is to discretize the partial differential equation by replacing the partial derivatives with their approximations, that is, finite differences. We will illustrate the scheme with Laplace's equation in the following.

Let us divide a two-dimensional region into small regions with increments in the x and y directions given as Δx and Δy, as shown in Figure 6.6. Each nodal point is designated by a numbering scheme i and j, where i indicates x increment and j indicates y increment, as shown in Figure 6.7. The temperature distribution in the medium is assumed to be represented by the nodal points' temperature. The temperature at each nodal point (x_i, y_j) (which is symbolically denoted by (i, j), as in Figure 6.7) is the average temperature of the surrounding hatched region.

A finite difference equation suitable for the interior nodes of a steady two-dimensional system can be obtained by considering Laplace's equation at the nodal point i, j as

$$\left.\frac{\partial^2 T}{\partial x^2}\right|_{i,j} + \left.\frac{\partial^2 T}{\partial y^2}\right|_{i,j} = 0 \tag{6.3.26}$$

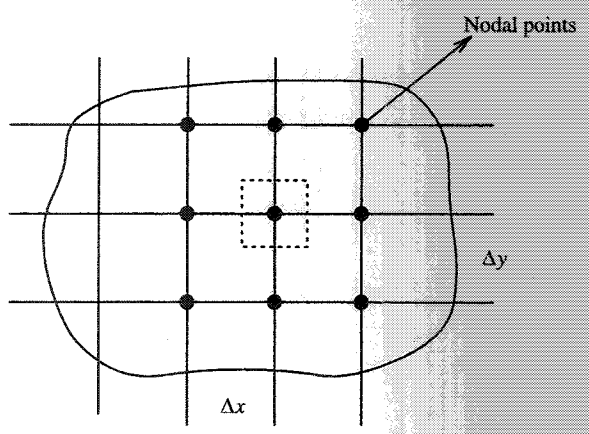


FIGURE 6.6

The second derivatives at the nodal point (i, j) can be approximated (derived from the Taylor series) as

$$\left.\frac{\partial^2 T}{\partial x^2}\right|_{i,j} \cong \frac{T_{i+1,j} - 2T_{i,j} + T_{i-1,j}}{(\Delta x)^2} \tag{6.3.27}$$

$$\left.\frac{\partial^2 T}{\partial y^2}\right|_{i,j} \cong \frac{T_{i,j+1} - 2T_{i,j} + T_{i,j-1}}{(\Delta y)^2} \tag{6.3.28}$$

Equation (6.3.26) then gives

$$\frac{T_{i+1,j} - 2T_{i,j} + T_{i-1,j}}{(\Delta x)^2} + \frac{T_{i,j+1} - 2T_{i,j} + T_{i,j-1}}{(\Delta y)^2} = 0$$

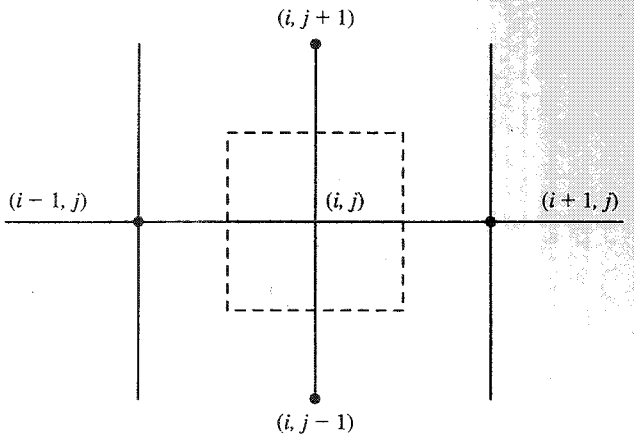


FIGURE 6.7

Assuming $\Delta x = \Delta y$, the *finite difference approximation of Laplace's equation for interior regions* can be expressed as

$$T_{i,j+1} + T_{i,j-1} + T_{i+1,j} + T_{i-1,j} - 4T_{i,j} = 0 \tag{6.3.29}$$

or

$$4T_{i,j} - T_{i-1,j} - T_{i,j-1} - T_{i+1,j} - T_{i,j+1} = 0 \tag{6.3.30}$$

More accurate higher order approximations for interior nodes and boundary nodes are also obtained in a similar manner.

Example 6.3.1 A two-dimensional rectangular plate ($0 \le x \le 1, 0 \le y \le 1$) is subjected to the uniform temperature boundary conditions (with top surface maintained at 100°C and all other surfaces at 0°C) shown in Figure 6.8; that is, $T(0, y) = 0$, $T(1, y) = 0$, $T(x, 0) = 0$, and $T(x, 1) = 100$°C. Suppose we are interested only in the values of the temperature at the nine interior nodal points (x_i, y_j), where $x_i = i\Delta x$ and $y_j = j\Delta y$, $i, j = 1, 2, 3$, with $\Delta x = \Delta y = \frac{1}{4}$. However, we assume symmetry for simplifying the problem. That is, we assume that $T_{3,3} = T_{1,3}$, $T_{3,2} = T_{1,2}$, and $T_{3,1} = T_{1,1}$. We thus have only six unknowns: $(T_{1,1}, T_{1,2}, T_{1,3})$ and $(T_{2,1}, T_{2,2}, T_{2,3})$. From Equation (6.3.30), we then have:

$$\begin{aligned}
4T_{1,1} - 0 - 100 - T_{2,1} - T_{1,2} &= 0\\
4T_{2,1} - T_{1,1} - 100 - T_{1,1} - T_{2,2} &= 0\\
4T_{1,2} - 0 - T_{1,1} - T_{2,2} - T_{1,3} &= 0\\
4T_{2,2} - T_{1,2} - T_{2,1} - T_{1,2} - T_{2,3} &= 0\\
4T_{1,3} - 0 - T_{1,2} - T_{2,3} - 0 &= 0\\
4T_{2,3} - T_{1,3} - T_{2,2} - T_{1,3} - 0 &= 0
\end{aligned}$$

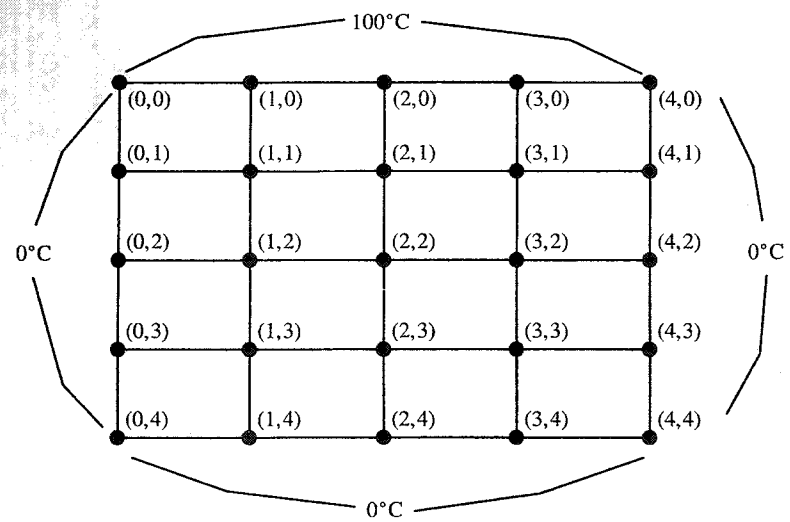


FIGURE 6.8

After suitable rearrangement, these equations can be written in the following form:

$$\begin{pmatrix} 4 & -1 & -1 & 0 & 0 & 0\\ -2 & 4 & 0 & -1 & 0 & 0\\ -1 & 0 & 4 & -1 & -1 & 0\\ 0 & -1 & -2 & 4 & 0 & -1\\ 0 & 0 & -1 & 0 & 4 & -1\\ 0 & 0 & 0 & -1 & -2 & 4 \end{pmatrix} \begin{pmatrix} T_{1,1}\\ T_{2,1}\\ T_{1,2}\\ T_{2,2}\\ T_{1,3}\\ T_{2,3} \end{pmatrix} = \begin{pmatrix} 100\\ 100\\ 0\\ 0\\ 0\\ 0 \end{pmatrix}$$

The solution of this system will give us temperatures at the nodal points.

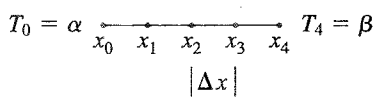
6.3.5 Special Linear Systems Arising in Applications

Many practical applications give rise to linear systems having special properties and structures, such as *tridiagonal*, *diagonally dominant*, *positive definite* and *block tridiagonal*. The solution methods for solving these special systems are described in Section 6.4.7. We first state a situation that gives rise to a tridiagonal system.

Tridiagonal Systems

Consider *one-dimensional* steady conduction of heat through a wire, as shown below. In such a case, the temperature remains constant with respect to time. The equation then is:

$$\frac{\partial^2 T}{\partial x^2} = \dot{q}/K = f(x)$$



Suppose the domain of the problem is $0 \le x \le 1$. Divide the domain into four segments of equal length, say Δx. Thus, $\Delta x = 0.25$. T at $x = i\Delta x$ will be denoted by T_i. Suppose that we know the temperature at the end points $x = 0$ and $x = 1$; that is,

$$T_0 = \alpha$$
$$T_4 = \beta$$

These are then the *boundary conditions* of the problem.

Using a similar numbering scheme as before, the temperature at any given point is given by

$$T_{i+1} - 2T_i + T_{i-1} = (\Delta x)^2 f(x_i)$$

From the equation, the temperature at each node, $x_0 = 0$, $x_1 = \Delta x$, $x_2 = 2\Delta x$, $x_3 = 3\Delta x$, $x_4 = 1$, is calculated as follows:

$$
\begin{array}{lll}
\text{At } x = 0, & T_0 = \alpha & \text{(given)} \\
\text{At } x = \Delta x, & T_0 - 2T_1 + T_2 = (\Delta x)^2 f(\Delta x) & \\
\text{At } x = 2\Delta x, & T_1 - 2T_2 + T_3 = (\Delta x)^2 f(2\Delta x) & \\
\text{At } x = 3\Delta x, & T_2 - 2T_3 + T_4 = (\Delta x)^2 f(3\Delta x) & \\
\text{At } x = 1, & T_4 = \beta & \text{(given)}
\end{array}
$$

In matrix form, these equations can be written as:

$$
\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}
\begin{pmatrix} T_0 \\ T_1 \\ T_2 \\ T_3 \\ T_4 \end{pmatrix}
=
\begin{pmatrix} \alpha \\ (\Delta x)^2 f(\Delta x) \\ (\Delta x)^2 f(2\Delta x) \\ (\Delta x)^2 f(3\Delta x) \\ \beta \end{pmatrix}
$$

The matrix of this system is *tridiagonal*.

Symmetric Tridiagonal and Diagonally Dominant Systems

In order to see how such systems arise, consider now the *unsteady conduction* of heat. This condition implies that the temperature T varies with the time t. The heat equation in this case is

$$\frac{1}{\alpha}\frac{\partial T}{\partial t} = \frac{\partial^2 T}{\partial x^2}$$

Let us divide the grid in the (x, t) plane with spacing Δx in the x-direction and Δt in the t-direction, as in Figure 6.9. Let the temperature at the nodal point $x_i = i\Delta x$ and $t_j = j\Delta t$, as before, be denoted by T_{ij}. Approximating $\partial T/\partial t$ and $\partial^2 T/\partial x^2$ by the implicit finite differences

$$\frac{\partial T}{\partial t} \approx \frac{1}{\Delta t}(T_{i,j+1} - T_{i,j})$$

$$\frac{\partial^2 T}{\partial x^2} \approx \frac{1}{(\Delta x)^2}(T_{i+1,j+1} - 2T_{i,j+1} + T_{i-1,j+1})$$

we obtain the following difference analog of the heat equation:

$$(1 + 2C)T_{i,j+1} - C(T_{i+1,j+1} + T_{i-1,j+1}) = T_{i,j}, \qquad i = 1, 2, \ldots, n$$

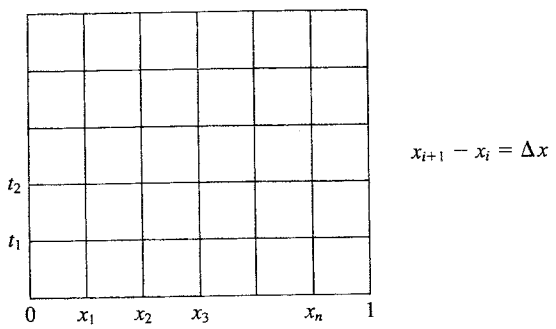


FIGURE 6.9

where

$$C = \alpha\frac{\Delta t}{(\Delta x)^2}$$

These equations enable us to determine the temperature at a time step $j = k + 1$, knowing the temperature at the previous time step $j = k$.

$$
\begin{array}{ll}
\text{For } i = 1,\ j = k: & (1 + 2C)T_{1,k+1} - CT_{2,k+1} = CT_{0,k+1} + T_{1,k} \\
\text{For } i = 2,\ j = k: & (1 + 2C)T_{2,k+1} - CT_{3,k+1} - CT_{1,k+1} = T_{2,k} \\
\vdots & \\
\text{For } i = n,\ j = k: & (1 + 2C)T_{n,k+1} - CT_{n-1,k+1} = T_{n,k} + CT_{n+1,k+1}
\end{array}
$$

Suppose now the temperatures at the two vertical sides are known, that is,

$$T_{0,t} = T_{W_1}$$
$$T_{n+1,t} = T_{W_2}$$

Then the preceding equations can be written in matrix notation as

$$
\begin{pmatrix}
(1+2C) & -C & 0 & \cdots & \cdots & 0 \\
-C & (1+2C) & -C & 0 & \cdots & 0 \\
0 & \ddots & \ddots & \ddots & & \vdots \\
\vdots & & \ddots & \ddots & \ddots & 0 \\
\vdots & & & \ddots & \ddots & -C \\
0 & \cdots & \cdots & 0 & -C & (1+2C)
\end{pmatrix}
\begin{pmatrix} T_{1,k+1} \\ T_{2,k+1} \\ \vdots \\ \vdots \\ T_{n,k+1} \end{pmatrix}
=
\begin{pmatrix} T_{1,k} + CT_{W_1} \\ T_{2,k} \\ \vdots \\ \vdots \\ T_{n,k} + CT_{W_2} \end{pmatrix}
$$

The matrix of this system is clearly *symmetric*, *tridiagonal*, and *diagonally dominant* (note that $C > 0$).

For example, when $C = 1$, and we have

$$
\begin{pmatrix}
3 & -1 & & 0 \\
-1 & \ddots & \ddots & \\
 & \ddots & \ddots & -1 \\
0 & & -1 & 3
\end{pmatrix}
\begin{pmatrix} T_{1,k+1} \\ T_{2,k+1} \\ \vdots \\ \vdots \\ T_{n,k+1} \end{pmatrix}
=
\begin{pmatrix} T_{1,k} + T_{W_1} \\ T_{2,k} \\ \vdots \\ \vdots \\ T_{n,k} + T_{W_2} \end{pmatrix}
$$

or

$$Ax = b$$

The matrix A is **symmetric, tridiagonal, diagonally dominant,** and **positive definite.**

Block Tridiagonal Systems

To see how *block tridiagonal* systems arise in applications, consider the two-dimensional **Poisson's equation**:

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = f(x, y); \qquad 0 \le x \le 1, \qquad 0 \le y \le 1$$

A discrete analog to this equation, similar to Laplace's equation derived earlier, is

$$T_{i+1,j} + T_{i-1,j} + T_{i,j+1} + T_{i,j-1} - 4T_{ij} = (\Delta x)^2 f_{ij};$$
$$i = 1, 2, \ldots, n \qquad j = 1, 2, \ldots, n$$

This will give rise to a linear system of $(n + 2)^2$ variables.

Assume now the values of T at the four sides of the unit square are known and we are interested in the values of T at the interior grid points, that is,

$$T_{0,j}; \quad T_{n+1,j} \quad \text{and} \quad T_{i,0}; \quad T_{i,n+1}$$
$$j = 0, 1, \ldots, n + 1, \qquad i = 0, 1, \ldots, n + 1$$

are given and $T_{11}, \ldots, T_{n1}$; $T_{12}, \ldots, T_{n2}$; and $T_{1n}, \ldots, T_{nn}$ are to be found. Then we have an $(n^2 \times n^2)$ system with n^2 unknowns, which, after suitable arrangements, will be a block tridiagonal system. For example, when $n = 3$, the system will be

$$\begin{pmatrix} 4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 4 & 0 & 0 & -1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 4 & -1 & 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & -1 & 4 & -1 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & -1 & 4 & 0 & 0 & -1 \\ 0 & 0 & 0 & -1 & 0 & 0 & 4 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & -1 & 4 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 4 \end{pmatrix} \times \begin{pmatrix} T_{11} \\ T_{21} \\ T_{31} \\ T_{12} \\ T_{22} \\ T_{32} \\ T_{13} \\ T_{23} \\ T_{33} \end{pmatrix}$$

$$= \begin{pmatrix} T_{01} + T_{10} - (\Delta x)^2 f_{11} \\ T_{20} - (\Delta x)^2 f_{21} \\ T_{30} + T_{41} - (\Delta x)^2 f_{31} \\ T_{02} - (\Delta x)^2 f_{12} \\ -(\Delta x)^2 f_{22} \\ T_{42} - (\Delta x)^2 f_{32} \\ T_{14} + T_{03} - (\Delta x)^2 f_{13} \\ T_{24} - (\Delta x)^2 f_{23} \\ T_{34} + T_{43} - (\Delta x)^2 f_{33} \end{pmatrix}$$

In the general case, the system will be,

$$\begin{pmatrix} A_n & -I_n & & & \\ -I_n & \ddots & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & -I_n \\ & & & -I_n & A_n \end{pmatrix} \begin{pmatrix} T_{11} \\ T_{21} \\ \vdots \\ T_{n1} \\ T_{12} \\ \vdots \\ T_{n2} \\ \vdots \\ T_{nn} \end{pmatrix} = \begin{pmatrix} T_{01} + T_{10} - (\Delta x)^2 f_{11} \\ T_{20} - (\Delta x)^2 f_{21} \\ \vdots \\ T_{n-1,0} - (\Delta x)^2 f_{n-1,1} \\ T_{n+1,1} + T_{n,0} - (\Delta x)^2 f_{n,1} \\ T_{02} - (\Delta x)^2 f_{12} \\ \vdots \end{pmatrix} \tag{6.3.31}$$

where

$$A_n = \begin{pmatrix} 4 & -1 & 0 & \\ -1 & \ddots & \ddots & \\ & \ddots & \ddots & -1 \\ 0 & & -1 & 4 \end{pmatrix} \tag{6.3.32}$$

The system matrix above is **block tridiagonal**, and each matrix A_n on the diagonal is symmetric, tridiagonal, and positive definite.

6.3.6 Linear Systems Arising from Finite Element Methods

We have seen in the last few sections how discretization of differential equations using finite differences gives rise to various types of linear systems problems. The finite element technique is another popular way to discretize differential equations and obtain linear systems of equations. To give some insight, we illustrate this idea by means of a simple differential equation. Interested readers are referred to some of the well-known books on the subject: Celia and Gray (1992), Strang and Fix (1973), Becker, Carey, and Oden (1981), Reddy (1993), and Ames (1992).

Variational Formulation of a Two-point Boundary Value Problem

Let us consider the following two-point boundary value problem:

$$-u'' + u = f(x) \qquad 0 < x < 1 \tag{6.3.33}$$
$$u(0) = u(1) = 0 \tag{6.3.34}$$

where $u' = du/dx$ and f is a continuous function on [0,1]. We further assume that f is such that the problem of (6.3.33)–(6.3.34) has a unique solution.

We introduce the space

$$V = \{v : v \text{ is a continuous function on } [0, 1], \text{ and } v' \text{ is piecewise continuous and bounded on } [0, 1], \text{ and } v(0) = v(1) = 0\}$$

If we now multiply the equation $-u'' + u = f(x)$ by an arbitrary function $v \in V$ (v is called a test function) and integrate the left-hand side by parts, we get

$$\int_0^1 (-u''(x) + u(x))v(x)\,dx = \int_0^1 f(x)v(x)\,dx$$

that is,

$$\int_0^1 (u'v' + uv)\,dx = \int_0^1 f(x)v(x)\,dx \tag{6.3.35}$$

We write (6.3.35) as:

$$a(u, v) = (f, v) \qquad \text{for every } v \in V$$

where

$$a(u, v) = \int_0^1 (u'v' + uv)\,dx$$

and

$$(f, v) = \int_0^1 f(x)v(x)\,dx$$

(Notice that the form $a(\cdot, \cdot)$ is symmetric (that is, $a(u, v) = a(v, u)$) and bilinear.) These two properties will be used later. *It can be shown that u is a solution of (6.3.35) if and only if u is a solution of (6.3.33)–(6.3.34).*

The Discrete Problem

We now discretize the problem in (6.3.35). We start by constructing a finite-dimensional subspace V_n of the space V.

Here, we consider only the simple case where V_n consists of continuous piecewise linear functions. For this purpose, we let $0 = x_0 < x_1 < x_2 < \cdots < x_n < x_{n+1} = 1$ be a partition of the interval $[0, 1]$ into subintervals $I_j = [x_{j-1}, x_j]$ of length $h_j = x_j - x_{j-1}$, $j = 1, 2, \ldots, n + 1$. With this partition, we associate the set V_n with the set of all functions $v(x)$ that are continuous on the interval [0,1], linear in each subinterval I_j, $j = 1, \ldots, n + 1$, and satisfy the boundary conditions $v(0) = v(1) = 0$.

We now introduce the basis functions $\{\phi_1, \phi_2, \ldots, \phi_n\}$, of V_n. We define $\phi_j(x)$ by

1. $\phi_j(x_i) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$
2. $\phi_j(x)$ is a continuous piecewise linear function.

$\phi_j(x)$ can be computed explicitly to yield:

$$\phi_j(x) = \begin{cases} \dfrac{x - x_{j-1}}{h_j}, & \text{when } x_{j-1} \le x \le x_j \\[2ex] \dfrac{x_{j+1} - x}{h_{j+1}}, & \text{when } x_j \le x \le x_{j+1} \end{cases}$$

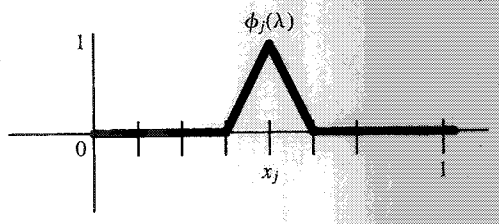


FIGURE 6.10

See Figure 6.10. Because $\phi_1, \ldots, \phi_n$ are the basis functions, any function $v \in V_n$ can be written uniquely as

$$v(x) = \sum_{i=1}^{n} v_i\phi_i(x), \qquad \text{where } v_i = v(x_i)$$

We easily see that $V_n \subset V$.

The discrete analogous of (6.3.35) then reads: Find $u_n \in V_n$ such that

$$a(u_n, v) = (f, v) \qquad \text{for every } v \in V_n \tag{6.3.36}$$

Now, if we let $u_n = \sum_{i=1}^{n} c_i\phi_i(x)$ and notice that equation (6.3.36) is true particularly for every function $\phi_j(x)$, $j = 1, \ldots, n$, we get n equations, namely,

$$a\left(\sum c_i\phi_i, \phi_j\right) = (f, \phi_j) \qquad \text{for every } j = 1, 2, \ldots, n$$

Using the linearity of a $(\cdot, \phi_j)$ leads to n linear equations in n unknowns:

$$\sum_{i=1}^{n} c_i a(\phi_i, \phi_j) = (f, \phi_j) \qquad \text{for every } j = 1, 2, \ldots, n$$

which can be written in the matrix form as

$$Ac = (f_n) \tag{6.3.37}$$

where $(f_n)_i = (f, \phi_i)$ and $A = (a_{ij})$ is a symmetric matrix given by

$$a_{ij} = a_{ji} = a(\phi_i, \phi_j) \quad \text{and} \quad c = (c_1, \ldots, c_n)^T$$

The entries of the matrix A can be computed explicitly: We first notice that

$$a_{ij} = a_{ji} = a(\phi_i, \phi_j) = 0 \qquad \text{if } |i - j| \ge 2$$

(This is due to the local support of the function $\phi_i(x)$). A direct computation now leads to

$$a_{j,j} = a(\phi_j, \phi_j) = \int_{x_{j-1}}^{x_j} \left[\frac{1}{h_j^2} + \frac{(x - x_{j-1})^2}{h_j^2}\right] dx$$

$$+ \int_{x_j}^{x_{j+1}} \left[\frac{1}{h_{j+1}^2} + \frac{(x_{j+1} - x)}{h_{j+1}^2}\right]^2 dx$$

$$= \left[\frac{1}{h_j} + \frac{1}{h_{j+1}}\right] + \frac{1}{3}[h_j + h_{j+1}]$$

$$a_{j,j-1} = \int_{x_{j-1}}^{x_j} \left[-\frac{1}{h_j^2} + \frac{(x_j - x)}{h_j}\frac{(x - x_{j-1})}{h_j}\right] dx = \frac{-1}{h_j} + \frac{h_j}{6}$$

Hence, system 6.3.37 can be written as:

$$\begin{pmatrix} a_1 & b_1 & & 0 \\ b_1 & \ddots & \ddots & \\ & \ddots & \ddots & b_{n-1} \\ 0 & & b_{n-1} & a_n \end{pmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} (f_n)_1 \\ (f_n)_2 \\ \vdots \\ (f_n)_n \end{bmatrix}$$

where $a_j = 1/h_j + 1/h_{j+1} + \frac{1}{3}[h_j + h_{j+1}]$ and $b_j = -1/h_j + h_j/6$. In the special case of uniform grid $h_j = h = 1/(n+1)$, the matrix A then takes the form

$$A = \frac{1}{h}\begin{pmatrix} 2 & -1 & & & 0 \\ -1 & 2 & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & -1 \\ 0 & & & -1 & 2 \end{pmatrix} + \frac{h}{6}\begin{pmatrix} 4 & 1 & 0 & \\ 1 & \ddots & \ddots & \\ & \ddots & \ddots & 1 \\ 0 & & 1 & 4 \end{pmatrix} \tag{6.3.38}$$

6.3.7 Approximation of a Function by a Polynomial: Hilbert System

In Section 3.6 we cited an ill-conditioned linear system with the Hilbert matrix. In this section we show how such a system arises. The discussion here has been taken from Forsythe and Moler (1967, 80–81).

Suppose a continuous function $f(x)$ defined on the interval $0 \le x \le 1$ is to be approximated by a polynomial of degree $n - 1$:

$$\sum_{i=1}^{n} p_i x^{i-1}$$

such that the error

$$E = \int_0^1 \left[\sum_{i=1}^{n} p_i x^{i-1} - f(x)\right]^2 dx$$

is minimized. The coefficients p_i of the polynomial are easily determined by setting

$$\frac{\partial E}{\partial p_i} = 0, \qquad i = 1, \ldots, n$$

(Note that the error is a differentiable function of the unknowns p_i and that a minimum occurs when all the partial derivatives are zero.) Thus, we have

$$\frac{\partial E}{\partial p_i} = 2\int_0^1 \left[\sum_{j=1}^{n} p_j x^{j-1} - f(x)\right] x^{i-1}\, dx = 0, \qquad i = 1, \ldots, n$$

or

$$\sum_{j=1}^{n}\left(\int_0^1 x^{i+j-2}\, dx\right) p_j = \int_0^1 f(x)x^{i-1}\, dx, \qquad i = 1, \ldots, n$$

(To obtain the latter form we have interchanged the summation and integration.)

Letting

$$h_{ij} = \int_0^1 x^{i+j-2}\, dx$$

and

$$b_i = \int_0^1 f(x)x^{i-1}\, dx \qquad (i = 1, 2, \ldots, n)$$

we have

$$\sum_{j=1}^{n} h_{ij} p_j = b_i, \qquad i = 1, \ldots, n$$

That is, we obtain the linear system

$$Hp = b$$

where

$$H = (h_{ij}), \qquad b = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

The matrix H is easily identified as the **Hilbert matrix** (see Section 3.6), because

$$h_{ij} = \int_0^1 x^{i+j-2}\, dx = \frac{1}{i + j - 1}$$

6.4 Direct Methods

In this section we study direct methods for solving the problem $Ax = b$. These methods include the following:

- Gaussian elimination without pivoting, based on LU factorization of A (Section 6.4.2)
- Gaussian elimination with partial pivoting, based on $MA = U$ factorization of A (Section 6.4.3)
- Gaussian elimination with complete pivoting, based on $MAQ = U$ factorization of A (Section 6.4.3)
- The method based on the QR factorization of A (Section 6.4.5)
- The method based on the Cholesky decomposition of a symmetric positive definite matrix (Section 6.4.7)
- Gaussian elimination for special systems: Hessenberg, positive definite, tridiagonal, and diagonally dominant. (Section 6.4.7)

For a comparison of these methods and their relative merits and demerits, see Section 6.4.9 and the accompanying tables.

The basic idea behind all the direct methods is first to reduce the linear system $Ax = b$ to an equivalent triangular system by finding triangular factors of the matrix A and then to solve the triangular system, which is much easier to solve than the original problem. In Chapter 5 we described various methods for computing triangular factors of A. In Section 3.1 we described the back-substitution method for solving an upper triangular system. We now describe an analogous method, called *forward elimination*, for solving a lower triangular system.

6.4.1 Solution of a Lower Triangular System

The solution $y = (y_1, y_2, \ldots, y_n)^T$ of the nonsingular lower triangular system

$$Ly = b$$

can be obtained analogously to the upper triangular system. Here y_1 is obtained from the first equation and then, inserting its value in the second equation, y_2 is obtained, and so on. This process is called **forward elimination**.

ALGORITHM 6.4.1 Forward Elimination

For $i = 1, 2, \ldots, n$ do

$$y_i = \frac{1}{l_{ii}}\left(b_i - \sum_{j=1}^{i-1} l_{ij}y_j\right)$$

Note: When $i = 1$, the summation ($\sum$) is skipped.

Flop count and stability. The algorithm requires about $n^2/2$ flops (1 flop to compute y_1, 2 flops to compute y_2, 3 flops to compute y_3, and so on). The algorithm is as stable as the back-substitution process (Algorithm 3.1.3).

MATCOM Note: Algorithm 6.4.1 has been implemented in MATCOM program FORELM.

6.4.2 Solution of the System $Ax = b$ Using Gaussian Elimination without Pivoting

The Gaussian elimination method for the linear system $Ax = b$ is based on the LU factorization of the matrix A. Recall from Section 5.2.1 that triangularization using Gaussian elimination without pivoting, when carried out to completion, yields a LU factorization of A. Once we have this factorization of A, the system $Ax = b$ becomes equivalent to two triangular systems:

$$Ly = b$$
$$Ux = y$$

Solving $Ax = b$ using Gaussian Elimination without Pivoting
The solution of the system $Ax = b$ using Gaussian elimination (without pivoting) can be achieved in two steps:

Step 1: Find an LU factorization of A.

Step 2: Solve two triangular systems: the lower triangular system $Ly = b$ first, followed by the upper triangular system $Ux = y$.

Flop count. Since an LU factorization requires about $n^3/3$ flops and the solution of each triangular system needs only $n^2/2$ flops, the total flop count for solving the system $Ax = b$ using Gaussian elimination is about $n^3/3 + n^2$.

6.4.3 Solution of $Ax = b$ Using Pivoting Triangularization

Solution with Partial Pivoting

If Gaussian elimination with partial pivoting is used to triangularize A, then—as we saw in Section 5.2.2—this process yields the factorization:

$$MA = U$$

In this case, the system $Ax = b$ is equivalent to the triangular system

$$Ux = Mb = b'$$

Solving $Ax = b$ Using Gaussian Elimination with Partial Pivoting
To solve $Ax = b$ using Gaussian elimination with partial pivoting:

Step 1: Find the factorization $MA = U$ by the triangularization algorithm using partial pivoting (Algorithm 5.2.2).

Step 2: Solve the triangular system by back substitution (Algorithm 3.1.3):

$$Ux = Mb = b'$$

Implementation of Step 2. The vector

$$b' = Mb = M_{n-1}P_{n-1}M_{n-2}P_{n-2}\cdots M_1P_1b$$

is given as follows:

Step 1: $s_1 = b = (b_1, b_2, \ldots, b_n)^T$

Step 2: For $k = 1, 2, \ldots, n-1$ do

$$s_{k+1} = M_kP_ks_k$$

Computational remarks: Algorithm 5.2.2 does not give M_k and P_k explicitly. But, we really do not need them. The vector s_{k+1} can be computed immediately from s_k once the index r_k of row interchange and the multipliers m_{ik} have been saved at the kth step. No explicit matrix multiplication is needed, as shown in Section 5.2.2. This is illustrated with a 3×3 example.

Example 6.4.1 Let

$$n = 3, \qquad P_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}, \qquad M_1 = \begin{pmatrix} 1 & 0 & 0 \\ m_{21} & 1 & 0 \\ m_{31} & 0 & 1 \end{pmatrix}$$

and let

$$s_2 = M_1 P_1 s_1 = \begin{pmatrix} s_1^{(2)} \\ s_2^{(2)} \\ s_3^{(2)} \end{pmatrix}$$

Then we have

$$P_1 s_1 = \begin{pmatrix} b_1 \\ b_3 \\ b_2 \end{pmatrix}$$

and the entries of s_2 are then given by

$$\begin{aligned} s_1^{(2)} &= b_1 \\ s_2^{(2)} &= m_{21} b_1 + b_3 \\ s_3^{(2)} &= m_{31} b_1 + b_2 \end{aligned}$$

Solution with Complete Pivoting

Gaussian elimination with complete pivoting (Section 5.2.3) gives

$$MAQ = U$$

Using this factorization, the system $Ax = b$ can be written as

$$Uy = Mb = b'$$

where

$$y = Q^T x$$

Thus, we have the following.

Solving $Ax = b$ Using Gaussian Elimination with Complete Pivoting

To solve $Ax = b$ using complete pivoting:

Step 1: Find the factorization $MAQ = U$ by the factorization algorithm using complete pivoting (Algorithm 5.2.3).

Step 2: Solve the triangular system (for y) (Algorithm 3.1.3):

$$Uy = b'$$

computing b' as shown previously.

Step 3: Finally, recover x from y: $x = Qy$.

Implementation of Step 3. Because $x = Qy = Q_1 Q_2 \cdots Q_{n-1} y$, the following scheme can be adopted to compute x from y in Step 3.

Step 1: Set $w_n = y$
Step 2: For $k = n-1, \ldots, 2, 1$ do
$\quad w_k = Q_k w_{k+1}$
Step 3: Then $x = w_1$

Note: Because Q_k is a permutation matrix, the entries of w_k are simply those of w_{k+1} reordered according to the permutation index.

Example 6.4.2 Solve

$$Ax = b$$

with

$$A = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 1 & 1 \end{pmatrix}, \qquad b = \begin{pmatrix} 2 \\ 6 \\ 3 \end{pmatrix}$$

(a) using partial pivoting and (b) using complete pivoting.

(a) Partial pivoting: With the results obtained earlier (Section 5.2.2, Example 5.2.3), we compute

$$P_1 b = \begin{pmatrix} 6 \\ 2 \\ 3 \end{pmatrix}; \qquad P_1 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$M_1 P_1 b = \begin{pmatrix} 6 \\ 2 \\ -3 \end{pmatrix}; \qquad M_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix}$$

$$P_2 M_1 P_1 b = \begin{pmatrix} 6 \\ 2 \\ -3 \end{pmatrix}; \qquad P_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$b' = M_2P_2M_1P_1b = \begin{pmatrix} 6 \\ 2 \\ -1 \end{pmatrix}; \qquad M_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$$

The solution of the system

$$Ux = b'$$

$$\begin{pmatrix} 1 & 2 & 3 \\ 0 & 1 & 1 \\ 0 & 0 & -1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 6 \\ 2 \\ -1 \end{pmatrix}$$

is $x_1 = x_2 = x_3 = 1$.

(b) Complete pivoting: Using the results of Example 5.2.5, we have

$$P_1b = \begin{pmatrix} 6 \\ 2 \\ 3 \end{pmatrix}, \qquad P_1 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$M_1P_1b = \begin{pmatrix} 6 \\ 0 \\ 1 \end{pmatrix}, \qquad P_2M_1P_1b = \begin{pmatrix} 6 \\ 1 \\ 0 \end{pmatrix}$$

$$b' = M_2P_2M_1P_1b = \begin{pmatrix} 6 \\ 1 \\ \frac{1}{2} \end{pmatrix}$$

The solution of the system

$$Uy = b'$$

$$\begin{pmatrix} 3 & 1 & 2 \\ 0 & \frac{2}{3} & \frac{1}{3} \\ 0 & 0 & \frac{1}{2} \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 6 \\ 1 \\ \frac{1}{2} \end{pmatrix}$$

is $y_1 = y_2 = y_3 = 1$. Because $\{x_k\}$, $k = 1, 2, 3$ is simply the rearrangement of $\{y_k\}$, we have $x_1 = x_2 = x_3 = 1$.

Some Details of Implementation

Note that it is not necessary to store the vectors s_i and w_i separately, because all we need is the vector s_n for partial pivoting and w_1 for complete pivoting. So starting with $x = b$, each new vector can be stored in x as it is computed.

Also note that if we use the practical algorithms, the matrices P_k, Q_k, and M_k are not available explicitly; they are given by the indices r_k, s_k and the multipliers m_{ik}. The s_k's and w_k's are to be computed out of these.

Flop count. We saw in Section 5.2.1 that the triangularization process using elementary matrices requires $n^3/3$ flops. The triangular system $Ux = b'$ or $Uy = b'$ can be solved using back substitution with $n^2/2$ flops, and the vector b' can be computed with n^2 flops, taking into account the special structures of the matrices M_k and P_k.

Recovering x from y in step 3 of the complete pivoting process does not need any flops. Note that x is obtained from y just by reshuffling the entries of y. Thus, the solution of the linear system $Ax = b$ using Gaussian elimination with complete or partial pivoting requires $n^3/3 + O(n^2)$ flops. However, Gaussian elimination with complete pivoting requires about $n^3/3$ comparisons to identify $(n-1)$ pivots, compared to only $O(n^2)$ comparisons needed by the partial pivoting method.

Roundoff property. In Section 5.3 we discussed the roundoff property of Gaussian elimination for triangularization of A. We have seen that the growth factor ρ determines the stability of the triangularization procedure. The next question is, How does the growth factor affect the solution procedure of $Ax = b$ using such a triangularization? The answer is given in the following.

Roundoff Error Result for Linear System Problem with Gaussian Elimination

It can be shown that the computed solution $\hat{x}$ of the linear system $Ax = b$, using Gaussian elimination, satisfies

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

where $\|E\|_\infty \le c(n^3 + 5n^2)\rho\, \|A\|_\infty \mu$, and c is a small constant. For a proof see Section 11.4.

Remark: From the preceding, we see that if the growth factor ρ is large, a large error can be expected.

6.4.4 Solution of $Ax = b$ without Explicit Factorization

As we have just seen, the Gaussian elimination method for solving $Ax = b$ comes in two stages. First, the matrix A is explicitly factorized:

$$A = LU \quad \text{(without pivoting)}$$
$$MA = U \quad \text{(with partial pivoting)}$$
$$MAQ = U \quad \text{(with complete pivoting)}$$

Second, the factorization of A is used to solve $Ax = b$. However, it is easy to see that these two stages can be combined so that *the solution can be obtained by solving an upper triangular system by processing the matrix A and the vector b simultaneously*. In this case, the augmented matrix (A, b) is triangularized and the solution is then obtained by back substitution. We illustrate this implicit process for Gaussian elimination with partial pivoting.

ALGORITHM 6.4.2 Solving $Ax = b$ with Partial Pivoting without Explicit Factorization

Given an $n \times n$ matrix nonsingular A and a vector b, the following algorithm computes the triangular factorization of the augmented matrix (A, b) using Gaussian elimination

with partial pivoting. A is overwritten by the transformed triangular matrix and b is overwritten by the transformed vector. The multipliers are stored in the lower half of A.

For $k = 1, 2, \ldots, n-1$ do

1: Choose the largest element in magnitude in the column k below row $(k-1)$; call it $a_{r_k k}$:

$$a_{r_k k} = \max\{|a_{ik}| : i \geq k\}$$

If $a_{r_k k} = 0$, Stop. Otherwise, go to Step 2.

2: Interchange the rows k and r_k of A and the kth and r_kth entries of b:

$$a_{kj} \leftrightarrow a_{r_k j} \qquad (j = k, k+1, \ldots, n)$$
$$b_k \leftrightarrow b_{r_k}$$

3: Form the multipliers:

$$a_{ik} \equiv m_{ik} = -\frac{a_{ik}}{a_{kk}} \qquad (i = k+1, \ldots, n)$$

4: Update the entries of A:

$$a_{ij} \equiv a_{ij} + m_{ik}a_{kj} \qquad (i = k+1, \ldots, n; \quad j = k+1, \ldots, n)$$

5: Update the entries of b:

$$b_i \equiv b_i + m_{ik}b_k \qquad (i = k+1, \ldots, n)$$

MATCOM Note: Algorithm 6.4.2 has been implemented in MATCOM program GAUSSWF.

Example 6.4.3

$$A = \begin{pmatrix} 0 & 1 & 1 \\ 2 & 2 & 3 \\ 4 & 1 & 1 \end{pmatrix}, \qquad b = \begin{pmatrix} 2 \\ 6 \\ 3 \end{pmatrix}$$

$k = 1$: The pivot entry is $a_{31} = 4$, $r_1 = 3$.
Interchange rows 3 and 1 of A and the third and first entries of b.

$$A \equiv \begin{pmatrix} 4 & 1 & 1 \\ 2 & 2 & 3 \\ 0 & 1 & 1 \end{pmatrix}, \qquad b \equiv \begin{pmatrix} 3 \\ 6 \\ 2 \end{pmatrix}$$

$$m_{21} = -\frac{a_{21}}{a_{11}} = -\frac{1}{2}$$

$$A \equiv A^{(1)} = \begin{pmatrix} 4 & 1 & 1 \\ 0 & \frac{3}{2} & \frac{5}{2} \\ 0 & 1 & 1 \end{pmatrix}, \qquad b \equiv b^{(1)} = \begin{pmatrix} 3 \\ \frac{9}{2} \\ 2 \end{pmatrix}$$

$k = 2$: The pivot entry is $a_{22} = \dfrac{3}{2}$.

$$m_{32} = -\frac{a_{32}}{a_{22}} = -\frac{2}{3}$$

$$A \equiv A^{(2)} = \begin{pmatrix} 4 & 1 & 1 \\ 0 & \frac{3}{2} & \frac{5}{2} \\ 0 & 0 & -\frac{2}{3} \end{pmatrix}, \qquad b \equiv b^{(2)} = \begin{pmatrix} 3 \\ \frac{9}{2} \\ -1 \end{pmatrix}$$

The reduced triangular system $A^{(2)}x = b^{(2)}$ is

$$\begin{aligned} 4x_1 + x_2 + x_3 &= 3 \\ \tfrac{3}{2}x_2 + \tfrac{5}{2}x_3 &= \tfrac{9}{2} \\ -\tfrac{2}{3}x_3 &= -1 \end{aligned}$$

The solution is

$$x_3 = \tfrac{3}{2}, \qquad x_2 = \tfrac{1}{2}, \qquad x_1 = \tfrac{1}{4}$$

6.4.5 Solution of $Ax = b$ Using QR Factorization

If we have

$$A = QR$$

the system $Ax = b$ then becomes

$$QRx = b$$

or

$$Rx = Q^T b = b'$$

Thus, once we have the QR factorization of A, the solution of the system $Ax = b$ can be obtained just by solving the equivalent upper triangular system:

$$Rx = b' = Q^T b$$

Solving $Ax = b$ Using QR Factorization
To solve $Ax = b$ using QR factorization:

1. Find the QR factorization of A : $Q^T A = R$ (Algorithms 5.4.3 or 5.5.4).
2. Form $b' = Q^T b$.
3. Solve $Rx = b'$.

Forming b'
To compute b', we do not need Q explicitly. It can be computed from the factored form of Q. For example, if the QR factorization is obtained using the Householder method (Section 5.4.1), then

$$Q^T = H_{n-1}H_{n-2}\cdots H_2H_1$$

and $b' = y_n = Q^T b$ can be computed as

Step 1: $y_1 = b$

Step 2: For $k = 1, 2, \ldots, n-1$ do

$y_{k+1} = H_k y_k$

Remark: Use Algorithm 5.4.2 to compute the vectors $y_{k+1} = H_k y_k$.

Example 6.4.4 Consider

$$A = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 1 & 1 \end{pmatrix}, \qquad b = \begin{pmatrix} 2 \\ 6 \\ 3 \end{pmatrix}$$

From Example 5.4.2, we know that the Householder method gives us

$$R = \begin{pmatrix} -1.4142 & -2.1213 & -2.8284 \\ 0 & 1.2247 & 1.6330 \\ 0 & 0 & -0.5774 \end{pmatrix}$$

$$H_1 = \begin{pmatrix} 0 & -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{\sqrt{2}} & -\frac{1}{2} & \frac{1}{2} \end{pmatrix} = \begin{pmatrix} 0 & -0.7071 & -0.7071 \\ -0.7071 & 0.5000 & -0.5000 \\ -0.7071 & -0.5000 & 0.5000 \end{pmatrix}$$

$$H_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -0.1691 & -0.9856 \\ 0 & -0.9856 & 0.1691 \end{pmatrix}$$

Compute b':

$$y_1 = b = \begin{pmatrix} 2 \\ 6 \\ 3 \end{pmatrix}$$

$$y_2 = H_1 y_1 = \begin{pmatrix} -6.3640 \\ 0.0858 \\ -2.9142 \end{pmatrix}$$

$$b' = y_3 = H_2 y_2 = \begin{pmatrix} -6.3640 \\ 2.8577 \\ -0.5774 \end{pmatrix}$$

(Note that b' above has been computed without explicitly forming the matrix Q.)

Solve:

$$Rx = b'$$

$$x = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

Flop count. If the *Householder method* is used to factor A into QR, the solution of $Ax = b$ requires $\frac{2}{3}n^3 + O(n^2)$ flops (Section 5.4.1); on the other hand, the *Givens rotations* technique requires about twice as many (Section 5.5.1).

Roundoff property. We know from Chapter 5 that both the Householder and the Givens methods for QR factorization of A are numerically stable. The back-substitution process for solving an upper triangular system is also numerically stable (Section 3.1.3). *Thus, the method for solving $Ax = b$ using QR factorization is likely to be stable.* Indeed, this is so.

Roundoff Error Result for Solving $Ax = b$ Using QR Factorization

It can be shown (Lawson and Hanson 1974, 92) that the computed solution $\hat{x}$, obtained by using the Householder QR method, is the exact solution of

$$(A + E)\hat{x} = b + \delta b$$

where $\|E\|_F \le (3n^2 + 41n)\mu\|A\|_F + O(\mu^2)$ and $\|\delta b\| \le (3n^2 + 40n)\mu\|b\| + O(\mu^2)$.

Remark: There is no "growth factor" in the preceding expression.

6.4.6 Solving a Linear System with Multiple Right-hand Sides

Consider the problem

$$AX = B$$

where $B = (b_1, \ldots, b_m)$ is an $n \times m$ matrix ($m \le n$) and $X = (x_1, x_2, \ldots, x_m)$. Here b_i and x_i, $i = 1, \ldots, m$ are n-vectors.

Problems of this type arise in many practical applications (one such application has been considered in Section 6.4.7, page 228).

Once the matrix A has been factored using any of the methods described in Chapter 5, the factorization can be used to solve the preceding m linear systems. We state the procedure only with partial pivoting.

ALGORITHM 6.4.3 Solving $AX = B$: Linear System with Multiple Right-hand Sides

Step 1: Factor $MA = U$, using Gaussian elimination with partial pivoting.

Step 2: Solve the m upper triangular systems:

$$Ux_i = b_i' = Mb_i, \quad i = 1, \ldots, m$$

Flop count. The algorithm will require about $n^3/3 + mn^2$ flops.

Example 6.4.5 Solve $AX = B$, where

$$A = \begin{pmatrix} 1 & 2 & 4 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}, \qquad B = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$$

$$U = \begin{pmatrix} 7 & 8 & 9 \\ 0 & \frac{6}{7} & \frac{19}{7} \\ 0 & 0 & \frac{-1}{2} \end{pmatrix}, \qquad M = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & -\frac{1}{7} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \end{pmatrix}$$

(See Example 5.2.4.)

Solve:

$$Ux_1 = Mb_1 = \begin{pmatrix} 5 \\ 0.2857 \\ 0 \end{pmatrix}$$

$$x_1 = \begin{pmatrix} 0.3333 \\ 0.3333 \\ 0 \end{pmatrix}$$

Solve:

$$Ux_2 = Mb_2 = \begin{pmatrix} 6 \\ 1.1429 \\ 0 \end{pmatrix}, \qquad x_2 = \begin{pmatrix} -0.6667 \\ 1.3333 \\ 0 \end{pmatrix}$$

6.4.7 Special Systems

In this subsection we study the following special systems:

1. Symmetric positive definite systems
2. Hessenberg systems
3. Diagonally dominant systems
4. Tridiagonal and block tridiagonal systems

We saw in Section 6.3 that these systems occur very often in practical applications, such as in numerical solution of partial differential equations. Indeed, it is very often said by practicing engineers that there are hardly any systems in practical applications that are not one of the above types. Hence, these systems deserve special treatment.

Symmetric Positive Definite Systems

First, we show that for a symmetric positive definite matrix A, there exists a unique factorization

$$A = HH^T$$

where H is a lower triangular matrix with positive diagonal entries. The factorization is called the **Cholesky factorization**, after the French engineer Cholesky (1875–1918). The existence of the Cholesky factorization for a symmetric positive definite matrix A can be seen either via LU factorization of A or by finding the matrix H directly from the preceding relation. In practical computations, we will use the latter.

To see this via LU factorization, we note that because A is positive definite and, therefore, has positive leading principal minors, by Theorem 5.2.1 the factorization

$$A = LU$$

is unique. The upper triangular matrix U can be written as

$$U = DU_1$$

where

$$\begin{aligned} D &= \operatorname{diag}(u_{11}, u_{22}, \ldots, u_{nn}) \\ &= \operatorname{diag}(a_{11}, a_{22}^{(1)}, \ldots, a_{nn}^{(n-1)}) \end{aligned}$$

and U_1 is a *unit* upper triangular matrix. Thus,

$$A = LDU_1$$

Because $A = A^T$, we have

$$U_1^T DL^T = LDU_1$$

or

$$D = (U_1^T)^{-1} LDU_1 (L^T)^{-1}$$

The matrix $(U_1^T)^{-1}L$ is a unit lower triangular matrix, and the matrix $U_1(L^T)^{-1}$ is unit upper triangular. It therefore follows that

$$(U_1^T)^{-1}L = U_1(L^T)^{-1} = I$$

that is,

$$U_1 = L^T$$

So A can be written as

$$A = LDL^T$$

where L is unit lower triangular. Because the leading principal minors of A are $a_{11}, a_{11}a_{22}^{(1)}, \ldots, a_{11}a_{22}^{(1)} \ldots a_{nn}^{(n-1)}$, A is positive definite if and only if the pivots $a_{11}, a_{22}^{(1)}, \ldots, a_{nn}^{(n-1)}$ are positive. This means that when A is positive definite, the diagonal entries of D are positive; therefore, we can write

$$D = D^{1/2} \cdot D^{1/2}$$

where

$$D^{1/2} = \operatorname{diag}\left(\sqrt{a_{11}}, \sqrt{a_{22}^{(1)}}, \ldots, \sqrt{a_{nn}^{(n-1)}}\right) = \operatorname{diag}\left(\sqrt{u_{11}}, \sqrt{u_{22}}, \ldots, \sqrt{u_{nn}}\right)$$

So,

$$A = LDL^T = LD^{1/2}D^{1/2}L^T = H \cdot H^T$$

Note that the diagonal entries of $H = LD^{1/2}$ are positive.

This discussion can be summarized in the following theorem.

THEOREM 6.4.1 The Cholesky Factorization Theorem Let A be a symmetric positive definite matrix. Then A can be written uniquely in the form

$$A = HH^T$$

where H is a lower triangular matrix with positive diagonal entries. An explicit expression for H is given by

$$H = LD^{1/2}$$

where L is the unit lower triangular matrix in the LU factorization of A obtained by Gaussian elimination without pivoting and

$$D^{1/2} = \operatorname{diag}\left(u_{11}^{1/2}, \ldots, u_{nn}^{1/2}\right)$$

Gaussian Elimination for a Symmetric Positive Definite Matrix

We now show that Gaussian elimination without pivoting is stable for symmetric positive definite matrices by exhibiting some remarkable invariant properties of symmetric positive definite matrices. The following example illustrates that *even when there is a small pivot, Gaussian elimination without pivoting does not give rise to the growth in the entries of the matrices* $A^{(k)}$. Let

$$A = \begin{pmatrix} 0.00003 & 0.00500 \\ 0.00500 & 1.0000 \end{pmatrix}$$

There is only one step. The pivot entry is 0.00003. It is *small*. The multiplier m_{21} is *large*:

$$m_{21} = -\frac{a_{21}}{a_{11}} = -\frac{0.00500}{0.00003} = -\frac{500}{3}$$

But

$$A^{(1)} = \begin{pmatrix} 0.00003 & 0.00500 \\ 0 & 0.166667 \end{pmatrix}$$

The entries of $A^{(1)}$ did not grow. In fact, $\max(a_{ii}^{(1)}) = 0.166667 < \max(a_{ii}) = 1$. This interesting phenomenon of Gaussian elimination without pivoting applied to the preceding 2×2 simple example (positive definite) can be explained through the following result.

THEOREM 6.4.2 Let $A = (a_{ij})$ be an $n \times n$ symmetric positive definite matrix and let $A^{(k)} = (a_{ij}^{(k)})$ be the reduced matrices obtained by applying Gaussian elimination without pivoting to A. Then

1. For each $1 \le k \le n-1$, the submatrix obtained by deleting the first k rows and k columns of $A^{(k)}$ is symmetric positive definite.
2. $\max a_{ii}^{(k)} \le \max a_{ii}^{(k-1)}$, $k = 1, 2, \ldots, n-1$.

Proof: We prove the results just for the first step of elimination, because the results for the other steps then follow inductively.

After the first step of elimination, we have

$$A^{(1)} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ 0 & a_{22}^{(1)} & \cdots & a_{2n}^{(1)} \\ \vdots & \vdots & & \vdots \\ 0 & a_{n2}^{(1)} & \cdots & a_{nn}^{(1)} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ 0 & & & \\ \vdots & & B & \\ 0 & & & \end{pmatrix}$$

To prove that B is positive definite, consider the quadratic form:

$$\sum_{i=2}^{n}\sum_{j=2}^{n} a_{ij}^{(1)} x_i x_j = \sum_{i=2}^{n}\sum_{j=2}^{n}\left(a_{ij} - \frac{a_{i1}a_{j1}}{a_{11}}\right) x_i x_j$$

$$= \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij} x_i x_j - a_{11}\left(x_1 + \sum_{i=2}^{n}\frac{a_{i1}}{a_{11}} x_i\right)^2$$

If B is not positive definite, then there will exist $x_2, \ldots, x_n$ such that

$$\sum_{i=2}^{n}\sum_{j=2}^{n} a_{ij}^{(1)} x_i x_j \le 0$$

With these values of x_2 through x_n, if we define

$$x_1 = -\sum_{i=2}^{n}\frac{a_{i1}}{a_{11}} x_i$$

then from the preceding expression we have the quadratic form

$$\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij} x_i x_j \le 0$$

which contradicts the fact that A is positive definite. Thus B is positive definite.

Also, we note

$$a_{ii}^{(1)} \le a_{ii}, \qquad i = 1, 2, \ldots, n$$

for

$$0 \le a_{ii}^{(1)} = a_{ii} - \frac{a_{i1}^2}{a_{11}} \le a_{ii} \qquad (\text{because } a_{11} > 0)$$

Thus, each diagonal entry of $A^{(1)}$ is less than or equal to the corresponding diagonal entry of A. ■

A Consequence of Theorem 6.4.2

From Theorem 6.4.2, we immediately conclude that if $|a_{ij}| \le 1$, then $|a_{ij}^{(k)}| \le 1$. This means that the growth factor ρ in this case is 1.

The Growth Factor and Stability of Gaussian Elimination for a Positive Definite Matrix

The growth factor ρ of Gaussian elimination without pivoting for a symmetric positive definite matrix is 1. Thus, Gaussian elimination without pivoting for a symmetric positive definite matrix is stable.

Example 6.4.6

$$A = \begin{pmatrix} 5 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 5 \end{pmatrix}$$

$$A^{(1)} = \begin{pmatrix} 5 & 1 & 1 \\ 0 & \frac{4}{5} & \frac{4}{5} \\ 0 & \frac{4}{5} & \frac{24}{5} \end{pmatrix}$$

The leading principal minors of the submatrix obtained by deleting the first row and first column are $\frac{4}{5}$ and $\frac{16}{5}$. Also,

$$\max(a_{ij}^{(1)}) = 5$$

$$A^{(2)} = \begin{pmatrix} 5 & 1 & 1 \\ 0 & \frac{4}{5} & \frac{4}{5} \\ 0 & 0 & 4 \end{pmatrix}$$

The leading principal minor of the submatrix obtained by deleting the first two rows and columns is 4. Furthermore, $\max(a_{ij}^{(2)}) = 5 = \max(a_{ij}^{(1)})$. The growth factor is $\rho = \frac{5}{5} = 1$.

Computing the Cholesky Factorization

Theorem 6.4.1 suggests that Gaussian elimination can be used to compute the Cholesky factorization of a symmetric positive definite matrix, and we have just seen that Gaussian elimination without pivoting is stable for such a matrix. However, in practice we do not use Gaussian elimination to compute the Cholesky factorization. Instead, we use a more efficient method, called the *Cholesky algorithm*, to do so.

The Cholesky Algorithm

From

$$A = HH^T$$

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} = \begin{pmatrix} h_{11} & 0 & \cdots & 0 \\ h_{21} & h_{22} & 0\cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ h_{n1} & h_{n2} & \cdots & h_{nn} \end{pmatrix} \begin{pmatrix} h_{11} & h_{21} & \cdots & h_{n1} \\ 0 & h_{22} & \cdots & h_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & h_{nn} \end{pmatrix}$$

we have

$$h_{11} = \sqrt{a_{11}}, \qquad h_{i1} = \frac{a_{i1}}{h_{11}}, \qquad i = 2, \ldots, n$$

$$\sum_{k=1}^{i} h_{ik}^2 = a_{ii}, \qquad a_{ij} = \sum_{k=1}^{j} h_{ik}h_{jk}, \qquad j < i$$

which leads to the **Cholesky algorithm.***

ALGORITHM 6.4.4 Cholesky Algorithm Given an $n \times n$ symmetric positive definite matrix A, the following algorithm computes the Cholesky factor H. The matrix H is computed row by row and is stored in the lower triangular part of A.

For $k = 1, 2, \ldots, n$ do

  For $i = 1, 2, \ldots, k-1$ do

    $a_{ki} \equiv h_{ki} = \frac{1}{h_{ii}}\left(a_{ki} - \sum_{j=1}^{i-1} h_{ij}h_{kj}\right)$

  $a_{kk} \equiv h_{kk} = \sqrt{a_{kk} - \sum_{j=1}^{k-1} h_{kj}^2}$

Remarks:

1. In the preceding pseudocode, $\sum_{j=1}^{0}(\,) \equiv 0$. Also when $k = 1$, the inner loop is skipped.
2. The Cholesky factor H is computed row by row.
3. The positive definiteness of A will make the quantities under the square root sign positive.

Roundoff property. Let the computed Cholesky factor be denoted by $\hat{H}$. Then, it can be shown (Demmel 1989) that

$$A + E = \hat{H}(\hat{H})^T$$

where $|e_{ij}| \le \dfrac{(n+1)\mu}{1-(n+1)\mu}(a_{ii}a_{jj})^{1/2}$ and

$$E = (e_{ij})$$

Thus, *the Cholesky factorization algorithm is stable.*

MATCOM Note: The Cholesky algorithm (Algorithm 6.4.4) has been implemented in MATCOM program CHOLES.

*This algorithm in some applications (such as in statistics) is known as the *square root algorithm*. A square root–free algorithm, however, can be developed.

Solution of $Ax = b$ Using the Cholesky Factorization

Having the Cholesky factorization $A = HH^T$ at hand, the positive definite linear system $Ax = b$ can now be solved by solving the lower triangular system $Hy = b$ first, followed by the upper triangular system $H^T x = y$.

ALGORITHM 6.4.5 The Cholesky Algorithm for the Positive Definite System $Ax = b$

Step 1: Find the Cholesky factorization of

$$A = HH^T$$

using Algorithm 6.4.4.

Step 2: Solve the lower triangular system for y:

$$Hy = b$$

Step 3: Solve the upper triangular system for x:

$$H^T x = y$$

Example 6.4.7 Let

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 5 & 5 \\ 1 & 5 & 14 \end{pmatrix}, \qquad b = \begin{pmatrix} 3 \\ 11 \\ 20 \end{pmatrix}$$

Step 1: Find the Cholesky factorization.

1st row ($k = 1$):

$$h_{11} = 1$$

2nd row ($k = 2$):

$$h_{21} = \frac{a_{21}}{h_{11}} = 1$$

$$h_{22} = \sqrt{a_{22} - h_{21}^2} = \sqrt{5 - 1} = \pm 2$$

(Because the diagonal entries of H have to be positive, we take the + sign.)

3rd row ($k = 3$):

$$h_{31} = \frac{a_{31}}{h_{11}} = 1$$

$$h_{32} = \frac{1}{h_{22}}(a_{32} - h_{21}h_{31}) = \frac{1}{2}(5 - 1) = 2$$

$$h_{33} = \sqrt{a_{33} - (h_{31}^2 + h_{32}^2)} = \sqrt{14 - 5} = \sqrt{9}$$

(We take the + sign.)

$$h_{33} = +3$$

$$H = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 2 & 0 \\ 1 & 2 & 3 \end{pmatrix}$$

Step 2: Solve $Hy = b$:

$$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 2 & 0 \\ 1 & 2 & 3 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 3 \\ 11 \\ 20 \end{pmatrix}$$

$$y_1 = 3, \qquad y_2 = 4, \qquad y_3 = 3$$

Step 3: Solve $H^T x = y$:

$$\begin{pmatrix} 1 & 1 & 1 \\ 0 & 2 & 2 \\ 0 & 0 & 3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 3 \\ 4 \\ 3 \end{pmatrix}$$

$$x_3 = 1, \qquad x_2 = 1, \qquad x_3 = 1$$

Flop count. The Cholesky algorithm requires $n^3/6$ flops to compute H, one-half of the number of flops required to do the same job using LU factorization. Note that the process also requires n square roots. The solution of each triangular system $Hy = b$ and $H^T x = y$ requires $n^2/2$ flops. *Thus, the solution of the positive definite system $Ax = b$ using the Cholesky algorithm requires $n^3/6 + n^2$ flops and n square roots.*

Roundoff property. If $\hat{x}$ is the computed solution of the system $Ax = b$ using the Cholesky algorithm, then it can be shown that $\hat{x}$ satisfies

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

where $\|E\|_2 \le c\mu\|A\|_2$ and c is a small constant depending upon n. *Thus, the Cholesky algorithm for solving a symmetric positive definite system is quite stable.*

Relative Error in the Solution by the Cholesky Algorithm

If $\hat{x}$ is the computed solution to the symmetric positive definite system $Ax = b$ using the Cholesky algorithm followed by triangular system solutions (as just described) then it can be shown that

$$\frac{\|x - \hat{x}\|_2}{\|\hat{x}\|_2} \le O(\mu)\ \mathrm{Cond}(A)$$

(Recall that $\mathrm{Cond}(A) = \|A\| \cdot \|A^{-1}\|$.)

Symmetric Indefinite Systems

Gaussian elimination with partial pivoting does not preserve symmetry. Thus, if Gaussian elimination with partial pivoting is applied to a symmetric system, about $n^3/3$ flops and $O(n^2)$ comparisons will be needed, resulting in no savings for symmetry. There are efficient symmetry preserving algorithms such as *diagonal pivoting factorization algorithms*. The diagonal pivoting factorization is a factorization of $A = UDU^T$, where U is a product of elementary unit upper triangular and permutation matrices, and D is a symmetric block diagonal matrix with diagonal blocks of order 1 or 2. The diagonal pivoting factorization requires about $n^3/6$ flops, one-half of that of Gaussian elimination. The factorization is also stable in practice. This fac-

torization is popularly known as *Bunch–Parlett* or *Bunch–Kaufman* factorizations. The details can be found in Bunch (1971), Bunch and Parlett (1971), and Bunch and Kaufman (1977). Given such a factorization, a symmetric system can be solved in $O(n^2)$ flops.

Hessenberg System

Consider the linear system

$$Ax = b$$

where A is an upper Hessenberg matrix of order n. If Gaussian elimination with partial pivoting is used to triangularize A and if $|a_{ij}| \leq 1$, then $|a_{ij}^{(k)}| \leq k + 1$ (Wilkinson 1965, 218). Thus, we can state the following.

> **Growth Factor and Stability of Gaussian Elimination for a Hessenberg System**
> The growth factor for a Hessenberg matrix using Gaussian elimination with partial pivoting is bounded by n. Thus, *a Hessenberg system can be safely solved using partial pivoting.*

Flop count. It requires only n^2 flops to solve a Hessenberg system, significantly fewer than the $n^3/3$ flops required to solve a system with an arbitrary matrix. This situation occurs because, at each step of elimination during triangularization process, only one element needs to be eliminated; since there are $n - 1$ steps, the triangularization process requires about $n^2/2$ flops. Once we have the factorization

$$MA = U$$

the upper triangular system

$$Ux = Mb = b'$$

can be solved in $n^2/2$ flops. Thus, *a Hessenberg system can be solved with only n^2 flops in a stable way using Gaussian elimination with partial pivoting.*

Example 6.4.8 Triangularize

$$A = \begin{pmatrix} 1 & 2 & 3 \\ \boxed{2} & 3 & 4 \\ 0 & 5 & 6 \end{pmatrix}$$

using partial pivoting.

Step 1:

$$P_1 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$P_1 A = \begin{pmatrix} 2 & 3 & 4 \\ 1 & 2 & 3 \\ 0 & 5 & 6 \end{pmatrix}$$

$$\widehat{M}_1 = \begin{pmatrix} 1 & 0 \\ -\frac{1}{2} & 1 \end{pmatrix}$$

$$M_1 = \begin{pmatrix} 1 & 0 & 0 \\ -\frac{1}{2} & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$M_1 P_1 A = A^{(1)} = \begin{pmatrix} 1 & 0 & 0 \\ -\frac{1}{2} & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 3 & 4 \\ 1 & 2 & 3 \\ 0 & 5 & 6 \end{pmatrix}$$

$$= \begin{pmatrix} 2 & 3 & 4 \\ 0 & \frac{1}{2} & 1 \\ 0 & \boxed{5} & 6 \end{pmatrix}$$

Step 2:

$$P_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

$$P_2 A^{(1)} = \begin{pmatrix} 2 & 3 & 4 \\ 0 & 5 & 6 \\ 0 & \frac{1}{2} & 1 \end{pmatrix}$$

$$\widehat{M}_2 = \begin{pmatrix} 1 & 0 \\ -\frac{1}{10} & 1 \end{pmatrix}$$

$$M_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -\frac{1}{10} & 1 \end{pmatrix}$$

$$U = A^{(2)} = M_2 P_2 A^{(1)} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -\frac{1}{10} & 1 \end{pmatrix} \begin{pmatrix} 2 & 3 & 4 \\ 0 & 5 & 6 \\ 0 & \frac{1}{2} & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 2 & 3 & 4 \\ 0 & 5 & 6 \\ 0 & 0 & \frac{2}{5} \end{pmatrix}$$

$$M = M_2 P_2 M_1 P_1 = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & -\frac{1}{2} & -\frac{1}{10} \end{pmatrix}$$

Compute the growth factor, ρ:

$$\rho = \frac{\max(6, 6, 6)}{6} = 1$$

An Application of a Hessenberg System: Computing the Frequency Response Matrix

In control theory it is often necessary to compute the matrix

$$G(j\omega) = C(j\omega I - A)^{-1}B$$

for many different values of ω in order to study the response of a control system. The matrices A, B, and C are matrices of a control system and are of order $n \times n$, $n \times m$, and $r \times n$, respectively ($m \leq n$). The matrix $G(j\omega)$ is called the **frequency response matrix** ($j = \sqrt{-1}$).

Because computing $(j\omega I - A)^{-1}B$ is equivalent to solving m linear systems (see Section 6.5.1):

$$(j\omega I - A)x_i = b_i, \qquad i = 1, \ldots, m$$

where b_i is the ith column of B; for *each* ω, we need about $n^3/3 + mn^2 + rnm$ flops to compute $G(j\omega)$ (using Gaussian elimination with partial pivoting). Typically, $G(j\omega)$ needs to be computed for a very large number of ω, and so, such computation will be formidable.

On the other hand, if A is transformed initially to a Hessenberg matrix by orthogonal similarity,

$$A = PHP^T$$

then

$$G(j\omega) = CP(j\omega I - H)^{-1}P^TB$$

The computation of $(j\omega I - H)^{-1}P^TB$ will now require solutions of m *Hessenberg systems*, each of which will require only n^2 flops. Then, for computing $G(j\omega) = CP(j\omega I - H)^{-1}P^TB$ for each ω, there will be a saving of order $O(n)$ for each ω. This count does not include reduction to Hessenberg form. Note that the matrix A is transformed to a Hessenberg matrix once and the same Hessenberg matrix is used to compute $G(j\omega)$ for each ω.

Thus, *the computation that uses an initial reduction of A to a Hessenberg matrix is much more efficient.* Moreover, as we have seen before, reduction to Hessenberg form and solutions of Hessenberg systems using partial pivoting are both *stable* computations. This approach was suggested by Laub (1981).

Remarks: Reduction of A to a Hessenberg matrix serves as a beginning for major computations in control theory. Numerically viable algorithms for important control problems such as controllability and observability, matrix equations (Lyapunov, Sylvester, Sylvester–Observer, Riccati, etc.) routinely transform the matrix A to a Hessenberg matrix before actual computations start. Readers familiar with and interested in these problems might want to look into the book *Numerical Methods in Control Theory* by B. N. Datta (forthcoming) for an account of these methods. For references on the individual papers, see Chapter 8. The recent reprint book *Numerical Linear Algebra Techniques for Systems and Control*, edited by R. V. Patel, Alan Laub, and Paul Van Dooren (1994), also contains all the relevant papers. See also the recent Ph.D. thesis of Arnold (1992) and the survey paper of Datta (1994).

Diagonally Dominant Systems

Recall that a matrix $A = (a_{ij})$ is **column diagonally dominant** if

$$|a_{11}| > |a_{21}| + |a_{31}| + \cdots + |a_{n1}|$$
$$|a_{22}| > |a_{12}| + |a_{32}| + \cdots + |a_{n2}|$$
$$\vdots$$
$$|a_{nn}| > |a_{1n}| + |a_{2n}| + \cdots + |a_{n-1,n}|$$

A column diagonally dominant matrix, like a symmetric positive definite matrix, possesses the attractive property that no row interchanges are necessary at any step during the triangularization procedure for Gaussian elimination with partial pivoting. The pivot element is already there in the right place. Thus, at the first step, a_{11} being the largest in magnitude of all the elements in the first column, no row interchange is necessary. At the end of the first step, we then have

$$A^{(1)} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ 0 & a_{22}^{(1)} & \cdots & a_{2n}^{(1)} \\ \vdots & \vdots & & \vdots \\ 0 & a_{n2}^{(1)} & \cdots & a_{nn}^{(1)} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ 0 & & & \\ \vdots & & & A' \\ 0 & & & \end{pmatrix}$$

and it can be shown (Exercise 16a) that A' is again column diagonally dominant and, therefore, $a_{22}^{(1)}$ is the pivot for the second step. This process can obviously be continued, showing that pivoting is not needed for column diagonally dominant matrices. Furthermore, the following can be proved (see Higham (1996), 253).

Growth Factor and Stability of Gaussian Elimination for Diagonally Dominant Systems

The **growth factor** ρ for a column diagonally dominant matrix using Gaussian elimination without pivoting is bounded by 2 (Exercise 16): $\rho \leq 2$. Thus, for *column diagonally dominant systems, Gaussian elimination without pivoting is stable.*

Example 6.4.9

$$A = \begin{pmatrix} \boxed{5} & 1 & 1 \\ 1 & 5 & 1 \\ 1 & 1 & 5 \end{pmatrix}$$

Step 1:

$$A^{(1)} = \begin{pmatrix} 5 & 1 & 1 \\ 0 & \boxed{\frac{24}{5}} & \frac{4}{5} \\ 0 & \frac{4}{5} & \frac{24}{5} \end{pmatrix}$$

Step 2:

$$A^{(2)} = \begin{pmatrix} 5 & 1 & 1 \\ 0 & \frac{24}{5} & \frac{4}{5} \\ 0 & 0 & \frac{14}{3} \end{pmatrix}$$

The growth factor is $\rho = 1$. (Note that, for this example, the matrix A is column diagonally dominant and positive definite; thus, $\rho = 1$.)

The next example shows that the growth factor of Gaussian elimination for a column diagonally dominant matrix can be greater than 1 but is always less than or equal to 2.

Example 6.4.10

$$A = \begin{pmatrix} 5 & -8 \\ 1 & 10 \end{pmatrix}$$

$$A^{(1)} = \begin{pmatrix} 5 & -8 \\ 0 & 11.6 \end{pmatrix}$$

The growth factor is

$$\rho = \frac{\max(10, 11.6)}{10} = \frac{11.6}{10} = 1.16$$

Tridiagonal Systems

The LU factorization of a tridiagonal matrix T, when it exists, may yield L and U having very special simple structures: both bidiagonal, L having 1s along the main diagonal and the superdiagonal entries of U the same as those of T. Specifically, if we write

$$T = \begin{pmatrix} a_1 & b_1 & & \\ c_2 & \ddots & \ddots & \\ & \ddots & \ddots & b_{n-1} \\ & & c_n & a_n \end{pmatrix}$$

$$= \begin{pmatrix} 1 & & & \\ \ell_2 & \ddots & 0 & \\ & \ddots & \ddots & \\ & 0 & \ddots & \ddots \\ & & & \ell_n \quad 1 \end{pmatrix} \begin{pmatrix} u_1 & b_1 & & \\ & \ddots & \ddots & 0 \\ & & \ddots & \ddots \\ & 0 & & \ddots \quad b_{n-1} \\ & & & u_n \end{pmatrix}$$

By equating the corresponding elements of the matrices on both sides, we see that

$$a_1 = u_1$$
$$c_i = \ell_i u_{i-1}, \qquad i = 2, \ldots, n$$
$$a_i = u_i + \ell_i b_{i-1}, \qquad i = 2, \ldots, n,$$

from which $\{\ell_i\}$ and $\{u_i\}$ can be easily computed.

Computing the LU Factorization of a Tridiagonal Matrix

$u_1 = a_1$

For $i = 2, \ldots, n$ do

$\ell_i = \dfrac{c_i}{u_{i-1}}$

$u_i = a_i - \ell_i b_{i-1}$

Flop count. The preceding procedure only takes $2n - 2$ flops.

Solving a Tridiagonal System

Once we have the preceding simple factorization of T, the solution of the tridiagonal system $Tx = b$ can be found by solving the two special *bidiagonal* systems:

$$Ly = b$$

and

$$Ux = y$$

Flop count. The solutions of these two bidiagonal systems also require $2n - 2$ flops. Thus, *a tridiagonal system can be solved by this procedure in only* $4n - 4$ *flops*, a very cheap procedure indeed.

Stability of the process. Unfortunately, this factorization procedure breaks down if any u_i is zero. Even if all u_i are theoretically nonzero, the *stability of the process in general cannot be guaranteed.* However, in many practical situations, such as in discretizing Poisson's equation, the tridiagonal matrices are *symmetric positive definite*, in which cases, as we have seen before, the above procedure is quite stable.

In fact, in the symmetric positive definite case, this procedure should be preferred over the Cholesky-factorization technique, because it does not involve computations of any square roots. It is true that the Cholesky factorization of a symmetric positive definite tridiagonal matrix can also be computed in $O(n)$ flops; however, an additional n square roots have to be computed.

In the general case, to maintain the stability, Gaussian elimination with partial pivoting should be used.

If $|a_i|, |b_i|, |c_i| \le 1$ $(i - 1, \ldots, n)$, then it can be shown (Wilkinson 1965, 219) that the entries of $A^{(k)}$ at each step will be bounded by 2.

Growth Factor and Stability of Gaussian Elimination for a Tridiagonal System

The growth factor for Gaussian elimination with partial pivoting for a tridiagonal matrix is bounded by 2:

$$\rho \le 2$$

Thus, Gaussian elimination with partial pivoting for a tridiagonal system is very stable.

The flop count in this case is little higher; it takes about $7n$ flops to solve the system $Tx = b$ ($3n$ flops for decomposition and $4n$ for solving two triangular systems), but it is still an $O(n)$ procedure.

If T is symmetric, we naturally want to take advantage of the symmetry; however, the diagonal-pivoting factorization algorithms can be used. These algorithms can be arranged to have a flop count comparable to that of Gaussian elimination with partial pivoting and require less storage than the latter.

Example 6.4.11 Triangularize

$$A = \begin{pmatrix} 0.9 & 0.1 & 0 \\ 0.8 & 0.5 & 0.1 \\ 0 & 0.1 & 0.5 \end{pmatrix}$$

using (a) the formula $A = LU$ and (b) Gaussian elimination.

(a) From $A = LU$,

$$u_1 = 0.9$$

$i = 2$:

$$\ell_2 = \frac{c_2}{u_1} = \frac{0.8}{0.9} = \frac{8}{9} = 0.8889$$

$$u_2 = a_2 - \ell_2 b_1 = 0.5 - \frac{8}{9} \times 0.1 = 0.4111$$

$i = 3$:

$$\ell_3 = \frac{c_3}{u_2} = \frac{0.1}{0.41} = 0.2432$$

$$u_3 = a_3 - \ell_3 b_2 = 0.5 - 0.24 \times 0.1 = 0.4757$$

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 0.8889 & 1 & 0 \\ 0 & 0.2432 & 0.1 \end{pmatrix}$$

$$U = \begin{pmatrix} 0.9 & 0.1 & 0 \\ 0 & 0.4111 & 0.1 \\ 0 & 0 & 0.4757 \end{pmatrix}$$

(b) Using Gaussian elimination with partial pivoting, we have the following.

Step 1: Multiplier $m_{21} = -\dfrac{0.8}{0.9} = -0.8889$:

$$A^{(1)} = \begin{pmatrix} 0.9 & 0.1 & 0 \\ 0 & 0.4111 & 0.1 \\ 0 & 0.1 & 0.5 \end{pmatrix}$$

Step 2: Multiplier $m_{32} = -\dfrac{0.1}{0.4111} = -0.2432$:

$$A^{(2)} = \begin{pmatrix} 0.9 & 0.1 & 0 \\ 0 & 0.4111 & 0.1 \\ 0 & 0 & 0.4757 \end{pmatrix} = U$$

$$L = \begin{pmatrix} 1 & 0 & 0 \\ -m_{21} & 1 & 0 \\ 0 & -m_{32} & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & 0 \\ 0.8889 & 1 & 0 \\ 0 & 0.2432 & 1 \end{pmatrix}$$

Block Tridiagonal Systems

In this section we consider solving the block tridiagonal system

$$Tx = b$$

where T is a block tridiagonal matrix and $b = (b_1, b_2, \ldots, b_n)^T$ is a block vector. The number of components of the block vector b_i is the same as the dimension of the ith diagonal block matrix in T.

Block LU Factorization

The factorization procedure given in the beginning of this section may be easily extended to the case of the block tridiagonal matrix

$$T = \begin{pmatrix} A_1 & B_1 & & & \\ C_2 & \ddots & \ddots & 0 & \\ & \ddots & \ddots & \ddots & \\ & 0 & \ddots & \ddots & B_{N-1} \\ & & & C_N & A_N \end{pmatrix}$$

Thus, if T has the block LU factorization

$$T = \begin{pmatrix} I & & & & \\ L_2 & \ddots & & 0 & \\ & \ddots & \ddots & & \\ & 0 & \ddots & \ddots & \\ & & & L_N & I \end{pmatrix} \begin{pmatrix} U_1 & B_1 & \cdots & & 0 \\ & \ddots & \ddots & & \vdots \\ & & \ddots & \ddots & \vdots \\ \vdots & & & \ddots & B_{N-1} \\ 0 & \cdots & & \cdots & U_N \end{pmatrix} = LU$$

then the matrices L_i, $i = 2, \ldots, N$, and U_i, $i = 1, \ldots, N$, can be computed as follows.

ALGORITHM 6.4.6 Block LU Factorization

Set $U_1 = A_1$

For $i = 2, \ldots, N$ do

Step 1: Solve for L_i:

$$L_i U_{i-1} = C_i$$

Step 2: Compute U_i:

$$U_i = A_i - L_i B_{i-1}$$

Solution of Block Systems

Once we have the preceding factorization, we can find the solution x of the system

$$Tx = b$$

by solving $Ly = b$ and $Ux = y$ successively. The solution of $Ly = b$ can be achieved by **block forward elimination**, and that of $Ux = y$ can be achieved by **block back substitution**.

ALGORITHM 6.4.7 Block Forward Elimination

Set $L_1 y_0 = 0$
For $i = 1, \ldots, n$ do
$y_i = b_i - L_i y_{i-1}$

ALGORITHM 6.4.8 Block Back Substitution

Set $B_N x_{N+1} = 0$.
For $i = N, \ldots, 1$ do
Solve for x_i: $U_i x_i = y_i - B_i x_{i+1}$

Example 6.4.12

$$A = \begin{pmatrix} 4 & -1 & 1 & 0 \\ -1 & 4 & 0 & 1 \\ 1 & 0 & 2 & -1 \\ 0 & 1 & -1 & 2 \end{pmatrix}, \qquad b = \begin{pmatrix} 4 \\ 4 \\ 2 \\ 2 \end{pmatrix}$$

$$A_1 = \begin{pmatrix} 4 & -1 \\ -1 & 4 \end{pmatrix}, \qquad A_2 = \begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix}, \qquad B_1 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$b_1 = \begin{pmatrix} 4 \\ 4 \end{pmatrix}, \qquad b_2 = \begin{pmatrix} 2 \\ 2 \end{pmatrix}$$

Block LU factorization: Set

$$U_1 = A_1 = \begin{pmatrix} 4 & -1 \\ -1 & 4 \end{pmatrix}$$

$i = 2$:

1. Solve for L_2:

$$L_2 U_1 = I_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$L_2 = U_1^{-1} = \begin{pmatrix} 0.2667 & 0.0667 \\ 0.667 & 0.2667 \end{pmatrix}$$

2. Compute U_2 from

$$\begin{aligned} U_2 &= A_2 - L_2 B_1 \\ &= \begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix} - \begin{pmatrix} 0.2667 & 0.0667 \\ 0.0667 & 0.2667 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \\ &= \begin{pmatrix} 1.7333 & -1.0667 \\ -1.0667 & 1.7333 \end{pmatrix} \end{aligned}$$

Block forward elimination:

$$y_1 = b_1 - L_1 y_0 = b_1 = \begin{pmatrix} 4 \\ 4 \end{pmatrix}$$

$$y_2 = b_2 - L_2 y_1 = \begin{pmatrix} 0.6667 \\ 0.6667 \end{pmatrix}$$

Block back substitution:

$$U_2 x_2 = y_2 - B_2 x_3 = y_2 = \begin{pmatrix} 0.6667 \\ 0.6667 \end{pmatrix} \qquad (B_2 x_3 = 0)$$

$$x_2 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

$$U_1 x_1 = y_1 - B_1 x_2 = \begin{pmatrix} 4 \\ 4 \end{pmatrix} - \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 3 \\ 3 \end{pmatrix}$$

$$x_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Block Cyclic Reduction

Frequently in practice, the block tridiagonal matrix of a system may possess some special properties that can be exploited to reduce the system to a single lower-order system by using a technique called **block cyclic reduction**. For details see the book by Golub and Van Loan (1989, 173–6) and the references therein.

6.4.8 Scaling

If the entries of the matrix A vary widely, then there is a possibility that a very small number needs to be added to a very large number during the process of elimination. Such a situation can influence the accuracy greatly, because the larger value can obliterate the smaller one. To circumvent this difficulty, it is often suggested that the rows of A be properly scaled before the elimination process begins. The following simple example illustrates this.

Consider the system

$$\begin{pmatrix} 10 & 10^6 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 10^6 \\ 2 \end{pmatrix}$$

Apply Gaussian elimination with partial pivoting. Because 10 is the largest entry in the first column, no interchange is needed. We have, after the first step of elimination using 4 digit arithmetic,

$$\begin{pmatrix} 10 & 10^6 \\ 0 & -10^5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 10^6 \\ -10^5 \end{pmatrix}$$

which gives $x_2 = 1 \quad x_1 = 0$. The exact solution, however, is approximately

$$\begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Therefore, even choosing the *false pivot* 10 did not help us. However, if we scale the entries of the first row of the matrix by dividing it by 10^6 and then solve the system (after modifying the first entry of b appropriately) using partial pivoting, we will then have the accurate solution, as we have seen before.

Scaling of the rows of a matrix A is equivalent to finding an invertible diagonal matrix D_1 so that the largest element (in magnitude) in each row of $D_1^{-1}A$ is about the same size. Once such D_1 is found, the solution of the system

$$Ax = b$$

is found by solving the scaled system

$$\tilde{A}x = \tilde{b}$$

where

$$\tilde{A} = D_1^{-1}A, \qquad \tilde{b} = D_1^{-1}b$$

The process can be easily extended to scale both the rows and columns of A. Mathematically, this is equivalent to finding nonsingular diagonal matrices D_1 and D_2 such that the largest (in magnitude) element in each row and column of $D_1^{-1}AD_2$ lies between two fixed numbers, say, $1/\beta$ and 1, where β is the base of the number system. Once such D_1 and D_2 are found, the solution of the system

$$Ax = b$$

is obtained by solving the equivalent system

$$\tilde{A}y = \tilde{b}$$

and then computing

$$x = D_2 y$$

where

$$\tilde{A} = D_1^{-1}AD_2$$
$$\tilde{b} = D_1^{-1}b$$

This process is known as **equilibration** (Forsythe and Moler 1967, 44–45).

In conclusion, we note that scaling or equilibration is recommended in general but should be used only on an ad hoc basis, depending upon the data in the problem. "The round-off error analysis for Gaussian elimination gives the most effective results when a matrix is equilibrated." (Forsythe and Moler 1967, 44)

6.4.9 LU Versus QR and a Table of Comparisons

We have just seen that the method for solving $Ax = b$ using QR factorization is about twice as expensive as the method of Gaussian elimination with partial pivoting if the Householder method is used to factor A and about four times as expensive as this method if Givens rotations are used. On the other hand, the QR factorization technique is unconditionally stable, whereas, from a theoretical point of view, with Gaussian elimination with partial or complete pivoting, there is always some risk involved. Thus, if stability is the main concern and the cost is not a major factor, one can certainly use the QR factorization technique. However, considering both efficiency and stability from a practical point of view, it is currently agreed that *Gaussian elimination with partial pivoting is the most practical approach for solution of* $Ax = b$. If we really want to use an orthogonalization technique, certainly the Householder method is to be preferred over the Givens method. Furthermore, *Gaussian elimination without pivoting should not be used unless the matrix A is symmetric positive definite or diagonally dominant.*

We summarize the preceding discussion in Tables 6.1 and 6.2.

TABLE 6.1 Comparison of Different Methods for the Linear System Problem with Arbitrary Matrices

Method	Flop Count (approx.)	Growth Factor ρ	Stability
Gaussian elimination without pivoting	$\frac{n^3}{3}$	Arbitrary	Unstable
Gaussian elimination with partial pivoting	$\frac{n^3}{3}$ $(+ O(n^2)$ comparisons)	$\rho \le 2^{n-1}$	Stable in practice
Gaussian elimination with complete pivoting	$\frac{n^3}{3}$ $(+ O(n^3)$ comparisons)	$\rho \le \{n \cdot 2^1 \cdot 3^{1/2} \cdots n^{(1/n-1)}\}^{1/2}$	Stable
QR factorization using Householder transformations	$\frac{2n^3}{3}$ $(+ n$ square roots)	None	Stable
QR factorization using Givens rotations	$\frac{4n^3}{3}$ $(+ \frac{n^2}{2}$ square roots)	None	Stable

TABLE 6.2 Comparison of Different Methods for the Linear System Problem with Special Matrices

Matrix Type	Method	Flop Count (approx.)	Growth Factor ρ	Stability
Symmetric positive definite	1. Gaussian elimination without pivoting	1. $\frac{n^3}{3}$	1. $\rho = 1$	Stable
	2. Cholesky	2. $\frac{n^3}{6}$ $(+ n$ square roots)	2. None	Stable
Diagonally dominant	Gaussian elimination without pivoting	$\frac{n^3}{3}$	$\rho \le 2$	Stable
Hessenberg	Gaussian elimination with partial pivoting	n^2	$\rho \le n$	Stable
Tridiagonal	Gaussian elimination with partial pivoting	$O(n)$	$\rho \le 2$	Stable

6.5 Inverses and Determinants

Associated with the problem of solving the linear system $Ax = b$ are the problems of finding the determinant, the inverse, and the leading principal minors of the matrix A. In this section we will see how these problems can be solved using the methods of various factorizations developed earlier.

6.5.1 Avoiding Explicit Computation of the Inverses

The inverse of a matrix A very seldom needs to be computed explicitly. Most computational problems involving inverses can be reformulated in terms of solution of linear systems. For example, consider

1. $A^{-1}b$ (inverse times a vector)
2. $A^{-1}B$ (inverse times a matrix)
3. $b^T A^{-1}c$ (vector times inverse times a vector)

The first problem, the computation of $A^{-1}b$, is equivalent to solving the linear system

$$Ax = b$$

Similarly, the second problem can be formulated in terms of solving sets of linear equations. Thus, if A is of order $n \times n$ and B is of order $n \times m$, then write $C = A^{-1}B = (c_1, c_2, \ldots, c_m)$. We see that the columns c_1 through c_m of C can be found by solving the systems

$$Ac_i = b_i, \qquad i = 1, \ldots, m$$

where b_i, $i = 1, \ldots, m$ are the successive columns of the matrix B.

The computation of $b^T A^{-1}c$ can be done in two steps:

1. Find $A^{-1}c$; that is, solve the linear system $Ax = c$.
2. Compute $b^T x$.

As we will see later in this section, *computing A^{-1} is three times as expensive as solving the linear system $Ax = b$.* Thus, all such problems just mentioned can be solved much more efficiently by formulating them in terms of linear systems rather than naively solving them using matrix inversion.

> The explicit computation of the inverse should be avoided whenever possible. A linear system should never be solved by explicit computation of the inverse of the system matrix.

Significance of the Inverse of a Matrix in Practical Applications

Having said that most computational problems involving inverses can be reformulated in terms of linear systems, let us remark that there are, however, certain practical applications where the inverse of a matrix needs to be computed explicitly; in fact, the entries of the inverse matrices in these applications have some physical significance.

Example 6.5.1 Consider once again the spring-mass problem discussed in Section 6.3.3. The (i, j)th entry of the inverse of the stiffness matrix K tells us what the displacement of the mass i will be if a unit external force is imposed on mass j. Thus the entries of K^{-1} will tell us how the systems components will respond to external forces.

Let's take a specific instance when the spring constants are all equal:

$$k_1 = k_2 = k_3 = 1$$

Then

$$K = \begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{pmatrix}, \qquad K^{-1} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 2 \\ 1 & 2 & 3 \end{pmatrix}$$

Because the entries of the first column of K^{-1} are all 1s, it means that a downward unit load to the first mass will make a downward displacement of all the masses by 1 unit. Similar interpretations can be given for the entries of the other columns of K^{-1}.

Some Easily Computed Inverses

Before we discuss the computation of A^{-1} for an arbitrary matrix A, we note that the inverses of some well-known matrices can be trivially computed.

1. The inverse of the elementary lower triangular matrix $M = I - me_k^T$ is given by $M^{-1} = I + me_k^T$.
2. The inverse of an orthogonal matrix Q is its transpose Q^T (note that a Householder matrix and a permutation matrix are orthogonal).
3. The inverse of triangular matrix T of one type is again a triangular matrix of the same type, the diagonal entries being the reciprocals of the diagonal entries of the matrix T.

6.5.2 The Sherman–Morrison and Woodbury Formulas

In many applications, given the inverse of a matrix A, we must find the inverse of another matrix B, which differs from A only by a rank-one perturbation. The question naturally arises if the inverse of B can be computed without starting all over again—that is, if the inverse of B can be found using the inverse of A which has already been computed. The Sherman–Morrison formula shows us how to do this.

> **The Sherman-Morrison Formula**
>
> If u and v are two n-vectors and A is a nonsingular matrix, then
>
> $$(A - uv^T)^{-1} = A^{-1} + \alpha(A^{-1}uv^TA^{-1})$$
>
> where
>
> $$\alpha = \frac{1}{(1 - v^TA^{-1}u)}, \qquad \text{if } v^TA^{-1}u \neq 1$$

Remarks:

1. Item 1 in the list preceding Section 6.5.2 is a special case of this formula.
2. The Sherman–Morrison formula shows how to compute the inverse of the matrix obtained from the matrix A by a rank-one change, once the inverse of A has been computed, without explicitly computing the inverse of the new matrix.

The Sherman–Morrison formula can be extended to the case where U and V are matrices. This generalization is known as the **Woodbury formula**.

> **The Woodbury Formula**
> $(A - UV^T)^{-1} = A^{-1} + A^{-1}U(I - V^TA^{-1}U)^{-1}V^TA^{-1}$, if $I - V^TA^{-1}U$ is nonsingular.

MATCOM Note: The Sherman–Morrison formula has been implemented in MATCOM program SHERMOR.

Example 6.5.2 Given

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 2 & 4 & 5 \\ 6 & 7 & 8 \end{pmatrix}$$

$$A^{-1} = \begin{pmatrix} -3 & -1 & 1 \\ 14 & 2 & -3 \\ -10 & -1 & 2 \end{pmatrix}$$

find $(A - uv^T)^{-1}$, where $u = v = (1, 0, 0)^T$.

$$\alpha = \frac{1}{1 - v^TA^{-1}u} = \frac{1}{4}$$

$$A^{-1} + \alpha A^{-1}uv^TA^{-1} = \begin{pmatrix} -\frac{3}{4} & -\frac{1}{4} & \frac{1}{4} \\ \frac{7}{2} & -\frac{3}{2} & \frac{1}{2} \\ -\frac{5}{2} & \frac{3}{2} & -\frac{1}{2} \end{pmatrix}$$

Thus,

$$(A - uv^T)^{-1} = \begin{pmatrix} -\frac{3}{4} & -\frac{1}{4} & \frac{1}{4} \\ \frac{7}{2} & -\frac{3}{2} & \frac{1}{2} \\ -\frac{5}{2} & \frac{3}{2} & -\frac{1}{2} \end{pmatrix}$$

6.5.3 Computing the Inverse of a Matrix

Computing the inverse of a matrix A is equivalent to solving the sets of linear systems

$$Ax_i = e_i, \qquad i = 1, \ldots, n$$

These n linear systems now can be solved using any of the techniques discussed earlier. However, Gaussian elimination with partial pivoting will be used in practice.

Because the matrix A is the same for all the systems, A has to be factored only once. Thus, if Gaussian elimination with partial pivoting is used, then replacing b_i by e_i in Algorithm 6.4.3, and denoting Me_i by e_i', we have the following.

ALGORITHM 6.5.1 Computing the Inverse of A Using Gaussian Elimination with Partial Pivoting

Step 1: Factor A into $MA = U$ using Algorithm 5.2.2.
Step 2: Solve the n upper triangular systems: $Ux_i = e_i' = Me_i, i = 1, \ldots, n$.

Computational Remark on Step 2: Note that $e_i' = Me_i = M_{n-1}P_{n-1} \cdots M_2P_2M_1P_1$. Thus, e_i' can be computed just by knowing the multipliers m_{ik} and the row-interchange indices r_k from Algorithm 5.2.2. No explicit formation of the matrices M's and P's is necessary.

Example 6.5.3

$$A = \begin{pmatrix} 1 & 2 & 4 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

From Example 5.2.4, we have

$$U = \begin{pmatrix} 7 & 8 & 9 \\ 0 & \frac{6}{7} & \frac{19}{7} \\ 0 & 0 & -\frac{1}{2} \end{pmatrix}, \quad M = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & -\frac{1}{7} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \end{pmatrix}$$

$$x_1 = (1, -2, 1)^T, \qquad x_2 = (-4.6667, 6.3333, 2.000)^T,$$
$$x_3 = (2.6667), -3.3333, 1)^T$$

Thus,

$$A^{-1} = X = \begin{pmatrix} 1 & -4.6667 & 2.6667 \\ -2 & 6.3333 & -3.3333 \\ 1 & -2.0000 & 1 \end{pmatrix}$$

Flop count. The algorithm requires about n^3 flops ($n^3/3$ flops to compute the triangular factorization and $n^2/2$ for the solution of each triangular system and $n^3/6$ flops to compute e_i', $i = 1, \ldots, n$).

6.5.4 Computing the Determinant of a Matrix

In theory, $\det(A) \neq 0$ if and only if A is invertible. However, in computational setting it is not a reliable measure of nonsingularity, as we will see in Section 6.7.4. Thus, *the determinant of a matrix is seldom needed in practice*. However, if one needs to compute the determinant of A, then the ordinary Gaussian elimination (with partial and complete pivoting) or any orthogonal triangularization method can be used.

If Gaussian elimination without pivoting is used, we have

$$A = LU, \qquad \det(A) = \det(L) \cdot \det(U)$$

U is an upper triangular matrix, so, $\det(U) = u_{11}u_{22}\cdots u_{nn} = a_{11}a_{22}^{(1)}\cdots a_{nn}^{(n-1)}$; L is a unit lower triangular matrix, so $\det(L) = 1$.

Computing det(*A*) from LU Factorization

$$\det(A) = a_{11}a_{22}^{(1)}\cdots a_{nn}^{(n-1)} = \text{the product of the pivots}$$

If Gaussian elimination with partial pivoting is used, we have

$$MA = U$$

Then, $\det(M)\det(A) = \det(U)$. Now, $M = M_{n-1}P_{n-1}\cdots M_2P_2M_1P_1$. Because the determinant of each of the lower elementary matrices is 1 and the determinant of each of the permutation matrices ± 1, we have

$$\det(M) = (-1)^r$$

where r is the number of row interchanges made during the pivoting process. So, we have

$$\begin{aligned}\det(A) &= \frac{1}{\det(M)}\det(U)\\ &= (-1)^r \cdot u_{11}\cdot u_{22}\cdots u_{nn}\\ &= (-1)^r a_{11}a_{22}^{(1)}\cdots a_{nn}^{(n-1)}\end{aligned}$$

Computing det(*A*) from *MA* = *U* Factorization

$$\det(A) = (-1)^r a_{11}a_{22}^{(1)}\cdots a_{nn}^{(n-1)}$$

where r is the number of interchanges.

If Gaussian elimination with complete pivoting is used, we have

$$MAQ = U$$

So,

$$\det(M)\cdot\det(A)\cdot\det(Q) = \det(U)$$

Let r and s be, respectively, the number of row and column interchanges. Then

$$\det(M) = (-1)^r$$
$$\det(Q) = (-1)^s$$

Thus we state the following.

Computing det(*A*) from *MAQ* = *U* Factorization

$$\begin{aligned}\det(A) &= (-1)^{r+s}\det(U)\\ &= (-1)^{r+s}a_{11}a_{22}^{(1)}\cdots a_{nn}^{(n-1)}\end{aligned}$$

where r and s are the number of row and column interchanges.

Example 6.5.4

$$A = \begin{pmatrix} 0 & 1 & 1\\ 1 & 2 & 3\\ 1 & 1 & 1\end{pmatrix}$$

Using Gaussian elimination with partial pivoting (Example 5.2.3),

$$U = \begin{pmatrix} 1 & 2 & 3\\ 0 & 1 & 1\\ 0 & 0 & -1\end{pmatrix}$$

Only one interchange occurs; therefore, $r = 1$. $\det(A) = (-1)\det(U) = (-1)(-1) = 1$.

Using Gaussian elimination with complete pivoting (Example 5.2.5),

$$U = \begin{pmatrix} 3 & 1 & 2\\ 0 & \frac{2}{3} & \frac{1}{3}\\ 0 & 0 & \frac{1}{2}\end{pmatrix}$$

In the first step, one row interchange and one column interchange occurred. In the second step, one row interchange and one column interchange occurred. Thus $r = 2, s = 2$, and

$$\det(A) = (-1)^{r+s}\det(U) = (-1)^4 3\cdot\tfrac{2}{3}\cdot\tfrac{1}{2} = 1$$

6.6 Perturbation Analysis of the Linear System Problem

In practice, the input data A and b may be contaminated by error. This error may be experimental, may come from the process of discretization, and so on. In order to estimate the accuracy of the computed solution, the error in the data should be taken into account. As we saw in Chapter 3, there are problems whose solutions may change drastically even with small changes in the input data, and this phenomenon of ill-conditioning is independent of the algorithms used to solve these problems. We discussed ill-conditioning of several problems in Chapter 3. Let's look at another simple example of an ill-conditioned linear system.

Consider the following linear system:

$$\begin{aligned} x_1 + 2x_2 &= 3\\ 2x_1 + 3.999x_2 &= 5.999\end{aligned}$$

The exact solution is $x_1 = x_2 = 1$. Now make a small perturbation in the right-hand side, obtaining the system

$$\begin{aligned} x_1 + 2x_2 &= 3 \\ 2x_1 + 3.999x_2 &= 6 \end{aligned}$$

The solution of the perturbed system now, obtained by Gaussian elimination with pivoting (considered to be a stable method in practice) is:

$$x_1 = 3, \qquad x_2 = 0$$

Thus, *a very small change in the right-hand side changed the solution altogether.*

In this section we study the effect of small perturbations of the input data A and b on the computed solution x of the system $Ax = b$. This study is very useful. Not only will this study help us in assessing an amount of error in the solution of the perturbed system, regardless of the algorithm used, but also, when the result of a perturbation analysis is combined with that of backward error analysis of a particular algorithm, it will help us obtain an error bound in the computed solution by the algorithm.

Because A and b are the input data in the linear system problem $Ax = b$, there could be impurities either in b or in A or in both. We will, therefore, consider the effect of perturbations on the solution x in each of these cases separately.

6.6.1 Effect of Perturbation in the Right-Hand-Side Vector b

We assume here that there are impurities in b, but the matrix A is exact.

THEOREM 6.6.1 Right Perturbation Theorem If δb and δx, are, respectively, the perturbations of b and x in the linear system $Ax = b$ and A is assumed to be nonsingular and $b \neq 0$, then

$$\frac{\|\delta b\|}{\text{Cond}(A)\,\|b\|} \leq \frac{\|\delta x\|}{\|x\|} \leq \text{Cond}(A)\,\frac{\|\delta b\|}{\|b\|}$$

Proof: We have

$$Ax = b$$

and

$$A(x + \delta x) = b + \delta b$$

The last equation can be written as

$$Ax + A\delta x = b + \delta b$$

or

$$A\delta x = \delta b, \qquad \text{because } Ax = b$$

that is,

$$\delta x = A^{-1}\delta b$$

Taking a subordinate matrix–vector norm, we get

$$\|\delta x\| \leq \|A^{-1}\|\,\|\delta b\| \tag{6.6.1}$$

Again, taking the same norm on both sides of $Ax = b$, we get

$$\|Ax\| = \|b\|$$

or

$$\|b\| = \|Ax\| \leq \|A\|\,\|x\| \tag{6.6.2}$$

Combining (6.6.1) and (6.6.2), we have

$$\frac{\|\delta x\|}{\|x\|} \leq \|A\|\,\|A^{-1}\|\frac{\|\delta b\|}{\|b\|} \tag{6.6.3}$$

On the other hand, $A\delta x = \delta b$ gives

$$\|\delta x\| \geq \frac{\|\delta b\|}{\|A\|} \tag{6.6.4}$$

Also, from $Ax = b$, we have

$$\frac{1}{\|x\|} \geq \frac{1}{\|A^{-1}\|\,\|b\|} \tag{6.6.5}$$

Combining (6.6.4) and (6.6.5), we have

$$\frac{\|\delta x\|}{\|x\|} \geq \frac{\|\delta b\|}{\|A\|\,\|A^{-1}\|\,\|b\|}$$

Recall from Chapter 3 that $\|A\|\,\|A^{-1}\|$ is the *condition number* of A and is denoted by Cond(A). The theorem is, therefore, proved. ■

Interpretation of Theorem 6.6.1. It is important to understand completely the implication of Theorem 6.6.1. *Theorem 6.6.1 says that a relative change in the solution can be as large as* Cond(A) *multiplied by the relative change in the vector b.* Thus, if the condition number is not too large, then a small perturbation in the vector b will have very little effect on the solution. On the other hand, if the condition number is large, then even a small perturbation in b might change the solution drastically.

Example 6.6.1 An ill-conditioned problem

$$A = \begin{pmatrix} 1 & 2 & 1 \\ 2 & 4.0001 & 2.002 \\ 1 & 2.002 & 2.004 \end{pmatrix}, \qquad b = \begin{pmatrix} 4 \\ 8.0021 \\ 5.006 \end{pmatrix}$$

The exact solution is

$$x = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

Change b to

$$b' = \begin{pmatrix} 4 \\ 8.0020 \\ 5.0061 \end{pmatrix}$$

Then the relative change in b is

$$\frac{\|b' - b\|}{\|b\|} = \frac{\|\delta b\|}{\|b\|} = 1.379 \times 10^{-5} \qquad \text{(small)}$$

If we solve the system $Ax' = b'$, we get

$$x' = x + \delta x = \begin{pmatrix} 3.0850 \\ -0.0436 \\ 1.0022 \end{pmatrix}$$

(x' is completely different from x).

Note: $\dfrac{\|\delta x\|}{\|x\|} = 1.3461$

It is easily verified that the inequality in Theorem 6.6.1 is satisfied:

$$\text{Cond}(A) \cdot \frac{\|\delta b\|}{\|b\|} = 4.4434$$

However, the predicted change is overly estimated.

Example 6.6.2 A well-conditioned problem

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}, \qquad b = \begin{pmatrix} 3 \\ 7 \end{pmatrix}$$

The exact solution is

$$x = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Let

$$b' = b + \delta b = \begin{pmatrix} 3.0001 \\ 7.0001 \end{pmatrix}$$

The relative change in b is

$$\frac{\|b' - b\|}{\|b\|} = 1.875 \times 10^{-5} \qquad \text{(small)}$$

$$\text{Cond}(A) = 14.9330 \qquad \text{(small)}$$

Thus, a drastic change in the solution x is not expected. In fact, x' satisfying

$$Ax' = b'$$

is

$$x' = \begin{pmatrix} 0.9999 \\ 1.0001 \end{pmatrix} \approx x = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Note: $\dfrac{\|\delta x\|}{\|x\|} = 10^{-5}$

6.6.2 Effect of Perturbation in Matrix *A*

Here we assume that there are impurities in A only, and as a result we have $A + \Delta A$ in hand, but b is exact.

THEOREM 6.6.2 Left Perturbation Theorem Assume A is nonsingular and $b \neq 0$. Suppose that ΔA and δx are, respectively, the perturbations of A and x in the linear system

$$Ax = b$$

Furthermore, assume that ΔA is such that $\|\Delta A\| < \dfrac{1}{\|A^{-1}\|}$. Then

$$\frac{\|\delta x\|}{\|x\|} \leq \text{Cond}(A)\frac{\|\Delta A\|}{\|A\|} \Big/ \left(1 - \text{Cond}(A)\frac{\|\Delta A\|}{\|A\|}\right)$$

Proof: We have

$$(A + \Delta A)(x + \delta x) = b$$

or

$$(A + \Delta A)x + (A + \Delta A)\delta x = b \tag{6.6.6}$$

Because

$$Ax = b$$

we have, from (6.6.6),

$$(A + \Delta A)\delta x = -\Delta A x \tag{6.6.7}$$

or

$$\delta x = -A^{-1}\Delta A(x + \delta x) \tag{6.6.8}$$

Taking the norm on both sides, we have

$$\|\delta x\| \leq \|A^{-1}\|\|\Delta A\| \cdot (\|x\| + \|\delta x\|) \tag{6.6.9}$$

$$= \frac{\|A^{-1}\|\|A\|\|\Delta A\|}{\|A\|}(\|x\| + \|\delta x\|)$$

that is,

$$\left(1 - \frac{\|A\|^{-1}\,\|A\|\,\|\Delta A\|}{\|A\|}\right)\|\delta x\| \leq \frac{\|A\|\,\|A^{-1}\|\,\|\Delta A\|}{\|A\|}\|x\| \tag{6.6.10}$$

Because

$$\|A^{-1}\|\,\|\Delta A\| < 1$$

the expression in parentheses on the left-hand side is positive. We can thus divide both sides of the inequality by this number without changing the inequality. After this, if we also divide by $\|x\|$, we obtain

$$\frac{\|\delta x\|}{\|x\|} \le \frac{\|A\|\|A^{-1}\|\dfrac{\|\Delta A\|}{\|A\|}}{\left(1 - \|A\|\|A^{-1}\|\dfrac{\|\Delta A\|}{\|A\|}\right)}$$

$$= \frac{\mathrm{Cond}(A)\dfrac{\|\Delta A\|}{\|A\|}}{\left(1 - \mathrm{Cond}(A)\dfrac{\|\Delta A\|}{\|A\|}\right)} \tag{6.6.11}$$

which proves the theorem. ■

Remarks: Because of the assumption that $\|\Delta A\| < 1/\|A^{-1}\|$ (which is quite reasonable), the denominator on the right hand side of the inequality in Theorem 6.6.2 is less than 1. Thus, *even if* $\|\Delta A\|/\|A\|$ *is small, there could be a drastic change in the solution if* Cond(A) *is large.*

Example 6.6.3 Consider Example 6.6.1 once more. Change $a_{2,3}$ to 2.0021; keep b fixed. Then

$$\Delta A = -10^{-4}\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \qquad \text{(small)}$$

Now solve the system $(A + \Delta A)x' = b$:

$$x' = \begin{pmatrix} 3.0852 \\ -0.0437 \\ 1.0021 \end{pmatrix}$$

$$\delta x = x' - x = \begin{pmatrix} 2.0852 \\ -1.0437 \\ 0.0021 \end{pmatrix}$$

$$\text{Relative error} = \frac{\|\delta x\|}{\|x\|} = 1.3463 \qquad \text{(quite large)}$$

6.6.3 Effect of Perturbations in Both Matrix *A* and Vector *b*

Finally, we assume now that both the input data A and b have impurities. As a result we have the system with $A + \Delta A$ as the matrix and $b + \delta b$ as the right-hand-side vector.

THEOREM 6.6.3 General Perturbation Theorem Assume that A is nonsingular, $b \neq 0$, and $\|\Delta A\| < \dfrac{1}{\|A^{-1}\|}$. Then

$$\frac{\|\delta x\|}{\|x\|} \le \left(\frac{\mathrm{Cond}(A)}{1 - \mathrm{Cond}(A)\cdot\dfrac{\|\Delta A\|}{\|A\|}}\right)\left(\frac{\|\Delta A\|}{\|A\|} + \frac{\|\delta b\|}{\|b\|}\right)$$

Proof: Subtracting

$$Ax = b$$

from

$$(A + \Delta A)(x + \delta x) = b + \delta b$$

we have

$$(A + \Delta A)(x + \delta x) - Ax = \delta b$$

or

$$(A + \Delta A)(x + \delta x) - (A + \Delta A)x + (A + \Delta A)x - Ax = \delta b$$

or

$$(A + \Delta A)(\delta x) + \Delta A x = \delta b$$

or

$$A(I - A^{-1}(-\Delta A))\delta x = \delta b - \Delta A x \tag{6.6.12}$$

Let $A^{-1}(-\Delta A) = F$. Then

$$\|F\| = \|A^{-1}(-\Delta A)\| \le \|A^{-1}\|\,\|\Delta A\| < 1 \qquad \text{(by assumption)}$$

Because $\|F\| < 1$, $I - F$ is invertible, and

$$\|(I - F)^{-1}\| \le \frac{1}{1 - \|F\|} \qquad \text{(See Theorem 1.7.7.)} \tag{6.6.13}$$

From (6.6.12), we then have

$$\delta x = (I - F)^{-1}A^{-1}(\delta b - \Delta A x)$$

or

$$\|\delta x\| \le \frac{\|A^{-1}\|}{1 - \|F\|}(\|\delta b\| + \|\Delta A\|\,\|x\|)$$

or

$$\frac{\|\delta x\|}{\|x\|} \le \frac{\|A^{-1}\|}{(1 - \|F\|)}\cdot\left(\frac{\|\delta b\|}{\|x\|} + \|\Delta A\|\right) \tag{6.6.14}$$

$$\le \frac{\|A^{-1}\|}{(1 - \|F\|)}\left(\frac{\|\delta b\|\,\|A\|}{\|b\|} + \|\Delta A\|\right) \qquad \left(\text{Note that } \frac{1}{\|x\|} \le \frac{\|A\|}{\|b\|}.\right)$$

That is,

$$\frac{\|\delta x\|}{\|x\|} \le \frac{\|A^{-1}\|\,\|A\|}{(1-\|F\|)}\left(\frac{\|\delta b\|}{\|b\|}+\frac{\|\Delta A\|}{\|A\|}\right) \tag{6.6.15}$$

Again,

$$\|F\| = \|A^{-1}(-\Delta A)\| \le \|A^{-1}\|\,\|\Delta A\| = \frac{\|A^{-1}\|\,\|A\|}{\|A\|}\cdot\|\Delta A\| \tag{6.6.16}$$

Because $\|F\| \le 1$, we can write, from (6.6.15) and (6.6.16),

$$\begin{aligned}\frac{\|\delta x\|}{\|x\|} &\le \left(\frac{\|A^{-1}\|\,\|A\|}{\left(1-\left(\frac{\|A^{-1}\|\,\|A\|}{\|A\|}\right)\cdot\|\Delta A\|\right)}\right)\left(\frac{\|\delta b\|}{\|b\|}+\frac{\|\Delta A\|}{\|A\|}\right)\\ &= \left(\frac{\mathrm{Cond}(A)}{\left(1-\frac{\mathrm{Cond}(A)}{\|A\|}\cdot\|\Delta A\|\right)}\right)\left(\frac{\|\delta b\|}{\|b\|}+\frac{\|\Delta A\|}{\|A\|}\right)\end{aligned} \tag{6.6.17}$$

■

Remarks: We again see from (6.6.17) that even if the perturbations

$$\frac{\|\delta b\|}{\|b\|} \quad \text{and} \quad \frac{\|\Delta A\|}{\|A\|}$$

are small, there might be a drastic change in the solution if Cond(A) is large. Thus, Cond(A) *plays the crucial role in the sensitivity of the solution.*

A Convention

Unless otherwise stated, when we write Cond(A), we will mean $\mathrm{Cond}_2(A)$—that is, the condition number with respect to a 2-norm. The condition number of a matrix A with respect to a subordinate p-norm ($p = 1, 2, \infty$) will be denoted by $\mathrm{Cond}_p(A)$; that is, $\mathrm{Cond}_1(A)$ will stand for the condition number of A with respect to 1-norm, and so on.

6.7 The Condition Number and Accuracy of Solution

The following are some important (but easy to prove) properties of the condition number of a matrix.

1. Cond(A) with respect to any p-norm is at least 1.
2. If A is an orthogonal matrix, then Cond(A) with respect to the 2-norm is 1. (Note that this property of an orthogonal matrix A makes the matrix attractive for use in numerical computations.)
3. $\mathrm{Cond}(A^TA) = (\mathrm{Cond}(A))^2$.
4. $\mathrm{Cond}(A) = \mathrm{Cond}(A^T)$.

5. $\mathrm{Cond}(AB) \le \mathrm{Cond}(A)\,\mathrm{Cond}(B)$.
6. $\mathrm{Cond}(\alpha A) = \mathrm{Cond}(A)$, where α is a nonzero scalar.
7. $\mathrm{Cond}(A) \ge |\lambda_1|/|\lambda_n|$, where $|\lambda_1| \ge |\lambda_2| \ge \cdots \ge |\lambda_n|$, and $\lambda_1, \ldots, \lambda_n$ are the eigenvalues of A.
8. $\mathrm{Cond}(A) = \sigma_1/\sigma_n$, where $\sigma_1 \ge \sigma_2 \ge \cdots \ge \sigma_n$ are the singular values of A.

We now formally define ill-conditioning and well-conditioning in terms of the condition number.

DEFINITION 6.7.1 The system $Ax = b$ is **ill conditioned** if Cond(A) is quite large. Otherwise, it is **well conditioned**.

Remark: Note that a system can be artificially ill-conditioned because of poor scaling. Thus, before deciding whether a system is really ill-conditioned, its entries should be properly scaled. (See Section 6.7.7 for more on condition numbers and scaling.)

Conditioning with Respect to Different Norms

Although the condition number, as just defined, is norm-dependent, the condition numbers with respect to two different norms are related. (For example, it can be shown that if A is an $n \times n$ matrix, then $1/n \le \mathrm{Cond}_2(A)/\mathrm{Cond}_\infty(A) \le n$.) In general, *if a matrix is well conditioned or ill conditioned with respect to one norm, it is also ill conditioned or well conditioned with respect to some other norms.*

Example 6.7.1 Consider

$$A = \begin{pmatrix} 1 & 0.9999 \\ 0.9999 & 1 \end{pmatrix}$$

$$A^{-1} = 10^3\begin{pmatrix} 5.0003 & -4.99997 \\ -4.9997 & 5.0003 \end{pmatrix}$$

The condition numbers with respect to the infinity norm and 1-norm are

$$\begin{aligned}\|A\|_\infty &= \|A\|_1 = 1.9999\\ \|A^{-1}\|_\infty &= \|A^{-1}\|_1 = 10^4\\ \mathrm{Cond}_\infty(A) &= \mathrm{Cond}_1(A) = 1.9999 \times 10^4\end{aligned}$$

The condition number with respect to the 2-norm is

$$\begin{aligned}\|A\|_2 &= \rho(A) = 1.9999\\ \|A^{-1}\|_2 &= \rho(A^{-1}) = 10^4\\ \mathrm{Cond}_2(A) &= 1.9999 \times 10^4\end{aligned}$$

Remark: For Example 6.7.1, the condition number with respect to any norm is the same. However, this is not always the case. In general, though, they are closely related. (See the following example on the Hilbert matrix with respect to different norms.)

6.7.1 Some Well-known Ill-conditioned Matrices

The following are among the well-known ill-conditional matrices.

1. The Hilbert matrix:

$$A = \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \cdots & \frac{1}{n} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \cdots & \frac{1}{n+1} \\ \vdots & \vdots & \vdots & & \vdots \\ \frac{1}{n} & \frac{1}{n+1} & \frac{1}{n+2} & \cdots & \frac{1}{2n-1} \end{pmatrix}$$

For $n = 10, \text{Cond}_2(A) = 1.6025 \times 10^{13}$

$$\text{Cond}_\infty(A) = 3.5353 \times 10^{13}$$
$$\text{Cond}_1(A) = 3.5353 \times 10^{13}$$

2. The Pie matrix A with $a_{ii} = \alpha, a_{ij} = 1$ for $i \neq j$. The matrix becomes ill-conditioned when α is close to 1. For example, when $\alpha = 0.9999$ and $n = 5$, $\text{Cond}(A) = 5 \times 10^4$.
3. The Wilkinson bidiagonal matrix of order 20 (see Chapter 3).

6.7.2 Effect of the Condition Number on Accuracy of the Computed Solution

Once a solution $\hat{x}$ of the system $Ax = b$ has been computed, it is natural to test how accurate the computed solution $\hat{x}$ is. If the exact solution x is known, then one could of course compute the relative error $\|x - \hat{x}\|/\|x\|$ to test $\hat{x}$. However, in most practical situations, the exact solution is not known. In such cases, the most obvious thing to do is to compute the residual $r = b - A\hat{x}$ and see how small the relative residual $\|r\|/\|b\|$ is. Interestingly, the solution obtained by the Gaussian elimination process, in general, produces a small residual. (Why?) Unfortunately, a small relative residual does not guarantee the accuracy of the solution. The following example illustrates this fact.

Example 6.7.2 Let

$$A = \begin{pmatrix} 1.0001 & 1 \\ 1 & 1 \end{pmatrix}$$
$$b = \begin{pmatrix} 2.0001 \\ 2 \end{pmatrix}$$

Let

$$\hat{x} = \begin{pmatrix} 0 \\ 2 \end{pmatrix}$$

Then

$$r = b - A\hat{x} = \begin{pmatrix} 0.0001 \\ 0 \end{pmatrix}$$

Note that r is small. However, the vector $\hat{x}$ is nowhere close to the exact solution

$$x = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

The preceding phenomenon can be explained mathematically from the following theorem. The proof can be easily worked out.

THEOREM 6.7.1 Residual Theorem

$$\frac{\|\hat{x} - x\|}{\|x\|} \leq \text{Cond}(A)\frac{\|r\|}{\|b\|}$$

Interpretation of Theorem 6.7.1

Theorem 6.7.1 tells us that the relative error in $\hat{x}$ depends not only on the relative residual, but also on the condition number of the matrix A. A computed solution can be guaranteed to be accurate only when the product of both $\text{Cond}(A)$ and the relative residual is small. Note that in Example 6.7.2,

$$\text{Cond}(A) = 4.0002 \times 10^4 \qquad \text{(large)}$$

6.7.3 Size of the Condition Number Required for Ill-Conditioning

A frequently asked question is, How large does $\text{Cond}(A)$ have to be before the system $Ax = b$ is considered to be ill conditioned? We restate Theorem 6.6.3 to answer the question.

RESTATEMENT OF THEOREM 6.6.3

$$\frac{\|\delta x\|}{\|x\|} \leq \left(\frac{\text{Cond}(A)}{1 - \text{Cond}(A)\frac{\|\Delta A\|}{\|A\|}} \right) \left(\frac{\|\Delta A\|}{\|A\|} + \frac{\|\delta b\|}{\|b\|} \right)$$

Suppose, for simplicity, that

$$\frac{\|\Delta A\|}{\|A\|} = \frac{\|\delta b\|}{\|b\|} = 10^{-d}, \text{ and } \text{Cond}(A)\frac{\|\Delta A\|}{\|A\|} << 1$$

Then, $\|\delta x\|/\|x\|$ is approximately less than or equal to $2 \times \text{Cond}(A) \times 10^{-d}$.

Thus, if the data have a relative error of 10^{-d} and if the relative error in the solution has to be guaranteed to be less than or equal to 10^{-t}, then $\text{Cond}(A)$ has to be less than or equal to $\frac{1}{2} \times 10^{d-t}$. Thus, *whether a system is ill conditioned or well conditioned depends on the accuracy of the data and how much error in the solution can be tolerated.*

For example, suppose that the data have a relative error of about 10^{-5} and an accuracy of about 10^{-3} is sought; then $\text{Cond}(A) \leq \frac{1}{2} \times 10^2 = 50$. On the other hand, if the accuracy of about 10^{-2} is sought, then $\text{Cond}(A) \leq \frac{1}{2} \times 10^3 = 500$. Thus, in the first case the system will be well conditioned if $\text{Cond}(A)$ is less than or equal to 50, whereas in the second case, the system will be well conditioned if $\text{Cond}(A)$ is less than or equal to 500.

Estimating Accuracy from the Condition Number

In general, if the data are approximately accurate and if $\text{Cond}(A) = 10^s$, then there will be only about $(t - s)$-significant-digit accuracy in the computed solution when the solution is computed in t-digit arithmetic.

For better understanding of conditioning, stability, and accuracy, we again refer the readers to Bunch (1987).

6.7.4 The Condition Number and Nearness to Singularity

The condition number also gives an indication when a matrix A is relatively close to a singular matrix: If $\text{Cond}(A)$ is large, A is close to singular.

This measure of nearness to singularity is a more accurate measure than the determinant of A. For example, consider the well-known $n \times n$ triangular matrix

$$A = \begin{pmatrix} 1 & -1 & -1 & \cdots & -1 & -1 \\ 0 & 1 & -1 & \cdots & -1 & -1 \\ \vdots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & \ddots & \ddots & -1 \\ 0 & \cdots & \cdots & \cdots & 0 & 1 \end{pmatrix}$$

The matrix has the determinant equal to 1; however, it is nearly singular for large n.

$$\text{Cond}_\infty(A) = n2^{n-1}$$

Similarly, the smallness of the determinant of a matrix does not necessarily mean that A is close to a singular matrix. Consider $A = \text{diag}(0.1, 0.1, \ldots, 0.1)$ of order 1000; $\det(A) = 10^{-1000}$, which is a small number. However, A is considered to be perfectly nonsingular, because $\text{Cond}_2(A) = 1$.

6.7.5 Conditioning and Pivoting

It is natural to wonder if ill-conditioning can be detected during the triangularization process using the Gaussian elimination with partial pivoting. By a normalized matrix, we mean here that $\|A\|_2 = 1$. Suppose that A, and b have been normalized. Then there are certain symptoms for ill-conditioning. Among the symptoms are the following.

Symptoms for Ill-Conditioning

1. A small pivot
2. A large computed solution
3. A large residual vector

Justification: Suppose there is a small pivot. Then the inverse of the U matrix in the triangular factorization of A will be large (see Algorithm 5.2.2 or Algorithm 5.2.3), and this large U^{-1} will make A^{-1} large. (Note that if partial pivoting is used, then $A^{-1} = U^{-1}M$). Similarly, if the computed solution $\hat{x}$ is large, then from $A\hat{x} = \hat{b}$, we have $\|\hat{x}\| = \|A^{-1}\hat{b}\| \leq \|A^{-1}\| \, \|\hat{b}\|$, showing that $\|A^{-1}\|$ is possibly large. Large $\|A^{-1}\|$, of course, means ill-conditioning, because $\text{Cond}(A) = \|A\| \, \|A^{-1}\|$ will then be large.

Remark: There are matrices that do not have any of these symptoms but are still ill-conditioned (Wilkinson 1965, 254–255).

6.7.6 Conditioning and the Eigenvalue Problem

If a *normalized* matrix A has a small eigenvalue, it must be ill conditioned. If $\lambda_1, \lambda_2, \ldots, \lambda_n$ are the eigenvalues of A, then it can be shown that

$$\|A^{-1}\|_2 \geq |\text{maximum eigenvalue of } A^{-1}| = \frac{1}{\min |\lambda_i|}$$

Example 6.7.3 Consider the linear system

$$Ax = b$$

with

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0.00001 & 0 \\ 0 & 0 & 0.00001 \end{pmatrix}$$

$$b = \begin{pmatrix} 0.1 \\ 0.1 \\ 0.1 \end{pmatrix}$$

$$x = 10^4 \begin{pmatrix} 0.00001 \\ 1 \\ 1 \end{pmatrix}$$

which is quite large.

The eigenvalues of A are 1, 0.00001, and 0.00001. Thus, A has a small eigenvalue.

$$A^{-1} = 10^5 \begin{pmatrix} 0.00001 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

which is large. Thus, for this example (1) the computed solution is large, (2) A has a small eigenvalue, and (3) A^{-1} is large. A is, therefore, likely to be ill conditioned. It is indeed true:

$$\text{Cond}(A) = 10^5$$

6.7.7 Conditioning and Scaling

In Section 6.4.8 we discussed scaling; the message there was that scaling is, in general, recommended if the entries of the matrix A vary widely. Scaling followed by a strategy of pivoting is helpful. One thing that we did not make clear is that *scaling has some effect on the condition number of the matrix.* For example, consider again the example of Section 6.4.8:

$$A = \begin{pmatrix} 10 & 10^6 \\ 1 & 1 \end{pmatrix}$$

$$\text{Cond}(A) = 10^6$$

However, if the first row of A is scaled to obtain

$$\tilde{A} = \begin{pmatrix} 0.00001 & 1 \\ 1 & 1 \end{pmatrix}$$

then

$$\text{Cond}(\tilde{A}) = 2.6181$$

The question naturally arises, Given a matrix A, how can we choose the diagonal matrices D_1 and D_2 such that $\text{Cond}(D_1^{-1}AD_2)$ will be as small as possible?

There is an almost classical solution due to Bauer (1963) for this problem. Unfortunately, the solution is not practical. For example the infinity-norm solution requires knowing the eigenvalue of maximum modulus and the corresponding eigenvector of the nonnegative matrix $C = |A| \cdot |A^{-1}|$, and to solve $Ax = b$, we will not know A^{-1} in advance. For details of the method, see Forsythe and Moler (1967, 43–44).

Remark: Demmel (1989) has shown that scaling to improve the condition number is not necessary when solving a symmetric positive definite system using the Cholesky algorithm. The error bound obtained for the solution by the algorithm for the unscaled system $Ax = b$ is almost the same as that of the scaled system with $\tilde{A} = D^{-1}AD^{-1}$, $D = \text{diag}(\sqrt{a_{11}}, \ldots, \sqrt{a_{nn}})$. See also Higham (1995).

6.7.8 Computing and Estimating the Condition Number

The obvious way to compute the condition number is to compute it from its definition:

1. Compute A^{-1}.
2. Compute $\|A\|$ and $\|A^{-1}\|$, and find their product.

We have seen that computing the inverse of A requires about n^3 flops. Thus, this approach is three times as expensive as finding the solution to $Ax = b$ itself. On

the other hand, to compute Cond(A), we need to know only $\|A^{-1}\|$, not the inverse itself. Furthermore, *the exact value of* Cond(A) *itself is seldom needed; an estimate is sufficient.* The question, therefore, arises whether we can get a reasonable estimate of $\|A^{-1}\|$ without computing the inverse of A explicitly.

In this context, we note that if y is any nonzero n-vector, then from

$$Az = y$$

we have

$$\|z\| = \|A^{-1}y\| \le \|A^{-1}\| \, \|y\|$$

Then,

$$\|A^{-1}\| \ge \frac{\|z\|}{\|y\|}, \qquad y \ne 0$$

Thus, if we choose y such that $\|z\|/\|y\|$ is quite large, we could have a reasonably good estimate of $\|A^{-1}\|$. Rice (1981, 93) remarks, "There is a heuristic argument which says that if y is picked at random, then the expected value of $\|z\|/\|y\|$ is about $\frac{1}{2}\|A^{-1}\|$."

A systematic way to choose y has been given in LINPACK (1979). It is based on an algorithm by Cline, Moler, Stewart, and Wilkinson (1979). The process involves solving two systems of equations,

$$A^T y = e$$

and

$$Az = y$$

where e is a scalar multiple of a vector with components ± 1 chosen in such a way that the possible growth is maximum. For details, see LINPACK (1979) or Cline, Moler, Stewart, and Wilkinson (1979).

To avoid overflow, the LINPACK condition estimator routine SGECO actually computes an estimate of $1/\text{Cond}(A)$, called

$$\text{RCOND} = \frac{\|y\|}{\|A\| \, \|z\|}$$

An Optimization Technique for Estimating $\|A^{-1}\|_1$

Hager (1984) has proposed a method for estimating $\|A^{-1}\|$ based on an optimization technique. This technique seems to be quite suitable for randomly generated matrices. Let $A^{-1} = B = (b_{ij})$.

Define a function $f(x)$:

$$f(x) = \|Bx\|_1 = \sum_{i=1}^{n} \left| \sum_{j=1}^{n} b_{ij} x_j \right|$$

Then

$$\|B\|_1 = \|A^{-1}\|_1 = \max\{f(x) : \|x\|_1 \le 1\}$$

Thus, the problem is to find maximum of the convex function f over the convex set

$$S = \{x \in R^n : \|x\|_1 \le 1\}$$

It is well known that the maximum of a convex function (defined on a convex, compact set) is obtained at an extreme point. Hager's method consists in finding this maximum systematically. We present the algorithm here (for details, see Hager (1984)). Hager remarks that the algorithm usually stops after two iterations. An excellent survey of different condition number estimators, including Hager's, and their performances has been given by Higham (1987).

ALGORITHM 6.7.1 Hager's Norm-1 Condition Number Estimator

- Set $\rho = \|A^{-1}\|_1 = 0$.

Set $b = \begin{pmatrix} \frac{1}{n} \\ \frac{1}{n} \\ \vdots \\ \frac{1}{n} \end{pmatrix}$

Step 1: Solve $Ax = b$.

Step 2: Test if $\|x\| \le \rho$. If so, go to step 6. Otherwise set $\rho = \|x\|_1$ and go to step 3.

Step 3: Solve $A^T z = y$, where

$$y_i = 1 \quad \text{if} \quad x_i \ge 0$$
$$y_i = -1 \quad \text{if} \quad x_i < 0$$

Step 4: Set $j = \arg\max\{|z_i|, i = 1 \text{ to } n\}$.

Step 5: If $|z_j| > z^T b$, update

$$b \equiv \begin{pmatrix} 0 \\ \vdots \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \qquad j\text{th entry}$$

and return to step 1. Else go to step 6.

Step 6: Set $\|A^{-1}\|_1 \approx \rho$. Then $\text{Cond}_1(A) = \rho\|A\|_1$.

LAPACK and MATCOM Notes: Hager's condition number has been implemented in MATCOM program HAGCOND1. Algorithm 6.7.1 has also been implemented in LAPACK. The LAPACK routine SGECON computes an estimate of the reciprocal of the 1-norm condition number of A using Higham's modification (Higham, 1988) of Hager's algorithm.

Example 6.7.4 We illustrate Hager's method by means of a very ill conditioned matrix.

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{pmatrix}, \qquad \text{Cond}(A) = 3.3819 \times 10^{16}$$

Iteration 1:

$$b = \begin{pmatrix} \frac{1}{3} \\ \frac{1}{3} \\ \frac{1}{3} \end{pmatrix}$$

$$x = \begin{pmatrix} 1.0895 \\ -2.5123 \\ 1.4228 \end{pmatrix}$$

$$\rho = \|x\|_1 = 5.0245$$

$$y = \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix}$$

$$z = 10^{16} \begin{pmatrix} 2.0271 \\ -3.3785 \\ 1.3514 \end{pmatrix}$$

$$j = 2$$

$$|z_2| = 10^{16}(3.3785) > z^T b = -1.3340$$

Update

$$b \equiv \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$$

Iteration 2:

$$b = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$$

$$x = 10^{17} \begin{pmatrix} -1.3564 \\ 2.7128 \\ -1.3564 \end{pmatrix}$$

$$\|x\|_1 = 5.4255 \times 10^{17}$$

Since $\|x\|_1 > \rho$, we set $\rho = 5.4255 \times 10^{17}$, and continue.

Comment: It turns out that this current value is an excellent estimate of $\|A^{-1}\|_1$.

Condition Estimation from Triangularization

If we use the QR factorization to solve a linear system or the Cholesky factorization to solve a symmetric positive definite system, then as a by-product of the triangularization, we can obtain an upper bound of the condition number with just a little additional cost.

If QR factorization with column pivoting is used, then from

$$Q^T AP = R$$

we have $\text{Cond}_2(A) = \text{Cond}_2(R)$.

If the Cholesky factorization is used, then from

$$A = HH^T$$

we have $\text{Cond}_2(A) = (\text{Cond}_2(H))^2$. Thus, the $\text{Cond}_2(A)$ can be determined if $\text{Cond}_2(R)$ or $\text{Cond}_2(H)$ is known. Since $\|R\|_2$ or $\|H\|_2$ is easily computed, all that is needed is an algorithm to estimate $\|R^{-1}\|_2$ or $\|H^{-1}\|_2$. There are several algorithms for estimation of the norms of the inverses of triangular matrices. We just state one from a paper of Higham (1987). For details, see that paper.

ALGORITHM 6.7.2 Condition Estimation of an Upper Triangular Matrix Given a nonsingular upper triangular matrix $T = (t_{ij})$ of order n, the following algorithm computes C_E such that $\|T^{-1}\|_\infty \le C_E$.

Step 1: Set $z_n = \dfrac{1}{t_{nn}}$

Step 2: For $i = n - 1$ to 1 do

$s \equiv 1$

$s \equiv s + |t_{ij}|z_j \ (j = i + 1, \ldots, n)$

$z_i = \dfrac{s}{|t_{ii}|}$

Step 3: Compute $C_E = \|z\|_\infty$, where $z = (z_1, z_2, \ldots, z_n)^T$

Flop count. The algorithm requires $n^2/2$ flops.

Remark: Once $\|T^{-1}\|_\infty$ is estimated by Algorithm 6.7.2, $\|T^{-1}\|_2$ can be estimated from the relation

$$\|T^{-1}\|_2 \le (\|M(T)^{-1}\|_1 C_E)^{1/2}$$

where $M(T) = (m_{ij})$ are defined by

$$m_{ij} = \begin{cases} |t_{ii}|, & i = j \\ -|t_{ij}|, & i \ne j \end{cases}$$

$\|M(T)^{-1}\|_1$ can be estimated by using Hager's algorithm (Algorithm 6.7.1).

6.8 Componentwise Perturbations and the Errors

If the componentwise bounds of the perturbations are known, then the following perturbation result obtained by Skeel (1979) holds. In the following $\|\cdot\|$ stands for infinity-norm.

THEOREM 6.8.1 (Skeel 1979) Let $Ax = b$ and $(A + \Delta A)(x + \delta x) = b + \delta b$. Let $|\Delta A| \le \epsilon|A|$, $|\delta b| \le \epsilon|b|$. Then

$$\frac{\|\delta x\|}{\|x\|} \le \epsilon \frac{\| \, |A^{-1}||A||x| + |A^{-1}||b| \, \|}{(1 - \epsilon\| \, |A^{-1}||A| \, \|)\|x\|}$$

provided that the denominator is positive.

DEFINITION 6.8.1 We shall call the number

$$\text{Cond}(A, x) = \frac{\| \, |A^{-1}| \, |A| \, |x| \, \|}{\|x\|}$$

the **Skeel's condition number** and $\text{Cond}_s(A) = \||A^{-1}||A|\|$ the upper bound of the Skeel's condition number.

Definition 6.8.1 calls for some explanation. From Theorem 6.8.1, it can be shown (Skeel, 1979) that the expression

$$\frac{\| \, |A^{-1}||A||x| + |A^{-1}||b| \, \|}{\|x\|}$$

is the condition number of the system $Ax = b$, when the data A and b both are subject to perturbations, and $\text{Cond}(A, x)$ is the condition number when only A is subject to perturbation. However, Skeel (1979) has shown that $\text{Cond}(A, x)$ is adequate when both A and b are subject to perturbations.

In the literature, $\text{Cond}(A, x)$ is usually denoted by $\text{Cond}(A)$, and the usual condition number $\|A\|\|A^{-1}\|$ is denoted by $\kappa(A)$. The reader should make a note to this effect to avoid any confusion.

Thus in our notations:

$$\text{Cond}(A) = \|A\|\|A^{-1}\|$$

$$\text{Cond}(A, x) = \frac{\| \, |A^{-1}| \, |A| \, |x| \, \|}{\|x\|}$$

$$\text{Cond}_s(A) = \| \, |A||A^{-1}| \, \|$$

An important property of $\text{Cond}(A, x)$ is that Skeel's condition number is invariant under row scaling. It can, therefore, be much smaller than the usual condition number $\text{Cond}(A)$. $\text{Cond}(A, x)$ is useful when the column norms of A^{-1} vary widely.

Chandrasekaran and Ipsen (1994b) have recently given an analysis of how the individual components of the solution vector x are affected when the data are perturbed. This has led to the *componentwise condition numbers* that measure the sensitivity of each solution component to perturbations. Their analysis can effectively be combined with Skeel's result, when the componentwise perturbations of the data are known. We give an example.

THEOREM 6.8.2 Componentwise Perturbation Theorem Let $(A + \Delta A)\bar{x} = b$, and $|\Delta A| \le \epsilon|A|$. Then

$$\frac{|\bar{x}_i - x_i|}{|x_i|} \le \epsilon \frac{|r_i^T||A||\bar{x}|}{|x_i|}$$

where $r_i^T = e_i^T A^{-1}$.

Thus, the componentwise perturbations in the error expressions have led to the componentwise version of Skeel's condition number. Similar results also hold for right-hand-side perturbation. For details see Chandrasekaran and Ipsen (1994b).

6.9 Iterative Refinement

Suppose a computed solution $\hat{x}$ of the system $Ax = b$ is not acceptable. It is then natural to wonder if $\hat{x}$ can be refined cheaply by making use of the triangularization of the matrix A already at hand.

The following process, known as **iterative refinement**, can be used to refine $\hat{x}$ iteratively up to some desirable accuracy.

Iterative Refinement Algorithm

The process is based on the following simple idea: Let $\hat{x}$ be a computed solution of the system

$$Ax = b$$

If $\hat{x}$ were an exact solution, then

$$r = b - A\hat{x}$$

would be zero. But in practice we shall not expect that. Let us now try to solve the system again with the computed residual $r(\neq 0)$; that is, let c satisfy

$$Ac = r$$

Then, $y = \hat{x} + c$ is the exact solution of $Ax = b$ provided that c is the exact solution of $Ac = r$, because

$$Ay = A(\hat{x} + c) = A\hat{x} + Ac = b - r + r = b$$

It is true that c will not be an exact solution of $Ac = r$ in practice; however, the preceding discussion suggests that y might be a better approximation than $\hat{x}$. If so, we can continue the process until a desired accuracy is achieved.

ALGORITHM 6.9.1 Iterative Refinement

Set $x^{(0)} = \hat{x}$

For $k = 0, 1, 2, \ldots$ do

Step 1: Compute the residual vector $r^{(k)}$:

$$r^{(k)} = b - Ax^{(k)}$$

Step 2: Calculate the correction vector $c^{(k)}$ by solving the system

$$Ac^{(k)} = r^{(k)}$$

using the triangularization of A obtained to get the initial solution $\hat{x}$.

Step 3: Form $x^{(k+1)} = x^{(k)} + c^{(k)}$.

Step 4: If

$$\frac{\|c^{(k)}\|}{\|x^{(k+1)}\|}$$

is less than a prescribed tolerance ϵ, or if the number of iterations exceeds a predetermined number, Stop.

MATCOM Note: The algorithm has been implemented in MATCOM program ITERREF.

Example 6.9.1

$$A = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 2 & 1 \\ 0 & 0 & 3 \end{pmatrix}, \qquad b = \begin{pmatrix} 0.0001 \\ 0.0001 \\ -1.6666 \end{pmatrix}$$

The exact solution is

$$x = \begin{pmatrix} -0.2777 \\ 0.2778 \\ -0.5555 \end{pmatrix} \qquad \text{(correct up to four figures)}$$

$$x^{(0)} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

$k = 0$:

$$r^{(0)} = b - Ax^{(0)} = \begin{pmatrix} -1.9999 \\ -2.9999 \\ -4.6666 \end{pmatrix}$$

The solution of $Ac^{(0)} = r^{(0)}$ is

$$c^{(0)} = \begin{pmatrix} -1.2777 \\ -0.7222 \\ -1.5555 \end{pmatrix}$$

$$x^{(1)} = x^{(0)} + c^{(0)} = \begin{pmatrix} -0.2777 \\ 0.2778 \\ -0.5555 \end{pmatrix}$$

Note that $\text{Cond}(A) = 3.8078$. A is *well conditioned.*

Accuracy Obtained by Iterative Refinement

Suppose that the iteration converges. Then the error at $(k+1)$st step will be less than the error at the kth step.

Relative Accuracy from Iterative Refinement

Let

$$\frac{\|\hat{x} - x^{(k+1)}\|}{\|\hat{x}\|} \le c\frac{\|\hat{x} - x^{(k)}\|}{\|\hat{x}\|}$$

Then, if $c \approx 10^{-s}$, there will be a gain of approximately s figures per iteration.

Flop count. The procedure is quite cheap. Because A has already been triangularized to solve the original system $Ax = b$, each iteration requires only $O(n^2)$ flops.

Usefulnes of Iterative Refinement and Some Remarks

Iterative refinement is a very useful technique. Gaussian elimination with partial pivoting followed by iterative refinement is the most practical approach for solving a linear system accurately. Skeel (1980) has shown that "Just one iterative refinement with just single precision accumulation of the residuals is enough to make Gaussian elimination stable." This statement seems to be in contradiction with the usual belief that "it is absolutely essential that the residuals be computed with a higher precision than that of the rest of the computations" (Forsythe and Moler 1967, 49). However, Skeel (1980) has shown that the double precision residual accumulations (that is, computing the residuals in double precision and rounding to single precision) has "an important advantage for problems which are both poorly scaled and ill-conditioned."

As a matter of fact, it can be shown that if $\text{Cond}(A) \leq \frac{1}{\mu}$, and if the residuals are accumulated in double precision, then the iterative refinement procedure ultimately converges to the full precision solution of $Ax = b$. That is, if all the computations except the residuals are performed using t digit arithmetic, then the procedure will ultimately produce a solution that is accurate to t digits. Furthermore, Jankowski and Woźniakowski (1977) have proved that *any linear system solver, including Gaussian elimination, can be made stable by the use of iterative refinement.* See also a recent paper of Higham (1991), on iterative refinement of QR decomposition methods for solving linear systems.

Another important aspect of iterative refinement is that the slow convergence of the process is an indicator of the ill conditioning of the problem, as shown in the following result.

Estimating Cond(A) from Iterative Refinement

A very crude estimate of Cond(A) may be obtained from the iterative refinement procedure (Algorithm 6.9.1). Let k be the number of iterations required for the refinement procedure to converge and t be the number of digits used in the arithmetic; then (Rice 1981, 98):

$$\text{a rough estimate of Cond}(A) \text{ is } 10^{t(1-1/k)}$$

Thus, if the iterative refinement procedure converges very slowly, then A is ill conditioned.

6.10 Iterative Methods

In this section, we study **iterative methods**, used primarily to solve a very large and sparse linear system $Ax = b$ arising from engineering applications. These include the following:

1. The Jacobi method (Section 6.10.1)
2. The Gauss–Seidel method (Section 6.10.2)
3. The successive overrelaxation method (Section 6.10.4)

4. The conjugate gradient method with and without preconditioner (Section 6.10.5)
5. The GMRES method (Section 6.10.6)

The Gauss–Seidel method is a modification of the Jacobi method and is a special case of the successive overrelaxation method (SOR). The conjugate gradient method is used primarily to solve symmetric positive definite systems. The Jacobi and the Gauss–Seidel methods converge for diagonally dominant matrices; in addition, the Gauss–Seidel method converges if A is symmetric positive definite. Recall that the diagonally dominant and the symmetric positive definite matrices are among the most important classes of matrices arising in practical applications.

The direct methods based on triangularization of the matrix A become prohibitive in terms of computer time and storage if the matrix A is quite large. On the other hand, there are practical situations such as the *discretization of partial differential equations*, where the matrix size can be as large as several hundred thousand. For such problems, the direct methods become impractical. For example, if A is of order $10{,}000 \times 10{,}000$, it may take as long as 2 or 3 days for an IBM 370 to solve the system $Ax = b$ using Gaussian elimination or orthogonalization techniques of Householder and Givens. Furthermore, most large problems are sparse, and the *sparsity is lost to a considerable extent during the triangularization procedure*. Thus, at the end we have to deal with a very large matrix with too many nonzero entries, and storage becomes a crucial issue. For such problems, it is advisable to use a class of methods called *iterative methods* that never alter the matrix A and require the storage of only a few vectors of length n at a time.

Basic idea

The basic idea behind an iterative method is first to write the system $Ax = b$ in an equivalent form:

$$x = Bx + d \tag{6.10.1}$$

Then, starting with an initial approximation $x^{(1)}$ of the solution vector x, generate a sequence of approximations $\{x^{(k)}\}$ iteratively defined by

$$x^{(k+1)} = Bx^{(k)} + d, \qquad k = 1, 2, \ldots \tag{6.10.2}$$

with a hope that under certain mild conditions, the sequence $\{x^{(k)}\}$ converges to the solution as $k \to \infty$.

To solve the linear system $Ax = b$ iteratively using the idea, we therefore need to know

1. How to write the system $Ax = b$ in the form of (6.10.1), and
2. How $x^{(1)}$ should be chosen so that iteration (6.10.2) converges to the limit or under what sort of assumptions the iteration converges to the limit with any arbitrary choice of $x^{(1)}$.

Stopping Criteria for Iteration (6.10.2)

It is natural to wonder when iteration (6.10.2) can be terminated. When convergence occurs, $x^{(k+1)}$ is a better approximation than $x^{(k)}$, so a natural stopping criterion is as follows.

Stopping Criterion 1
Stop iteration (6.10.2) if

$$\frac{\|x^{(k+1)} - x^{(k)}\|}{\|x^{(k)}\|} < \epsilon$$

for a prescribed small positive number ϵ (ϵ should be chosen according to the accuracy desired) or if the number of iterations exceeds a predetermined number.

A stopping criterion in terms of the residual is given in the following:

Stopping Criterion 2
Stop iteration (6.10.2) if

$$\|r^{k+1}\| \le \epsilon(\|A\| \cdot \|x^{(k+1)}\| + \|b\|)$$

where

$$r^{(k+1)} = Ax^{(k+1)} - b$$

or if the number of iterations exceeds a predetermined number. ϵ is the tolerance to be chosen such that $\mu < \epsilon < 1$, μ is the machine precision.

Remark: Stopping Criterion 2 and some others appear in the book *Templates* (1994). For more on these criteria and their justifications, see *Templates* (1994, 53–54).

6.10.1 The Jacobi Method

The system

$$Ax = b$$

or

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n \end{aligned}$$

can be rewritten (under the assumption that $a_{ii} \neq 0$, $i = 1, \ldots, n$) as

$$x_1 = \frac{1}{a_{11}}(b_1 - a_{12}x_2 - \cdots - a_{1n}x_n)$$

$$x_2 = \frac{1}{a_{22}}(b_2 - a_{21}x_1 - a_{23}x_3 - \cdots - a_{2n}x_n)$$

$$\vdots$$

$$x_n = \frac{1}{a_{nn}}(b_n - a_{n1}x_1 - \cdots - a_{n,n-1}x_{n-1})$$

In matrix notation,

$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} 0 & -\frac{a_{12}}{a_{11}} & \cdots & \cdots & -\frac{a_{1n}}{a_{11}} \\ -\frac{a_{21}}{a_{22}} & 0 & -\frac{a_{23}}{a_{22}} & \cdots & -\frac{a_{2n}}{a_{22}} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ & & \ddots & \ddots & -\frac{a_{n-1,n}}{a_{n-1,n-1}} \\ -\frac{a_{n1}}{a_{nn}} & \cdots & \cdots & -\frac{a_{n,n-1}}{a_{nn}} & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} + \begin{pmatrix} \frac{b_1}{a_{11}} \\ \frac{b_2}{a_{22}} \\ \vdots \\ \frac{b_n}{a_{nn}} \end{pmatrix}$$

or

$$x = Bx + d$$

If we write the matrix A in the form

$$A = L + D + U$$

where

$$L = \begin{pmatrix} 0 & 0 & 0 & \cdots & 0 \\ a_{21} & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ a_{n1} & \cdots & \cdots & a_{n,n-1} & 0 \end{pmatrix}$$

$$D = \operatorname{diag}(a_{11}, \ldots, a_{nn}),$$

and

$$U = \begin{pmatrix} 0 & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & 0 & a_{23} & \cdots & a_{2n} \\ \vdots & & \ddots & \ddots & \vdots \\ \vdots & & & \ddots & a_{n-1,n} \\ 0 & \cdots & \cdots & \cdots & 0 \end{pmatrix}$$

then it is easy to see that

$$B = -D^{-1}(L + U) = (I - D^{-1}A)$$
$$d = D^{-1}b$$

(Note that, because of our assumption that $a_{ii} \neq 0$, $i = 1, \ldots, n$, D is nonsingular.)

We shall call the matrix

$$B = -D^{-1}(L + U) = (I - D^{-1}A)$$

the **Jacobi iteration matrix** and denote it by B_J. Similarly, we shall denote the vector $D^{-1}b$ by b_J, which we call the **Jacobi vector**.

The Jacobi Iteration Matrix and the Jacobi Vector

Let $A = L + D + U$. Then

$$B_J = -D^{-1}(L + U)$$
$$b_J = D^{-1}b$$

With the Jacobi iteration matrix and the Jacobi vector as defined, iteration (6.10.2) becomes the following.

The Jacobi Iteration

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{\substack{j=1\\ j\neq i}}^{n} a_{ij}x_j^{(k)}\right), \qquad i = 1, \ldots, n$$

We thus have the following iterative procedure, called the **Jacobi method**.

ALGORITHM 6.10.1 The Jacobi Method

Step 1: Choose $x^{(1)} = \begin{pmatrix} x_1^{(1)} \\ x_2^{(1)} \\ \vdots \\ x_n^{(1)} \end{pmatrix}$.

Step 2: For $k = 1, 2, \ldots,$ do until a stopping criterion is satisfied

$$x^{(k+1)} = B_J x^{(k)} + b_J \tag{6.10.3}$$

or

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{\substack{j=1\\ j\neq i}}^{n} a_{ij}x_j^{(k)}\right), \qquad i = 1, \ldots, n \tag{6.10.4}$$

Note: In practical implementation, we will use Equation (6.10.4).

MATCOM Note: Algorithm 6.10.1 has been implemented in MATCOM program JACOBI.

Example 6.10.1

$$A = \begin{pmatrix} 5 & 1 & 1 \\ 1 & 5 & 1 \\ 1 & 1 & 5 \end{pmatrix}, \qquad b = \begin{pmatrix} 7 \\ 7 \\ 7 \end{pmatrix}$$

$$x^{(1)} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

$$B_J = \begin{pmatrix} 0 & -0.2000 & -0.2000 \\ -0.2000 & 0 & -0.2000 \\ -0.2000 & -0.2000 & 0 \end{pmatrix}, \qquad b_J = \begin{pmatrix} 1.4000 \\ 1.4000 \\ 1.4000 \end{pmatrix}$$

$k = 1$:

$$x^{(2)} = B_J x^{(1)} + b_J = \begin{pmatrix} 1.4000 \\ 1.4000 \\ 1.4000 \end{pmatrix}$$

$k = 2$:

$$x^{(3)} = B_J x^{(2)} + b_J = \begin{pmatrix} 0.8400 \\ 0.8400 \\ 0.8400 \end{pmatrix}$$

$k = 3$:

$$x^{(4)} = B_J x^{(3)} + b_J = \begin{pmatrix} 1.0640 \\ 1.0640 \\ 1.0640 \end{pmatrix}$$

$k = 4$:

$$x^{(5)} = B_J x^{(4)} + b_J = \begin{pmatrix} 0.9744 \\ 0.9744 \\ 0.9744 \end{pmatrix}$$

$k = 5$:

$$x^{(6)} = B_J x^{(5)} + b_J = \begin{pmatrix} 1.0102 \\ 1.0102 \\ 1.0102 \end{pmatrix}$$

$k = 6$:

$$x^{(7)} = B_J x^{(6)} + b_J = \begin{pmatrix} .9959 \\ .9959 \\ .9959 \end{pmatrix}$$

$k = 7$:

$$x^{(8)} = B_J x^{(7)} + b_J = \begin{pmatrix} 1.0016 \\ 1.0016 \\ 1.0016 \end{pmatrix}$$

Jacobi with SOR:

$$x_i^{(k+1)} = \frac{w}{a_{ii}}\left(b_i - \sum_{j\neq i}^{n} a_{ij}x_j^{(k)}\right) + (1-w)x_i^{(k)}$$

In matrix formulation:

$$B_J = I - wD^{-1}A$$
$$d = wD^{-1}b$$

6.10.2 The Gauss–Seidel Method

In the Jacobi method, to compute the components of the vector $x^{(k+1)}$, only the components of the vector $x^{(k)}$ are used; however, note that to compute $x_i^{(k+1)}$, we

could have used $x_1^{(k+1)}$ through $x_{i-1}^{(k+1)}$, which were already available to us. Thus, a natural modification of the Jacobi method will be to rewrite the Jacobi iteration (6.10.4) in the following form.

The Gauss–Seidel Iteration

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\right), i = 1, 2, \ldots, n \qquad (6.10.5)$$

The idea is to use each new component, as soon as it is available, in the computation of the next component.

The iteration (6.10.5) is known as the **Gauss–Seidel iteration**, and the iterative method based on this iteration is called the **Gauss–Seidel Method.***

In the notation used earlier, the Gauss–Seidel iteration is:

$$x^{(k+1)} = -(D+L)^{-1}Ux^{(k)} + (D+L)^{-1}b$$

(Note that the matrix $D + L$ is a lower triangular matrix with $a_{11}, \ldots, a_{nn}$ on the diagonal; because we have assumed that these entries are nonzero, the matrix $(D + L)$ is nonsingular).

We call the matrix

$$B = -(D+L)^{-1}U$$

the **Gauss–Seidel matrix** and denote it by the symbol B_{GS}. Similarly, the Gauss–Seidel vector $(D + L)^{-1}b$ is denoted by b_{GS}.

The Gauss–Seidel Matrix and the Gauss–Seidel Vector

Let $A = L + D + U$. Then

$$B_{GS} = -(D+L)^{-1}U$$
$$b_{GS} = (D+L)^{-1}b$$

ALGORITHM 6.10.2 The Gauss–Seidel Method

Step 1: Choose an initial approximation $x^{(1)}$.

Step 2: For $k = 1, 2, \ldots,$ do until a stopping criterion is satisfied

$$x^{(k+1)} = B_{GS}x^{(k)} + b_{GS} \qquad (6.10.6)$$

*Association of Seidel's name with Gauss' for this method does not seem to be well documented in history.

or

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\right),$$
$$i = 1, 2, \ldots, n \qquad (6.10.7)$$

Note: In practice, we will use Equation (6.10.7).

MATCOM Note: Algorithm 6.10.2 has been implemented in MATCOM program GAUSED.

Example 6.10.2

$$A = \begin{pmatrix} 5 & 1 & 1 \\ 1 & 5 & 1 \\ 1 & 1 & 5 \end{pmatrix}, \qquad b = \begin{pmatrix} 7 \\ 7 \\ 7 \end{pmatrix}$$

$$B_{GS} = \begin{pmatrix} 0 & -0.2 & -0.2 \\ 0 & 0.04 & -0.16 \\ 0 & 0.032 & 0.072 \end{pmatrix}$$

$$b_{GS} = \begin{pmatrix} 1.4000 \\ 1.1200 \\ 0.8960 \end{pmatrix}$$

$k = 1$:

$$x^{(2)} = B_{GS}x^{(1)} + b_{GS} = \begin{pmatrix} 1.4000 \\ 1.1200 \\ 0.8960 \end{pmatrix}$$

$k = 2$:

$$x^{(3)} = B_{GS}x^{(2)} + b_{GS} = \begin{pmatrix} 0.9968 \\ 1.0214 \\ 0.9964 \end{pmatrix}$$

$k = 3$:

$$x^{(4)} = B_{GS}x^{(3)} + b_{GS} = \begin{pmatrix} 0.9964 \\ 1.0014 \\ 1.0004 \end{pmatrix}$$

$k = 4$:

$$x^{(5)} = B_{GS}x^{(4)} + b_{GS} = \begin{pmatrix} 0.9996 \\ 1.0000 \\ 1.0001 \end{pmatrix}$$

$k = 5$:

$$x^{(6)} = B_{GS}x^{(5)} + b_{GS} = \begin{pmatrix} 1.0000 \\ 1.0000 \\ 1.0000 \end{pmatrix}$$

Computer Implementations

On actual computer implementations, it is certainly economical to use equations (6.10.4) and (6.10.7). The use of (6.10.3) and (6.10.6) necessitates the storage of D, L, and U, which is a waste of storage.

6.10.3 Convergence of Iterative Methods

It is often hard to make a good guess of the initial approximation $x^{(1)}$. Thus, it will be nice to have conditions that will guarantee the convergence of iteration (6.10.2) for any arbitrary choice of the initial approximation.

In the following we derive such a condition.

THEOREM 6.10.1 Iteration Convergence Theorem The iteration

$$x^{(k+1)} = Bx^{(k)} + c$$

converges to a limit with an arbitrary choice of the initial approximation $x^{(1)}$ if and only if the matrix $B^k \to 0$ as $k \to \infty$; that is, B is a convergent matrix.

Proof: From

$$x = Bx + c$$

and

$$x^{(k+1)} = Bx^{(k)} + c$$

we have

$$x - x^{(k+1)} = B(x - x^{(k)}) \tag{6.10.8}$$

Because this is true for any value of k, we can write

$$x - x^{(k)} = B(x - x^{(k-1)}) \tag{6.10.9}$$

Substituting (6.10.9) in (6.10.8), we have

$$x - x^{(k+1)} = B^2(x - x^{(k-1)}) \tag{6.10.10}$$

Continuing this process k times, we can write

$$x - x^{(k+1)} = B^k(x - x^{(1)})$$

This shows that $\{x^{(k)}\}$ converges to the solution x for any arbitrary choice of $x^{(1)}$ if and only if $B^k \to 0$ as $k \to \infty$ (unless, of course, the initial guess $x^{(1)}$ is the same as the exact solution). ■

Recall now from Chapter 1 (Theorem 1.7.4) that B is a *convergent matrix if and only if the spectral radius of* B, $\rho(B)$, *is less than* 1. Now $\rho(B) = \max\{|\lambda_i|,\ i = 1, \ldots, n\}$, where λ_1 through λ_n are the eigenvalues of B, because $|\lambda_i| \le \|B\|$ for each i and for any subordinate matrix norm; in particular, $\rho(B) \le \|B\|$. Thus, a good way to see if B is convergent is to compute $\|B\|$ with a row-sum or column-sum norm and see if this is less than 1. (Note that the converse is not true.)

We combine the result of Theorem 6.10.1 with the observation just made in the following.

Conditions for Convergence of Iteration (6.10.2)
A necessary and sufficient condition for the convergence of the iteration (6.10.2), for any arbitrary choice of $x^{(1)}$, is that $\rho(B) < 1$. A sufficient condition is that $\|B\| < 1$, for some norm.

We now apply this result to identify classes of matrices for which the Jacobi and/or Gauss–Seidel methods converge for any choice of initial approximation $x^{(1)}$.

The Jacobi and Gauss–Seidel Methods for Diagonally Dominant Matrices

COROLLARY 6.10.1 If A is row diagonally dominant, then the Jacobi method converges for any arbitrary choice of the initial approximation $x^{(1)}$.

Proof: Because $A = (a_{ij})$ is row diagonally dominant, we have, by definition

$$|a_{ii}| > \sum_{\substack{j=1 \\ i \ne j}}^{n} |a_{ij}|, \qquad i = 1, 2, \ldots, n \tag{6.10.11}$$

Recall that the Jacobi iteration matrix B_J is given by

$$B_J = \begin{pmatrix} 0 & -\frac{a_{12}}{a_{11}} & \cdots & \cdots & -\frac{a_{1n}}{a_{11}} \\ -\frac{a_{21}}{a_{22}} & 0 & -\frac{a_{23}}{a_{22}} & \cdots & -\frac{a_{2n}}{a_{22}} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ & & \ddots & \ddots & -\frac{a_{n-1,n}}{a_{n-1,n-1}} \\ -\frac{a_{n1}}{a_{nn}} & \cdots & \cdots & -\frac{a_{n,n-1}}{a_{nn}} & 0 \end{pmatrix}$$

From (6.10.11) we therefore have that the absolute row sum (that is, the row sum taking absolute values) of each row is less than 1, which means

$$\|B_J\|_\infty < 1$$

Thus, by Theorem 6.10.1, we have Corollary 6.10.1. ■

COROLLARY 6.10.2 If A is a row diagonally dominant matrix, then the Gauss–Seidel method converges for any arbitrary choice of $x^{(1)}$.

Proof: The Gauss–Seidel iteration matrix is given by

$$B_{GS} = -(D + L)^{-1}U$$

Let λ be an eigenvalue of this matrix and $x = (x_1, \ldots, x_n)^T$ be the corresponding eigenvector. Then from

$$B_{GS}\, x = \lambda x$$

or

$$-Ux = (D + L)\lambda x$$

we have

$$-\sum_{j=i+1}^{n} a_{ij}x_j = \lambda \sum_{j=1}^{i} a_{ij}x_j, \qquad 1 \le i \le n$$

which can be rewritten as

$$\lambda a_{ii}x_i = -\lambda \sum_{j=1}^{i-1} a_{ij}x_j - \sum_{j=i+1}^{n} a_{ij}x_j, \qquad 1 \le i \le n$$

Let x_k be the largest component (having the magnitude 1) of the vector x. Then, from the preceding equation, we have

$$|\lambda||a_{kk}| \le |\lambda| \sum_{j=1}^{k-1} |a_{kj}| + \sum_{j=k+1}^{n} |a_{kj}| \tag{6.10.12}$$

that is,

$$|\lambda| \left(|a_{kk}| - \sum_{j=1}^{k-1} |a_{kj}| \right) \le \sum_{j=k+1}^{n} |a_{kj}| \tag{6.10.13}$$

or

$$|\lambda| \le \frac{\sum_{j=k+1}^{n} |a_{kj}|}{(|a_{kk}| - \sum_{j=1}^{k-1} |a_{kj}|)} \tag{6.10.14}$$

Since A is row diagonally dominant,

$$|a_{kk}| > \sum_{\substack{j=1 \\ j \ne k}}^{n} |a_{kj}|$$

or

$$|a_{kk}| - \sum_{j=1}^{k-1} |a_{kj}| > \sum_{j=k+1}^{n} |a_{kj}|$$

Thus from (6.10.14), we conclude that $|\lambda| < 1$; that is,

$$\rho(B_{GS}) < 1$$

Thus, from Theorem 6.10.1, we have Corollary 6.10.2. ■

A remark on Corollary 6.10.1. It is usually true that the greater the diagonal dominance of A, the faster is the convergence of the Jacobi method. However, there are simple counterexamples that show that this does not always happen.

The following simple 2×2 example in support of this statement appears in Golub and Van Loan (1989, 514). The example was supplied by Richard S. Varga.

$$A_1 = \begin{pmatrix} 1 & -\frac{1}{2} \\ -\frac{1}{2} & 1 \end{pmatrix}, \qquad A_2 = \begin{pmatrix} 1 & -\frac{3}{4} \\ -\frac{1}{2} & 1 \end{pmatrix}$$

It is easy to verify that $\rho(B_J)$ of A_1 is greater than $\rho(B_J)$ of A_2.

We now discuss the convergence of the Gauss–Seidel method for yet another very important class of matrices, namely, the symmetric positive definite matrices.

The Gauss–Seidel Method for A Symmetric Positive Definite Matrix

We show that the Gauss–Seidel method converges, with an arbitrary choice of $x^{(1)}$, for a symmetric positive definite matrix.

THEOREM 6.10.2 Let A be a symmetric positive definite matrix. Then the Gauss–Seidel method converges for any arbitrary choice of the initial approximation $x^{(1)}$.

Proof: Because A is symmetric, we have

$$A = L + D + L^T$$

Thus, $B_{GS} = -(D + L)^{-1}L^T$. We now show that $\rho(B_{GS}) < 1$.

Let $-\lambda$ be an eigenvalue of B_{GS} and u be the corresponding eigenvector. Then

$$(D + L)^{-1}L^T u = \lambda u$$

or

$$L^T u = \lambda(D + L)u$$

Thus,

$$u^* L^T u = \lambda u^*(D + L)u \quad (u^* = (\bar{u})^T)$$

or

$$u^* Au - u^*(L + D)u = \lambda u^*(L + D)u$$

(Note that $L^T = A - (L + D)$) or

$$u^* Au = (1 + \lambda)u^*(L + D)u \tag{6.10.15}$$

Taking conjugate transpose on both sides, we have

$$u^* Au = (1 + \bar{\lambda})u^*(L^T + D^T)u \tag{6.10.16}$$

From (6.10.15) and (6.10.16), we obtain

$$\left(\frac{1}{(1 + \lambda)} + \frac{1}{(1 + \bar{\lambda})} \right) u^* Au = u^*(L + D)u + u^*(L^T + D^T)u \tag{6.10.17}$$

$$= u^*(L + D + L^T + D^T)u \tag{6.10.18}$$

$$= u^*(A + D^T)u \tag{6.10.19}$$

$$= u^*(A + D)u > u^* Au \tag{6.10.20}$$

(Note that since A is positive definite, so is D; therefore, $u^* Du > 0$.)

Dividing both sides of (6.10.20) by $u^* Au (> 0)$, we have

$$\frac{1}{1 + \lambda} + \frac{1}{1 + \bar{\lambda}} > 1$$

or

$$\frac{2+\lambda+\bar{\lambda}}{(1+\lambda)(1+\bar{\lambda})} > 1 \tag{6.10.21}$$

Let $\lambda = \alpha + i\beta$. Then $\bar{\lambda} = \alpha - i\beta$. From (6.10.21) we then have

$$\frac{2(1+\alpha)}{(1+\alpha)^2+\beta^2} > 1$$

from which it follows that $\alpha^2 + \beta^2 < 1$. That is, $\rho(B_{GS}) < 1$, because $|\lambda| = \sqrt{\alpha^2+\beta^2}$. ■

Rates of Convergence and a Comparison Between the Gauss–Seidel and the Jacobi Methods

We have seen that for row diagonally dominant matrices both the Jacobi and the Gauss–Seidel methods converge for an arbitrary $x^{(1)}$. The question naturally arises whether this is true for some other matrices as well. Also, when both methods converge, another question arises: Which one converges faster?

A consequence of the well-known Stein–Rosenberg Theorem (for a statement and proof of this theorem, see Varga (1962)), is that if A is a matrix having diagonal entries positive and off-diagonal entries negative, then:

1. The Jacobi and the Gauss–Seidel either both converge or diverge.
2. When both the methods converge, the Gauss–Seidel method converges faster than the Jacobi method.

Note that the asymptotic rate of convergence is defined as follows.

DEFINITION 6.10.1 The quantity $-\ln\rho(B)$ is called the asymptotic rate of convergence of the iteration (6.10.2).

Remarks: Unfortunately, in the general case no such statements about the convergence and the asymptotic rates of convergence of two iterative methods can be made. In fact, there are examples where one method converges but the other diverges (see the following example). However, when both the Gauss–Seidel and the Jacobi converge, because of the lower storage requirement and the asymptotic rates of convergence, the Gauss–Seidel method should be preferred over the Jacobi.

Example 6.10.3 The following example shows that the Jacobi method can converge even when the Gauss–Seidel method does not.

$$A = \begin{pmatrix} 1 & 2 & -2 \\ 1 & 1 & 1 \\ 2 & 2 & 1 \end{pmatrix}$$

$$b = \begin{pmatrix} 1 \\ 2 \\ 5 \end{pmatrix}$$

$$B_J = \begin{pmatrix} 0 & -2 & 2 \\ -1 & 0 & -1 \\ -2 & -2 & 0 \end{pmatrix}$$

$$B_{GS} = \begin{pmatrix} 0 & -2 & 2 \\ 0 & 2 & -3 \\ 0 & 0 & 2 \end{pmatrix}$$

$$\rho(B_{GS}) = 2$$

$$\rho(B_J) = 6.7815 \times 10^{-6}$$

A Few Iterations with Gauss–Seidel

The following iterations show that the Gauss–Seidel method is clearly diverging.

$$x^{(1)} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \quad x^{(2)} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \quad x^{(3)} = \begin{pmatrix} 1 \\ 0 \\ 3 \end{pmatrix}$$

$$x^{(4)} = \begin{pmatrix} 7 \\ -8 \\ 7 \end{pmatrix}, \quad x^{(5)} = \begin{pmatrix} 31 \\ -36 \\ 15 \end{pmatrix}$$

The exact solution is

$$x = \begin{pmatrix} 7 \\ -4 \\ -1 \end{pmatrix}$$

On the other hand, with the Jacobi method we have convergence with only two iterations:

$$x^{(1)} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

$$x^{(2)} = \begin{pmatrix} 1 \\ 2 \\ 5 \end{pmatrix}$$

$$x^{(3)} = \begin{pmatrix} 7 \\ -4 \\ -1 \end{pmatrix}$$

6.10.4 The Successive Overrelaxation (SOR) Method

The Gauss–Seidel method is frustratingly slow when $\rho(B_{GS})$ is close to unity. However, the rate of convergence of the Gauss–Seidel iteration can, in certain cases, be improved by introducing a parameter ω, known as the relaxation parameter. The following modified Gauss–Seidel iteration is known as the successive overrelaxation iteration or, in short, the SOR iteration, if $\omega > 1$.

The SOR Iteration

$$x_i^{(k+1)} = \frac{\omega}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\right) + (1-\omega)x_i^k,$$
$$i = 1, 2, \ldots \qquad (6.10.22)$$

From (6.10.22) we note the following:

1. When $\omega = 1$, the SOR iteration reduces to the Gauss–Seidel iteration.
2. When $\omega > 1$, in computing the $(k+1)$ approximation, more weight is placed on the most current value than when $\omega < 1$, with a hope that the convergence will be faster.

In matrix notation the SOR iteration is

$$x^{(k+1)} = (D + \omega L)^{-1}[(1-\omega)D - \omega U]x^{(k)} + \omega(D + \omega L)^{-1}b,$$
$$k = 1, 2, \ldots \qquad (6.10.23)$$

(Note that because $a_{ii} \neq 0$, $i = 1, \ldots, n$, the matrix $(D + \omega L)$ is nonsingular.)

The matrix $(D + \omega L)^{-1}[(1-\omega)D - \omega U]$ is called the SOR matrix and is denoted by B_{SOR}. Similarly, the vector $(D + \omega L)^{-1}b$ is denoted by b_{SOR}.

The SOR Matrix and the SOR Vector

$$B_{SOR} = (D + \omega L)^{-1}[(1-\omega)D - \omega U]$$
$$b_{SOR} = \omega(D + \omega L)^{-1}b$$

In matrix notation the SOR iteration algorithm is as follows.

ALGORITHM 6.10.3 The Successive Overrelaxation Method

Step 1: Choose $x^{(1)}$.

Step 2: For $k = 1, 2, \ldots$ do until a stopping criterion is satisfied

$$x^{(k+1)} = B_{SOR}x^{(k)} + b_{SOR}$$

or compute $x^{(k+1)}$ using Equation 6.10.22.

MATCOM Note: Algorithm 6.10.3 has been implemented in MATCOM program SUCOV.

Example 6.10.4 Let A and b be the same as in Example 6.10.2. Let

$$x_1^{(1)} = x_2^{(1)} = x_3^{(1)} = 0, \qquad \omega = 1.2$$

Then

$$B_{SOR} = \begin{pmatrix} -0.2000 & -0.2400 & -0.2400 \\ 0.0480 & -0.1424 & -0.1824 \\ 0.0365 & 0.0918 & -0.0986 \end{pmatrix}, \qquad b_{SOR} = \begin{pmatrix} 1.6800 \\ 1.2768 \\ 0.9704 \end{pmatrix}$$

$k = 1$:

$$x^{(2)} = \begin{pmatrix} x_1^{(2)} \\ x_2^{(2)} \\ x_3^{(2)} \end{pmatrix} = B_{SOR}x^{(1)} + b_{SOR} = \begin{pmatrix} 1.6800 \\ 1.2768 \\ 0.9704 \end{pmatrix}$$

$k = 2$:

$$x^{(3)} = \begin{pmatrix} x_1^{(3)} \\ x_2^{(3)} \\ x_3^{(3)} \end{pmatrix} = B_{SOR}x^{(2)} + b_{SOR} = \begin{pmatrix} 0.8047 \\ 0.9986 \\ 1.0531 \end{pmatrix}$$

$k = 3$:

$$x^{(4)} = \begin{pmatrix} x_1^{(4)} \\ x_2^{(4)} \\ x_3^{(4)} \end{pmatrix} = B_{SOR}x^{(3)} + b_{SOR} = \begin{pmatrix} 1.0266 \\ 0.9811 \\ 0.9875 \end{pmatrix}$$

Choice of ω in the Convergent SOR Iteration

It is natural to wonder about the range of ω for which the SOR iteration converges and the optimal choice of ω. To this end, we first prove the following important result due to William Kahan.

THEOREM 6.10.3 Kahan The SOR iteration cannot converge for an arbitrary initial approximation if ω lies outside the interval $(0, 2)$.

Proof: Recall that the SOR iteration matrix B_{SOR} is given by

$$B_{SOR} = (D + \omega L)^{-1}[(1-\omega)D - \omega U]$$

where $A = L + D + U$.

The matrix $(D + \omega L)^{-1}$ is a lower triangular matrix with $1/a_{ii}$, $i = 1, \ldots, n$, as the diagonal entries, and the matrix $(1-\omega)D - \omega U$ is an upper triangular matrix with $(1-\omega)a_{ii}$, $i = 1, \ldots, n$, as the diagonal entries. So

$$\det(B_{SOR}) = (1-\omega)^n$$

Because the determinant of a matrix is equal to the product of its eigenvalues, we conclude that

$$\rho(B_{SOR}) \geq |1 - \omega|$$

where $\rho(B_{SOR})$ is the special radius of the matrix B_{SOR}.

Since by Theorem 6.10.1, for the convergence of any iterative method the spectral radius of the iteration matrix has to be less than 1, we conclude that ω must lie in the interval $(0, 2)$. ■

The next theorem, known as Ostrowski–Reich theorem, shows that the preceding condition is also sufficient in case the matrix A is symmetric and positive definite.

The theorem was proved by Reich for the Gauss–Seidel iteration ($\omega = 1$) in 1949 and subsequently extended by Ostrowski in 1954.

We state the theorem without proof. For proof, see Varga (1962, Section 3.4) or Ortega (1990, 123). The Ostrowski–Reich theorem is a generalization of Theorem 6.10.3 for symmetric positive definite matrices.

THEOREM 6.10.4 Ostrowski–Reich Let A be a symmetric positive definite matrix and let $0 < \omega < 2$. Then the SOR method will converge for any arbitrary choice of $x^{(1)}$.

Optimal Choice of ω in the Convergent SOR Iteration

We now turn to the question of optimal choice of ω. Again, for an arbitrary matrix, no criterion has been developed so far. However, such a criterion is known for a very useful class of matrices that arise in many practical applications. The matrices in this class are called consistently ordered matrices.

DEFINITION 6.10.2 A matrix $A = L + D + U$ is **consistently ordered** if the eigenvalues of the matrix

$$C(\mu) = -D^{-1}\left(\frac{1}{\mu}L + \mu U\right)$$

do not depend on μ, $\mu \neq 0$.

Young has defined a consistently ordered matrix differently (Young 1971). The latter does not depend on the eigenvalues.

DEFINITION 6.10.3 The matrix A is **2-cyclic** if there is a permutation matrix P such that

$$PAP^T = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$$

where A_{11} and A_{22} are diagonal.

David Young has defined such a matrix as the matrix having *Property A*. This definition can be generalized to block matrices where the diagonal matrices A_{11} and A_{22} are block diagonal matrices; we call such matrices **block 2-cyclic matrices**. A well-known example of a consistently ordered block 2-cyclic matrix is the block tridiagonal matrix

$$A = \begin{pmatrix} T & -I & & 0 \\ -I & \ddots & \ddots & \\ & \ddots & \ddots & -I \\ 0 & & -I & T \end{pmatrix}$$

where

$$T = \begin{pmatrix} 4 & -1 & \cdots & 0 \\ -1 & \ddots & \ddots & \vdots \\ \ddots & \ddots & \ddots & -1 \\ 0 & \cdots & -1 & 4 \end{pmatrix}$$

Recall that this matrix arises in the discretization of the Poisson's equations:

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = f$$

on the unit square. In fact, it can be shown (Exercise 39b) that *every block tridiagonal matrix with nonsingular diagonal blocks is consistently ordered and 2-cyclic.*

The following important and well-known theorem on the optimal choice of ω for consistently ordered matrices is due to David Young (1971).

THEOREM 6.10.5 Young Let A be consistently ordered and 2-cyclic with nonzero diagonal elements. Then

$$\rho(B_{GS}) = (\rho(B_J))^2$$

Furthermore, if the eigenvalues of B_J are real and $\rho(B_J) < 1$, then the optimal choice for ω in terms of producing the smallest spectral radius in SOR, denoted by ω_{opt}, is given by

$$\omega_{opt} = \frac{2}{1 + \sqrt{1 - \rho(B_J)^2}}$$

and $\rho(B_{SOR}) = \omega_{opt} - 1$.

As an immediate consequence, we get the following corollary.

COROLLARY 6.10.3 For consistently ordered 2-cyclic matrices, if the Jacobi method converges, so does the Gauss–Seidel method, and the Gauss–Seidel method converges twice as fast as the Jacobi method.

Example 6.10.5

$$A = \begin{pmatrix} 4 & -1 & 0 & -1 & 0 & 0 \\ -1 & 4 & -1 & 0 & -1 & 0 \\ 0 & -1 & 4 & 0 & 0 & -1 \\ -1 & 0 & 0 & 4 & -1 & 0 \\ 0 & -1 & 0 & -1 & 4 & -1 \\ 0 & 0 & -1 & 0 & -1 & 4 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

The eigenvalues of B_J are 0.1036, 0.2500, −0.1036, −0.2500, 0.6036, and −0.6036.

$$\rho(B_J) = 0.6036$$

$$\omega_{opt} = \frac{2}{1 + \sqrt{1 - (0.6036)^2}} = 1.1128$$

$$\rho(B_{GS}) = 0.3643$$

It took five iterations for the SOR method with ω_{opt} to converge to the exact solution (up to four significant figures), starting with

$$x_{SOR}^{(1)} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

$$x_{SOR}^{(5)} = \begin{pmatrix} 0.2948 \\ 0.0932 \\ 0.0282 \\ 0.0861 \\ 0.0497 \\ 0.0195 \end{pmatrix}$$

With the same starting vector $x^{(1)}$, the Gauss–Seidel method required 9 iterations. (Try it!) Also find out that 18 iterations will be required by Jacobi.

Comparison of Rates of Convergence Between the Gauss–Seidel and the SOR Methods

The following theorem, also due to D. Young (1971), relates the rates of convergence between the Gauss–Seidel method and the SOR method with the optimum relaxation factor, ω_{opt}, for consistently ordered matrices.

THEOREM 6.10.6 Young Let A be a consistently ordered matrix with nonzero diagonal elements. Assume that the eigenvalues of B_J are all real and $\rho(B_J) < 1$. Let R_{GS} and R_{SOR} denote, respectively, the asymptotic rates of convergence of the Gauss–Seidel and the SOR methods with optimum relaxation factor ω_{opt}. Then

$$2\rho(B_J)R_{GS}^{1/2} \leq R_{SOR} \leq R_{GS} + 2[R_{GS}]^{1/2}$$

The second inequality holds if $R_{GS} \leq 3$.

Remark: Theorem 6.10.6 basically states that the SOR method with the optimum relaxation factor converges much faster (which is expected) than the Gauss–Seidel method when the asymptotic rate of convergence of the Gauss–Seidel method is small.

Example 6.10.6 Consider again the matrix arising in the discretization of Poisson's equation with the mesh size h (matrix (6.3.31)).

For this matrix it is easily verified that

$$\rho(B_J) = \cos(\pi h)$$

We also know that $\rho(B_{GS}) = (\rho(B_J))^2$. Thus,

$$R_{GS} = 2(-\ln\cos\pi h)$$

$$= 2\frac{\pi^2 h^2}{2} + O(h^4)$$

$$= \pi^2 h^2 + O(h^4)$$

For small h, R_{GS} is small, and by the second inequality of Theorem 6.10.6 we have

$$R_{SOR} \approx 2R_{GS}^{1/2} \equiv 2\pi h$$

and

$$\frac{R_{SOR}}{R_{GS}} \approx \frac{2}{(\pi h)}$$

Thus, when h is small, the asymptotic rate of convergence of the SOR method with the optimum relaxation is much greater than the Gauss–Seidel method. For example, when $h = \frac{1}{50}$, then

$$\frac{R_{SOR}}{R_{GS}} \equiv \frac{100}{\pi} = 31.8$$

Thus, in this case the SOR method converges about 31 times faster than the Gauss–Seidel method. Furthermore, the rate of convergence becomes greater as h decreases; the improvement is really remarkable when h is very very small.

6.10.5 The Conjugate Gradient Method

We describe here a method known as the **conjugate gradient (CG) method** for solving a symmetric positive definite linear system

$$Ax = b$$

The method was originally devised by Hestenes and Stiefel (1952) and now is widely used to solve *large and sparse* symmetric positive definite systems. *It is direct in theory, but iterative in practice*. The method is based on the following well-known result on optimization (Luenberger 1973).

THEOREM 6.10.7 If A is a real symmetric positive definite matrix, then solving

$$Ax = b$$

is equivalent to minimizing the quadratic function

$$\phi(x) = \frac{1}{2}x^T Ax - x^T b$$

Furthermore, the minimum value of $\phi(x)$ is $-\frac{1}{2}b^T A^{-1} b$ and is obtained by choosing

$$x = A^{-1}b$$

There are a large number of iterative methods in the literature of optimization for solving this minimization problem (Luenberger 1973). In these iterative methods the successive approximations x_k to the solution vector x are computed recursively:

$$x_{k+1} = x_k + \alpha_k p_k$$

where the vectors $\{p_k\}$ are called the **direction vectors** and the scalars α_k are chosen to minimize $\phi(x)$ in the directions of p_k; that is, α_k is chosen to minimize the function $\phi_\alpha(x_k + \alpha p_k)$. It can be shown (Ortega, 1988a, 186) that this will happen if we choose

$$\alpha = \alpha_k = \frac{p_k^T(b - Ax_k)}{p_k^T A p_k}$$

$$= p_k^T r_k / p_k^T A p_k \qquad \text{where } r_k = b - Ax_k$$

(In Algorithm 6.10.4, we will use a slightly different but equivalent expression of α.)

Choosing the Direction Vectors

The next question is how to choose the direction vectors p_k. The conjugate gradient (CG) method is a method that automatically generates the direction vectors. The direction vector needed at each step is generated at the previous step. Moreover, the direction vectors p_k have the remarkable property that

$$(p_i)^T A p_j = 0, \qquad 0 \le j < i, \quad i = 1, 2, \ldots, n-1$$

That is, *these vectors are orthogonal with respect to the inner product $x^T Ay$ defined by A.* The direction vectors p_k satisfying the above property are called **conjugate vectors**.

ALGORITHM 6.10.4 The Basic Conjugate Gradient Algorithm (The Classical Conjugate Gradient Method)

Step 1: Choose x_0 and ϵ. Set $p_0 = r_0 = b - Ax_0$.

Step 2: For $i = 0, 1, 2, 3, \ldots$ do

$w = Ap_i$

$\alpha_i = \|r_i\|_2^2 / p_i^T w$

$x_{i+1} = x_i + \alpha_i p_i$

$r_{i+1} = r_i - \alpha_i w$

Test for convergence: If $\|r_{i+1}\|_2^2 \ge \epsilon$, continue.

$\beta_i = \|r_{i+1}\|_2^2 / \|r_i\|_2^2$

$p_{i+1} = r_{i+1} + \beta_i p_i$

MATCOM Note: Algorithm 6.10.4 has been implemented in MATCOM program CONGRAD.

Example 6.10.7

$$A = \begin{pmatrix} 5 & 1 & 1 \\ 1 & 5 & 1 \\ 1 & 1 & 5 \end{pmatrix}, \qquad b = \begin{pmatrix} 7 \\ 7 \\ 7 \end{pmatrix}$$

$$x_0 = (0, 0, 0)^T, \qquad p_0 = r_0 = b - Ax^{(0)} = \begin{pmatrix} 7 \\ 7 \\ 7 \end{pmatrix}$$

$i = 0$:

$$w = Ap_0 = \begin{pmatrix} 49 \\ 49 \\ 49 \end{pmatrix}$$

$$\alpha_0 = \frac{\|r_0\|_2^2}{p_0^T w} = 0.1429$$

$$x_1 = x_0 + \alpha_0 p_0 = \begin{pmatrix} 1.0003 \\ 1.0003 \\ 1.0003 \end{pmatrix}$$

$$r_1 = r_0 - \alpha_0 w = \begin{pmatrix} -0.0021 \\ -0.0021 \\ -0.0021 \end{pmatrix}$$

$$\beta_0 = 9 \times 10^{-8}$$

$$p_1 = r_1 + \beta_0 p_0 = \begin{pmatrix} -0.0021 \\ -0.0021 \\ -0.0021 \end{pmatrix}$$

$i = 1$:

$$w = Ap_1 = \begin{pmatrix} -0.0147 \\ -0.0147 \\ -0.0147 \end{pmatrix}$$

$$\alpha_1 = 0.1429, \qquad x_2 = x_1 + \alpha_1 p_1 = \begin{pmatrix} 1.0000 \\ 1.0000 \\ 1.0000 \end{pmatrix}$$

Convergence

In the absence of roundoff errors the conjugate gradient method should converge in no more than n iterations. In fact, it can be shown that the error at every step decreases. Specifically, it can be proved (Ortega 1988a, 277) that Theorem 6.10.8 holds.

THEOREM 6.10.8

$$\|x - x_k\|_2 < \|x - x_{k-1}\|_2$$

where x is the exact solution, unless $x_{k-1} = x$.

However, the convergence can be extremely slow due to the ill-conditioning of A. This can be seen from the following result.

Convergence of the Conjugate Gradient Method

Define $\|s\|_A = \sqrt{s^T A s}$. Then an estimate of the rate of convergence is

$$\|x_k - x\|_A \le 2\alpha^k \|x_0 - x\|_A$$

where

$$\alpha = (\sqrt{\kappa} - 1)/(\sqrt{\kappa} + 1)$$

and

$$\kappa = \mathrm{Cond}(A) = \|A\|_2 \|A^{-1}\|_2 = \lambda_n / \lambda_1$$

Here λ_n and λ_1 are the largest and smallest eigenvalues of the symmetric positive definite matrix A (note that the eigenvalues of A are all positive).

Note: $\alpha = 0$ when $\mathrm{Cond}(A) = 1$. When $\alpha \to 1$, $\mathrm{Cond}(A) \to \infty$. Thus, *the larger* $\mathrm{Cond}(A)$, *the slower is the rate of convergence.* (For a proof, see Luenberger 1973, 187).

Preconditioning

Because a large condition number of A slows down the convergence of the conjugate gradient method, it is natural to see if the condition number of A can be improved before the method is applied; in this case we will be able to apply the basic conjugate gradient method to a preconditioned system. Indeed, the use of a good preconditioner accelerates the rate of convergence of the method substantially.

A basic idea of preconditioning is to find a nonsingular S such that $\text{Cond}(\tilde{A}) < \text{Cond}(A)$, where $\tilde{A} = SAS^T$. Once such an S is found, then we can solve $\tilde{A}\tilde{x} = \tilde{b}$, where $\tilde{x} = (S^{-1})^T x$, $\tilde{b} = Sb$, and then recover x from $\tilde{x}$ using

$$x = S^T \tilde{x}$$

The matrix S is usually defined for simplicity by

$$(S^T S)^{-1} = M$$

Note that M is symmetric positive definite and is called a **preconditioner**.

ALGORITHM 6.10.5 The Preconditioned Conjugate Gradient (PCG) Method

Find a preconditioner M.
Choose x_0 and ϵ.
Set $r_0 = b - Ax_0$, $p_0 = y_0 = M^{-1}r_0$.
For $i = 0, 1, 2, 3 \cdots$ do

(a) $w = Ap_i$
(b) $\alpha_i = y_i^T r_i / p_i^T w$
(c) $x_{i+1} = x_i + \alpha_i p_i$
(d) $r_{i+1} = r_i - \alpha_i w$
(e) $y_{i+1} = M^{-1} r_{i+1}$; that is, solve $My_{i+1} = r_{i+1}$
(f) Test for convergence: If $r_{i+1}^T y_{i+1} \geq \epsilon$, then if $\|r_{i+1}\|^2 \geq \epsilon$, continue.
(g) $\beta_i = y_{i+1}^T r_{i+1} / y_i^T r_i$
(h) $p_{i+1} = y_{i+1} + \beta_i p_i$

Note: If $M = I$, then the preconditioned conjugate gradient methods reduces to the basic conjugate gradient.

The next question, therefore, is how we find a preconditioner M. Several possibilities have been explored in the literature. Among them are

1. Polynomial preconditioning
2. Incomplete Cholesky factorization (ICF)

We shall describe ICF in the following. For a description of polynomial preconditioning, see Ortega (1988a, 206–208).

Incomplete Cholesky Factorization

Because A is symmetric positive definite, it admits the factorization:

$$A = LDL^T$$

where L is unit lower triangular and D is diagonal (Theorem 6.4.1).

If A is sparse, L is generally less sparse than A, because fill-in can occur. However, we can ignore the fill-in and obtain what is known as an *incomplete Cholesky factorization*.

The basic principle of the incomplete Cholesky factorization of $A = (a_{ij})$ is

If $a_{ij} \neq 0$, calculate l_{ij}.

If

$$a_{ij} = 0$$

set

$$l_{ij} = 0$$

ALGORITHM 6.10.6 Incomplete Cholesky Factorization (Ortega 1988a, 212)

Set $\ell_{11} = \sqrt{a_{11}}$
For $i = 2, \ldots, n$ do
  For $j = 1, 2, \ldots, i-1$ do
    If $a_{ij} = 0$, then $l_{ij} = 0$ else

$$\ell_{ij} = \frac{1}{\ell_{jj}}\left(a_{ij} - \sum_{k=1}^{j-1} \ell_{ik}\ell_{jk}\right)$$

$$\ell_{ii} = \sqrt{\left(a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2\right)}$$

Remark: Algorithm 6.10.6 requires computations of square roots. It may, therefore, not be carried out to completion. However, we can obtain a no-fill incomplete LDL^T factorization of A that avoids square root computations.

ALGORITHM 6.10.7 No-Fill Incomplete LDL^T Set $d_{11} = a_{11}$.

For $i = 2, \ldots, n$ do
  For $j = 1, 2, \ldots, i-1$ do
    if $a_{ij} = 0$, $\ell_{ij} = 0$,
    else,

$$\ell_{ij} = \left(a_{ij} - \sum_{k=1}^{j-1} \ell_{ik} d_{kk} \ell_{jk}\right) / d_{jj}.$$

$$d_{ii} = a_{ii} - \sum_{k=1}^{i-1} \ell_{ik}^2 d_{kk}$$

MATCOM Notes: Algorithms 6.10.6 and 6.10.7 have been implemented in MATCOM programs ICHOLES and NICHOL, respectively.

Use of Incomplete Cholesky Factorization in Preconditioning

Note that the incomplete Cholesky factorization algorithm mathematically gives the factorization of A in the form

$$A = LL^T + R$$

where $R \neq 0$. Because the best choice for a preconditioner M is the matrix A it-

preconditioner M is taken as

$$M = LL^T$$

In the preconditioned conjugate gradient algorithm (Algorithm 6.10.5), a symmetric positive definite system of the form

$$My = r$$

needs to be solved at each iteration with M as the coefficient matrix (step e). Because $M = LL^T$, this is equivalent to solving

$$Lx = r, \qquad L^T y = x$$

Because the coefficient matrix at each iteration is the same, the incomplete Cholesky factorization will be done only once. If the no-fill incomplete LDL^T is used, then mathematically we have

$$A = LDL^T + R$$

In this case we take the preconditioner M as

$$M = LDL^T$$

Then at each iteration of the preconditioned conjugate gradient algorithm, we need to solve a system of the form

$$My = r$$

which is equivalent to $Lx = r$, $Dz = x$, and $L^T y = z$. Again, L and D have to be computed once for all.

6.10.6 The Arnoldi Process and GMRES

In the last few years, a method called GMRES has received a considerable amount of attention by numerical linear algebraists in the context of solving large and sparse linear systems. The method is based on a classical scheme due to Arnoldi (1951) that constructs an orthonormal basis of a space called the Krylov subspace. The Krylov subspace of dimension m defined by an $n \times n$ matrix A and a vector v_1 is given by

$$\kappa_m(A, v_1) = \text{span}\{v_1, Av_1, \ldots, A^{m-1}v_1\}$$

The Arnoldi method can be implemented just by using matrix–vector multiplications and is, therefore, suitable for sparse matrices, because the zero entries are preserved.

The basic idea behind solving a large and sparse problem using the Arnoldi method is to project the problem onto the Krylov subspace of dimension $m < n$ using the orthonormal basis constructed by the method, solve the m-dimensional problem using a standard approach, and then recover the solution of the original problem from the solution of the projected problem.

We now summarize the essentials of the Arnoldi method, followed by an algorithmic description of GMRES.

Basic Arnoldi; (MGS is used in practice)

ALGORITHM 6.10.8 The Arnoldi Method

Step 1: Start: Choose a vector v_1 of norm 1 and an integer $m \le n$.

Step 2: Iterate: For $j = 1, 2, \ldots, m$ do

$$h_{ij} = (v_i^T A v_j), \quad i = 1, 2, \ldots, j$$

$$\hat{v}_{j+1} = Av_j - \sum_{i=1}^{j} h_{i,j} v_i$$

$$h_{j+1,j} = \|\hat{v}_{j+1}\|_2$$

and

$$v_{j+1} = \hat{v}_{j+1}/h_{j+1,j}$$

Notes:

1. The scalars $h_{i,j}$ have been chosen so that the vectors v_j are orthonormal.
2. Let V_m be the $n \times m$ matrix whose jth column is the column vector v_j; that is, $V_m \equiv (v_1, v_2, \ldots, v_m)$. Then the vectors in V_m form an orthonormal basis of the Krylov subspace $\kappa_m(A, v_1)$.
3. Define $\tilde{H}_m$ as the $(m+1) \times m$ matrix whose nonzero entries are the coefficients h_{ij} and H_m is the $m \times m$ matrix obtained from $\tilde{H}_m$ by deleting its last row. Then the matrix H_m is such that

$$AV_m - V_m H_m = h_{m+1,m}[0, 0, \ldots, 0, v_{m+1}] \tag{6.10.24}$$

or equivalently

$$AV_m = V_{m+1}\tilde{H}_m$$

where $V_{m+1} = (v_1, v_2, \ldots, v_{m+1})$.

4. A numerically viable way to implement the Arnoldi method is to use modified Gram–Schmidt or complete or partial reorthogonalization in step 2. Modified Gram–Schmidt is discussed in Chapter 7. See also Saad (1992).

Example 6.10.8

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 1 & 1 \end{pmatrix}, \qquad m = 2$$

$$v_1 = (1, 0, 0)^T$$

$j = 1$:

$$i = 1, \qquad h_{11} = 1$$

$$\hat{v}_2 = Av_1 - h_{11}v_1 = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$$

$$h_{21} = 1.4141$$

$$v_2 = \frac{\hat{v}_2}{h_{21}} = \begin{pmatrix} 0 \\ 0.7071 \\ 0.7071 \end{pmatrix}$$

$j = 2$:

$$i = 1, \qquad h_{12} = 3.5355, \qquad h_{22} = 3.5000$$

Form:

$$V_2 = \begin{pmatrix} 1 & 0 \\ 0 & 0.7071 \\ 0 & 0.7071 \end{pmatrix}, \qquad H_2 = \begin{pmatrix} 1 & 3.5335 \\ 1.4142 & 3.5000 \end{pmatrix}$$

Verify:

$$AV_2 - V_2H_2 = \begin{pmatrix} 0 & 0 \\ 0 & 1.0607 \\ 0 & -1.0607 \end{pmatrix}$$

GMRES Method

The GMRES (Generalized Minimal Residual) method is designed to minimize the norm of the residual vector $b - Ax$ over all vectors of the affine subspace $x_0 + \kappa_m(A, r_0)$, where x_0 is the initial vector and $r_0 = b - Ax_0$.

Given an initial approximation x_0 of the solution x of $Ax = b$, the generalized minimal residual (GMRES) computes an improved solution

$$x_1 = x_0 + z \in x_0 + \kappa_m(A, r_0)$$

where

$$r_0 = b - Ax_0$$

such that

$$\|b - Ax_1\| = \min_{z \in \kappa_m(A, r_0)} \|b - A(x_0 + z)\|$$

Suppose we run m steps of the Arnoldi method with $v_1 = \frac{r_0}{\|r_0\|}$; then at the end of m steps, we have

$$AV_m = V_{m+1}\tilde{H}_m$$

and

$$V_{m+1}^T r_0 = \|r_0\| e_1$$

So,

$$\begin{aligned} \min_{z \in \kappa_m(A, r_0)} \|r_0 - Az\| &= \min_{y \in R^m} \|r_0 - AV_m y\| \\ &= \min_{y \in R^m} \|r_0 - V_{m+1}\tilde{H}_m y\| \\ &= \min_{y \in R^m} \|V_{m+1}^T(r_0 - V_{m+1}\tilde{H}_m y)\| \\ &= \min_{y \in R^m} \| \|r_0\| e_1 - \tilde{H}_m y\| \end{aligned}$$

Once this least-squares problem,

$$\min_{y \in R^m} \| \|r_0\| e_1 - \tilde{H}_m y\|$$

is solved, obtaining a vector $y_0 \in R^m$, we have

$$x_1 \approx x_0 + V_m y_0$$

here the method of choice would be Givens because $\tilde{H}_m$ is upper Hessen.

Several methods of solving least squares problems including the one based on the QR factorization of $\tilde{H}_m$ (used by Saad and Schultz (1986)) will be described in Chapter 7.

The above discussions lead to the following algorithm.

ALGORITHM 6.10.9 Generalized Minimal Residual (GMRES) Method (Saad and Schultz 1986)

Step 1: Start: Choose x_0 and a dimension m of the Krylov subspace. Compute $r_0 = b - Ax_0$.

Step 2: Arnoldi process: Perform m steps of the Arnoldi algorithm starting with $v_1 = r_0/\|r_0\|$, to generate the matrix $\tilde{H}_m$ and the orthogonal matrix V_m.

Step 3: Form the approximate solution:

- Find the vector y_0 that minimizes the function $J(y) = \|\beta e_1 - \tilde{H}_m y\|$ where $e_1 = [1, 0, \ldots, 0]^T$, among all vectors y of R^m. $\beta = \|r_0\|$.
- Compute $x_1 = x_0 + V_m y_0$.

Remarks:

1. As noted earlier, the original implementation of the GMRES method in 1986 by Saad and Schultz uses the modified Gram–Schmidt method (to be described in Chapter 7) to compute the orthonormal basis V_m.

   Since then several other implementations have been proposed. Among them, the notable ones are by Walker (1988) and by Bai, Hu, and Reichel (1994). The latter implementation is more suitable for vector and parallel computers.
2. Very often in practice, the GMRES needs to be restarted after a suitably chosen value of m; or a suitable preconditioner, as in the case of the conjugate gradient method, has to be used. The discussions of these are beyond the scope of this book. For details, see the recent book by Barrett, et al. (1994) and Saad (1995).

Solving a shifted system. An important observation is that the Arnoldi basis V_m is invariant under a diagonal shift of A: If we were to use $A - \sigma I$ instead of A in Arnoldi, we would obtain the same sequence, $\{v_1, \ldots, v_m\}$. This is because the Krylov subspace κ_m is the same for A and $A - \sigma I$, provided the initial vector v_1 is the same. Note that from (6.10.24) we have

$$(A - \sigma I)V_m = V_m(H_m - \sigma I) + h_{m+1,m} v_{m+1} e_m^T$$

which means that if we run the Arnoldi process with the matrix $A - \sigma I$, we would obtain the same matrix V_m, but the matrix H_m would have its diagonal shifted by σI.

Solving shifted systems arises in many applications, such as the computation of the frequency response matrix of a control system (see Section 6.4.7). Some efficient techniques of computing the frequency reponse matrix for large sparse linear control systems, using restarted GRMES with incomplete LU factorization, appears in a recent Ph.D. thesis of Choudhury (1994).

Relationship Between the GMRES and the Conjugate Gradient Method

From our description of the conjugate gradient method, it was not clear that the conjugate gradient method is also another Krylov subspace method. As the GMRES method minimizes the norm of the residual vector $r_0 = b - Ax_0$ over all vectors of the affine Krylov subspace $x_0 + \kappa_m(A, r_0)$, the conjugate gradient iterate x_{k+1} minimizes the error (in A–norm) over all the vectors in the affine Krylov subspace $x_0 + \kappa_{k+1}(A, p_0)$. Also, note that $\kappa_{k+1}(A, p_0) = \kappa_{k+1}(A, r_0)$.

Specifically, we state the following theorem from Ortega (1988a), without proof. The proof can be found in Ortega (1988a, 273–4).

THEOREM 6.10.9 Each conjugate gradient iterate $x_{k+1}(k = 0, 1, \ldots, n-1)$ minimizes $\|\hat{x} - x\|_A$ over $x_0 + \kappa_{k+1}(A, p_0)$, where p_0 is the initial direction vector.

The QMR Method

A method similar to the GMRES method called the Quasi-Minimal Residual (QMR) method, devised by Freund and Nachtigal (1991), has also become popular in recent years in the context of solving a sparse nonsymmetric linear system. The method reduces the full linear system problem to a tridiagonal system and then solves the reduced system in the least-squares sense. It uses a variant of the well-known Lanczos method called look-ahead Lanczos to deal with the breakdowns in the Lanczos process. The Lanczos process will be described briefly in Chapter 8 of this book. For details of the QMR method, see Freund and Nachtigal (1991).

Iterative Methods Not Discussed in the Book

In addition to the QMR method, some other important iterative methods not discussed in the book are: BiConjugate Gradient (BiCG), BiConjugate Gradient Stabilized (Bi-CGSTAB), Conjugate Gradient Squared (CGS), Chebyshev iteration, MINRES and SYMMLQ. While BICG, Bi-CGSTAB, CGS, and the Chebyshev iteration methods are all applicable to nonsymmetric systems, MINRES and SYMMLQ are variants of the conjugate gradient method, and are designed for symmetric indefinite large and sparse linear systems problems. For brief discussions, implementational details, and comparative study, see the book *Templates* (Barrett, et al. 1994) and Saad (1995).

6.11 Review and Summary

The following reviews the most important results discussed in this chapter.

1. *Numerical methods for arbitrary linear system problems:* Two types of methods—direct and iterative—were discussed.

   (a) Direct methods are Gaussian elimination and QR factorization methods.

   - Gaussian elimination without row interchanges, when it exists, gives a factorization of $A : A = LU$. The system $Ax = b$ is then solved by first solving the lower triangular system $Ly = b$, followed by solving the upper triangular system $Ux = y$.

     The method requires $n^3/3$ flops. It is unstable for arbitrary matrices and is not recommended for practical use unless the matrix

A is symmetric positive definite. The growth factor can be arbitrarily large for an arbitrary matrix.

- Gaussian elimination with partial pivoting gives a factorization of A:

  $$MA = U$$

  Once we have this factorization, we can solve $Ax = b$ by solving the upper triangular system $Ux = b'$, where $b' = Mb$. The process requires $n^3/3$ flops and $O(n^2)$ comparisons. In theory, there are some risks involved, but in practice, this is a stable algorithm. It is the most widely used practical algorithm for solving a dense linear system.

- Gaussian elimination with complete pivoting gives

  $$MAQ = U$$

  Once we have this factorization, we can solve $Ax = b$ by first solving

  $$Uy = b', \qquad \text{where } y = Q^T x, \qquad b' = Mb$$

  and then recovering x from

  $$x = Qy$$

  The process requires $n^3/3$ flops and $O(n^3)$ comparisons. Thus, it is more expensive than Gaussian elimination with partial pivoting, but it is more stable (the growth factor ρ in this case is bounded by a slowly growing function of n, whereas the growth factor ρ with Gaussian elimination using partial pivoting can be as big as 2^{n-1}).

- The orthogonal triangularization methods are based on the QR factorization of A:

  $$A = QR$$

  Once we have this factorization, we can solve $Ax = b$ by solving the upper triangular system $Rx = b'$, where $b' = Q^T b$. We can use either the Householder method or the Givens method to achieve this factorization. The Householder method is more economical than the Givens method ($2n^3/3$ flops versus $4n^3/3$ flops). Both the methods have the guaranteed stability.

(b) Iterative methods are the Jacobi, Gauss–Seidel, and SOR methods we discussed. A generic formulation of an iterative method is:

$$x^{(k+1)} = Bx^{(k)} + d$$

Different methods differ in the way B and d are chosen. Writing $A = L + D + U$, we have:

- For the Jacobi method:

  $$B = B_J = -D^{-1}(L + U)$$
  $$d = b_J = D^{-1}b$$

- For the Gauss–Seidel method:

  $$B = B_{GS} = -(D + L)^{-1}U$$
  $$d = b_{GS} = (D + L)^{-1}b$$

- For the SOR method:

$$B = B_{SOR} = (D + \omega L)^{-1}[(1 - w)D - wU]$$
$$d = b_{SOR} = \omega(D + \omega L)^{-1}b$$

(ω is the relaxation parameter).

- The iteration

$$x^{(k+1)} = Bx^{(k)} + d$$

converges for any arbitrary choice of the initial approximation $x^{(1)}$ if and only if the spectral radius of B is less than 1 (Theorem 6.10.1). A sufficient condition for convergence is $\|B\| < 1$ (Theorem 6.10.1).

- Using this sufficient condition, it has been shown that both the Jacobi and Gauss–Seidel methods converge when A is a diagonally dominant matrix (Corollaries 6.10.1 and 6.10.2).
- The Gauss–Seidel method also converges when A is symmetric positive definite (Theorem 6.10.2).
- The SOR iteration cannot converge for any arbitrary choice of the initial approximation, unless the relaxation parameter ω lies in $(0, 2)$ (Theorem 6.10.3).
- If the matrix A is symmetric positive definite, then the SOR iteration is guaranteed to converge for any arbitrary choice of ω in the interval $(0, 2)$ (Theorem 6.10.4).
- For a consistently ordered and 2-cyclic matrix A with nonzero diagonal entries, the optimal choice of ω, denoted by ω_{opt}, is given by

$$\omega_{opt} = \frac{2}{1 + \sqrt{1 - \rho(B_J)^2}}$$

assuming that the eigenvalues of B_J are real and $\rho(B_J) < 1$. $\rho(A)$ stands for the spectral radius of A (Theorem 6.10.4).

2. *Special Systems:* Symmetric positive definite, diagonally dominant, Hessenberg, and tridiagonal systems were discussed.

(a) Two methods—Gaussian elimination without pivoting and Cholesky—are described for symmetric positive definite systems.

- The Gaussian elimination without pivoting gives a factorization of A:

$$A = LDL^T$$

The system $Ax = b$ is then solved by first solving the lower triangular system $LDy = b$, followed by solving the upper triangular system $L^Tx = y$. The method requires $n^3/3$ flops and does not require any square root evaluations. It is stable ($\rho = 1$).

- The Cholesky factorization algorithm computes a factorization of A in the form $A = HH^T$, where H is lower triangular with positive diagonal entries. Once we have this factorization, we can solve the system $Ax = b$ by first solving the lower triangular system $Hy = b$, where $y = H^Tx$, followed by solving the upper triangular system $H^Tx = y$. The method requires $n^3/6$ flops and n square root evaluations. It is stable.

(b) Gaussian elimination without pivoting is stable ($\rho \le 2$) for the diagonally dominant system.

(c) For the Hessenberg system, Gaussian elimination with partial pivoting requires only $O(n^2)$ flops to solve an $n \times n$ Hessenberg system. It is stable ($\rho \le n$).

(d) For a tridiagonal system, Gaussian elimination with partial pivoting requires only $O(n)$ flops. It is stable ($\rho \le 2$).

3. *Inverse, determinants, and leading principal minors:* The inverse and the determinant of a matrix A can be readily computed once a factorization of A is available.

(a) First, consider inverses.

- Once a factorization of the matrix A is obtained, the inverse can be computed by solving a set of linear systems with different right-hand sides. The same factorization can be used to solve the different systems (see Section 6.5). Note that most problems involving inverses can be recast so that the inverse does not have to be computed explicitly. Furthermore, there are matrices (such as triangular, etc.) whose inverses are easily computed.
- The inverse of a matrix B that differs from a matrix A only by a rank-one perturbation can be readily computed, once the inverse of A has been found, by using the *Sherman–Morrison formula*: Let $B = A - uv^T$. Then $B^{-1} = A^{-1} + \alpha(A^{-1}uv^TA^{-1})$, where $\alpha = 1/(1 - v^TA^{-1}u)$. There is a generalization of this formula, known as the *Woodbury formula*.

(b) The determinant is rarely required in practical applications. However, the determinant of A can be computed immediately, once a factorization of A has been obtained.

- If we use Gaussian elimination with partial pivoting, we have

$$MA = U$$

Then

$$\det(A) = (-1)^r a_{11}a_{22}^{(1)} \dots a_{nn}^{(n-1)}$$

where r is the number of row interchanges made during the elimination process. $a_{11}, a_{22}^{(1)}, \dots, a_{nn}^{(n-1)}$ are the pivot entries, which appear as the diagonal entries of U. Similarly, other factorizations can be readily used to find the determinant.

4. *The condition number and accuracy of solution:* In the linear system problem $Ax = b$, the input data are A and b. Impurities may exist either in A or in b or in both.

We have presented perturbation analyses in all three cases. The results are contained in Theorems 6.6.1, 6.6.2, and 6.6.3. Theorem 6.6.3 is the most general theorem.

In all three cases, it turns out that

$$\text{Cond}(A) = \|A\| \, \|A^{-1}\|$$

is the deciding factor. If this number is large, then a small perturbation in the input data may cause a large relative error in the computed solution. In this case, the system is called an ill-conditioned system, otherwise it is well conditioned. The matrix A having a large condition number is called an ill-conditioned matrix.

Some important properties of the condition number of a matrix were listed. Some well-known ill-conditioned matrices are the **Hilbert matrix**, the **Pie matrix**, and the **Wilkinson bidiagonal matrix**.

The condition number, of course, has a noticeable effect on the accuracy of the solution.

A computed solution can be considered accurate only when the product of both Cond(A) and the relative residual is small (Theorem 6.7.1). *Thus, a small relative residual alone does not guarantee the accuracy of the solution.*

A frequently asked question is: How large does Cond(A) have to be before the system $Ax = b$ is considered to be ill conditioned?

The answer depends upon the accuracy of the input data and the level of tolerance of the error in the solution.

In general, if the data are approximately accurate and if Cond(A) = 10^s, then there are about $(t - s)$ significant digits of accuracy in the solution if it is computed in t-digit arithmetic.

- Computing the condition number from the definition is clearly expensive; it involves finding the norm of the inverse of A, and finding the inverse of A is about three times as expensive as solving the linear system itself.
- Two condition number estimators, the LINPACK condition number estimator and the Hager's norm-1 condition number estimator, were described.
- There are symptoms exhibited during the Gaussian elimination with pivoting, such as a small pivot, a large computed solution, and a large residual, that merely indicate if a system is ill conditioned, but these are not sure tests.
- When componentwise perturbations are known, Skeel's condition number can be useful, especially when the norms of the columns of the inverse matrix vary widely.

5. *Iterative refinement:* Once a solution has been computed, an inexpensive way to refine the solution iteratively, known as the iterative refinement procedure, was described (Algorithm 6.9.1).

   The iterative refinement technique is a very useful technique.

6. *The conjugate gradient and GMRES methods:* The conjugate gradient method, when used with an appropriate preconditioner, is one of the most widely used methods for solving a large and sparse symmetric positive definite linear system.

   Only one type of preconditioner, the incomplete Cholesky factorization (ICF), was described in this book. The basic idea of ICF is to compute the Cholesky factorization of A for the nonzero entries of A only, leaving the zero entries as zeros.

We described only basic Arnoldi and Arnoldi-based GMRES methods. In practice, the modified Gram–Schmidt process needs to be used in the implementation of the Arnoldi method, and GMRES needs to be used with a proper preconditioner. The conjugate gradient method is direct in theory but iterative in practice. It is extremely slow when A is ill conditioned, and a preconditioner is needed to accelerate the convergence.

EXERCISES

Note: Use MATLAB wherever needed and appropriate.

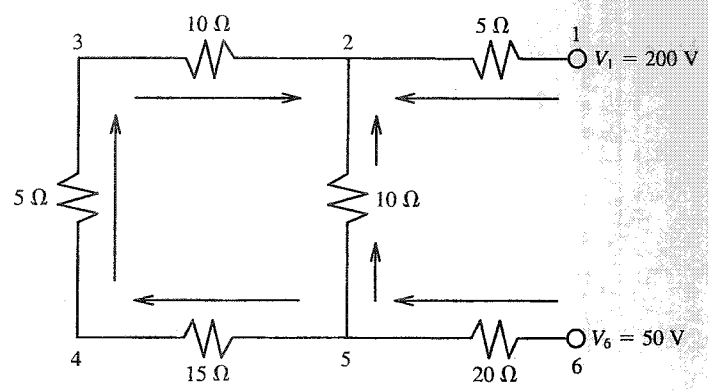
Section 6.3

1. An engineer requires 5000, 5500, and 6000 yd^3 of sand, cement, and gravel for a building project. She obtains material from three sources. A distribution of each material in these sources is given as follows:

Source	Sand %	Cement %	Gravel %
1	60	20	20
2	40	40	20
3	20	30	50

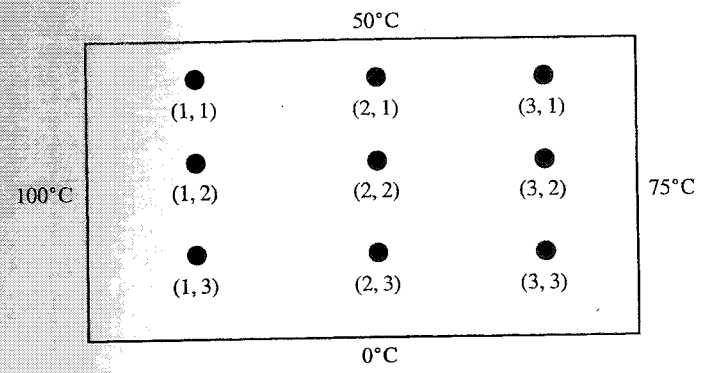
   How many cubic yards must the engineer obtain from each source to meet her needs?

2. If the input to reactor 1 in the reactor problem of Section 6.3.2 is decreased 10%, what is the percent change in the concentration of the other reactors?
3. Consider the following circuit diagram:



   Set up a linear system to determine the current between nodes.

4. Using the difference equation (6.3.30), set up a linear system for heat distribution at the following interior points of a heated plate whose boundary temperatures are held constant:



5. Derive the linear system for the finite difference approximation of the elliptic equation

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = f(x, y)$$

The domain is in the unit square, $\Delta x = \Delta y = 0.25$, and the boundary conditions are given by

$$T(x, 0) = 1 - x$$
$$T(1, y) = y$$
$$T(0, y) = 1$$
$$T(x, 1) = 1$$

6. For the previous problem, if

$$f(x, y) = -\pi^2 \sin(\pi x)\sin(\pi y)$$

then the analytic solution to the elliptic equation

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = f(x, y)$$

with the same boundary conditions as in Exercise 5, is given by

$$T(x, y) = 1 - x + xy + (\tfrac{1}{2})\sin(\pi x)\sin(\pi y)$$

(Celia and Gray 1992, 105–6).

(a) Use the finite difference scheme of Section 6.3 to approximate the values of T at the interior points with $\Delta x = \Delta y = 1/n$, $n = 4, 8, 16$.
(b) Compare the values obtained in (a) with the exact solution.
(c) Write down the linear system arising from finite element method of the solution of the two-point boundary value problem: $-2u'' + 3u = x^2$, $0 \le x \le 1$, $u(0) = u(1) = 0$, using the same basic functions $\phi_j(x)$ as in the book and uniform grid.

Section 6.4

7. Solve the linear system $Ax = b$, where b is a vector with each component equal to 1 and with each A from Exercise 18 of Chapter 5, using
(a) Gaussian elimination without pivoting
(b) Gaussian elimination with partial and complete pivoting
(c) QR factorization
8. (a) Solve each of the systems of Exercise 7 using partial pivoting but without explicit factorization (Section 6.4.4).
(b) Compute the residual vector in each case.
9. Solve

$$\begin{pmatrix} 0.0001 & 1 & 1 \\ 3 & 1 & 1 \\ 1 & 2 & 3 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 2.0001 \\ 3 \\ 3 \end{pmatrix}$$

using Gaussian elimination without and with partial pivoting and compare the answers.

10. Consider m linear systems

$$Ax_i = b_i, \qquad i = 1, 2, \ldots, m$$

(a) Develop an algorithm to solve the above systems using Gaussian elimination with complete pivoting. Your algorithm should take advantage of the fact that all the m systems have the same system matrix A.
(b) Determine the flop count of your algorithm.
(c) Apply your algorithm in the special case where b_i is the ith column of the identity matrix, $i = 1, \ldots, n$.
(d) Use the algorithm in (c) to show that the inverse of an $n \times n$ matrix A can be computed using Gaussian elimination with complete pivoting in about $\frac{4}{3}n^3$ flops.
(e) Apply the algorithm in (c) to compute the inverse of

$$A = \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \end{pmatrix}$$

11. Consider the system $Ax = b$, where both A and b are complex. Show how the system can be solved using real arithmetic only. Compare the flop count in this case with that needed to solve the system with Gaussian elimination using complex arithmetic.
12. (a) Compute the Cholesky factorization of

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1.001 & 1.001 \\ 1 & 1.001 & 2 \end{pmatrix}$$

using

(i) Gaussian elimination without pivoting
(ii) Cholesky algorithm

(b) In part (a) verify that $\max |a_{ij}^{(k)}| \le \max |a_{ij}^{(k-1)}|$, $k = 1, 2$.
(c) What is the growth factor?

13. Using the results of Exercise 12, solve the system $Ax = b$ where

$$b = \begin{pmatrix} 3 \\ 3.0020 \\ 4.0010 \end{pmatrix}$$

and A is the same as in Exercise 12.

14. (a) Show that

$$A = \begin{pmatrix} 4 & -1 & -1 & 0 \\ -1 & 4 & 0 & -1 \\ -1 & 0 & 4 & -1 \\ 0 & -1 & -1 & 4 \end{pmatrix}$$

is positive definite with and without finding the Cholesky factorization.

(b) Solve the system

$$Ax = \begin{pmatrix} 2 \\ 2 \\ 2 \\ 2 \end{pmatrix}$$

15. (a) Develop an algorithm to solve a tridiagonal system using Gaussian elimination with partial pivoting.
(b) Show that the growth factor in this case is bounded by 2. (*Hint:* $\max |a_{ij}^{(1)}| \le 2 \max |a_{ij}|$.)

16. (a) Show that Gaussian elimination applied to a column diagonally dominant matrix preserves diagonal dominance at each step of reduction; that is, if $A = (a_{ij})$ is column diagonally dominant, then so is $A^{(k)} = (a_{ij}^{(k)})$, $k = 1, 2, \ldots, n-1$.
(b) Show that the growth factor using Gaussian elimination without pivoting is bounded by 2.
(c) Verify the statement of part (a) with the matrix of Exercise 14.
(d) Construct a 2×2 column diagonally dominant matrix whose growth factor with Gaussian elimination is larger than 1 but less than or equal to 2.

17. Solve the tridiagonal system

$$\begin{pmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

(a) Using Gaussian elimination
(b) Computing the LU factorization of A directly from $A = LU$

18. Solve the diagonally dominant system

$$\begin{pmatrix} 10 & 1 & 1 & 1 \\ 1 & 10 & 1 & 1 \\ -1 & 0 & 10 & 1 \\ -1 & -1 & -1 & 10 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 13 \\ 13 \\ 10 \\ 7 \end{pmatrix}$$

using Gaussian elimination without pivoting. Compute the growth factor.

19. (a) Develop efficient algorithms for triangularizing (i) an upper Hessenberg matrix and (ii) a lower Hessenberg matrix, using Gaussian elimination with partial pivoting.
(b) Show that when A is upper Hessenberg, Gaussian elimination with partial pivoting gives

$$|a_{ij}^{(k)}| \le k + 1, \qquad \text{if } |a_{ij}| \le 1$$

Hence, deduce that the growth factor in this case is bounded by n.
(c) Apply your algorithms to solve the systems:

$$\text{(i)} \quad \begin{pmatrix} 1 & 2 & 2 & 3 \\ 3 & 4 & 5 & 7 \\ 0 & 0.0001 & 2 & 3 \\ 0 & 0 & 0.0001 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 6 \\ 7 \\ 8 \\ 9 \end{pmatrix}$$

$$\text{(ii)} \quad \begin{pmatrix} 1 & 0.0001 & 0 \\ 0 & 2 & 0.0001 \\ 0 & 0 & 3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1.0001 \\ 2.0001 \\ 3.0000 \end{pmatrix}$$

$$\text{(iii)} \quad \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & 4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}$$

(d) Compute the growth factor in each case.
(e) Suppose the data in these problems are accurate to four digits and you seek an accuracy of three digits in your solution. Identify which problems are ill conditioned.

20. (a) Find the QR factorization of

$$A = \begin{pmatrix} 1 & 1 \\ 10^{-5} & 0 \\ 0 & 10^{-5} \end{pmatrix}$$

(b) Using the results of (a), solve $Ax = b$, where

$$b = \begin{pmatrix} 2 \\ 10^{-5} \\ 10^{-5} \end{pmatrix}$$

21. Find the LU factorization of

$$A = \begin{pmatrix} T & I & 0 \\ I & T & I \\ 0 & I & T \end{pmatrix}$$

where

$$T = \begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{pmatrix}$$

Use this factorization to solve $Ax = b$; where each element of b is 2.

Section 6.5

22. Find the determinant and the inverse of each of the matrices of Exercise 18 of Chapter 5 using
    (a) Gaussian elimination with partial and complete pivoting
    (b) QR factorization
23. (a) Let

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$$

Assume that A_{11} and A_{22} are square and that A_{11} and $A_{22} - A_{21}A_{11}^{-1}A_{12}$ are nonsingular. Let

$$B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

be the inverse of A. Show that

$$B_{22} = (A_{22} - A_{21}A_{11}^{-1}A_{12})^{-1}$$
$$B_{12} = -A_{11}^{-1}A_{12}B_{22}$$
$$B_{21} = -B_{22}A_{21}A_{11}^{-1}$$

and

$$B_{11} = A_{11}^{-1} - B_{12}A_{21}A_{11}^{-1}$$

(b) How many flops are needed to compute A^{-1} using the results of (a) if A_{11} and A_{22} are, respectively, $m \times m$ and $p \times p$?
(c) Use your results to compute A^{-1}, where

$$A = \begin{pmatrix} 4 & 0 & -1 & -1 \\ 0 & 4 & -1 & -1 \\ -1 & -1 & 4 & 0 \\ -1 & 1 & 0 & 4 \end{pmatrix}$$

24. Let

$$A = \begin{pmatrix} 1 & 2 & 1 \\ 2 & 4.0001 & 2.0002 \\ 1 & 2.0002 & 2.0004 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} 0 & 2 & 1 \\ 2 & 4.0001 & 2.0002 \\ 1 & 2.0002 & 2.0004 \end{pmatrix}$$

Write B in the form $B = A - uv^T$; then compute B^{-1} using the Sherman–Morrison formula, knowing

$$A^{-1} = 10^4 \begin{pmatrix} 4.0010 & -2.0006 & 0.0003 \\ -2.0006 & 1.0004 & -0.0002 \\ 0.0003 & -0.0002 & 0.0001 \end{pmatrix}$$

25. Suppose you have solved a linear system with A as the system matrix. Then show how you can solve the augmented system

$$\begin{pmatrix} A & \delta \\ c & \alpha \end{pmatrix} \begin{pmatrix} x \\ x_{n+1} \end{pmatrix} = \begin{pmatrix} b \\ b_{n+1} \end{pmatrix}$$

where A is nonsingular and $n \times n$ and a, b, and c are vectors, using the solution you have already obtained. Apply your result to solve

$$\begin{pmatrix} 1 & 2 & 3 & 1 \\ 4 & 5 & 6 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 2 \end{pmatrix} y = (6, 15, 3, 1)^T$$

Sections 6.6 and 6.7

26. Consider the symmetric system $Ax = b$, where

$$A = \begin{pmatrix} 0.4445 & 0.4444 & -0.2222 \\ 0.4444 & 0.4445 & -0.2222 \\ -0.2222 & -0.2222 & 0.1112 \end{pmatrix}, \quad b = \begin{pmatrix} 0.6667 \\ 0.6667 \\ -0.3332 \end{pmatrix}$$

The exact solution of the system is

$$x = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

(a) Make a small perturbation δb in b, keeping A unchanged. Solve the system $Ax' = b + \delta b$. Compare x' with x. Compute Cond(A) and verify the appropriate inequality in the text.
(b) Make a small perturbation ΔA in A such that $\|\Delta A\| \le 1/\|A^{-1}\|$. Solve the system $(A + \Delta A)x' = b$. Compare x' with x and verify the appropriate inequality in the text. (*Hint:* $\|A^{-1}\|_2 = O(10^4)$.)

27. Prove the inequality

$$\frac{\|\delta x\|}{\|x + \delta x\|} \le \text{Cond}(A)\frac{\|\Delta A\|}{\|A\|}$$

where $Ax = b$ and $(A + \Delta A)(x + \delta x) = b$.
Verify the inequality for the system

$$\begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \quad \text{using} \quad \Delta A = \begin{pmatrix} 0 & 0 & 0.00003 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

28. (a) How are Cond(A) and Cond(A^{-1}) related?
    (b) Show that
        (i) Cond(A) ≥ 1
        (ii) Cond(A^TA) = (Cond(A))^2
29. (a) Let O be an orthogonal matrix. Then show that Cond(O) with respect to the 2-norm is 1.

(b) Show that the Cond(A) with respect to the 2-norm is 1 if and only if A is a scalar multiple of an orthogonal matrix.

30. Let $U = (u_{ij})$ be a nonsingular upper triangular matrix. Then show that with respect to the infinity norm,

$$\text{Cond}(U) \geq \frac{\max(|u_{ii}|)}{\min(|u_{ii}|)}$$

Then construct a simple example of an ill-conditioned nondiagonal symmetric positive definite matrix.

31. Let $A = LDL^T$ be a symmetric positive definite matrix. Let $D = \text{diag}(d_{ii})$. Then show that, with respect to 2-norm,

$$\text{Cond}(A) \geq \frac{\max(d_{ii})}{\min(d_{ii})}$$

Then construct an example for an ill-conditioned nondiagonal symmetric positive definite matrix.

32. (a) Show that for any matrix A, Cond(A) with respect to 2-norm is given by

$$\text{Cond}(A) = \frac{\sigma_{\max}}{\sigma_{\min}}$$

where $\sigma_{\max}$ and $\sigma_{\min}$ are, respectively, the largest and the smallest singular values of A.

(b) Use the preceding expressions for Cond(A) to construct an example of an ill-conditioned matrix as follows: Choose two nondiagonal orthogonal matrices U and V (in particular, they can be chosen as Householder matrices) and a diagonal matrix Σ with one or several small diagonal entries. Then the matrix

$$A = U\Sigma V^T$$

has the same condition-number as Σ and is ill conditioned.

33. (a) Construct your own example to show that a small residual does not necessarily guarantee that the solution is accurate.

(b) Give a proof of Theorem 6.7.1 (residual theorem).

(c) Using the residual theorem, prove that if an algorithm produces a small residual for every well-conditioned matrix, it is weakly stable.

34. (a) Find for what values of a is the matrix

$$A = \begin{pmatrix} 1 & a \\ a & 1 \end{pmatrix}$$

ill conditioned?

(b) Let $a = 0.999$. Solve the system

$$Ax = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

using Gaussian elimination without pivoting.

(c) What is the condition number of A?

Section 6.9

35. Apply iterative refinement to the system of Exercise 9 using

$$x^{(1)} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

Estimate Cond(A) from $x^{(1)}$ and $x^{(2)}$ and compare it with the actual $\text{Cond}_2(A) = 24.6414$.

Section 6.10

36. Consider the linear system of Exercise 14(a).

(a) Why do both the Jacobi and Gauss–Seidel methods converge with an arbitrary choice of initial approximation for this system?

(b) Carry out five iterations of both the methods with the same initial approximation

$$x^{(1)} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

and compare the rates of convergence.

37. Construct an example to show that the convergence of the Jacobi method does not necessarily imply that the Gauss–Seidel method will converge.

38. Let the $n \times n$ matrix A be partitioned into the form

$$A = \begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1N} \\ A_{21} & A_{22} & \cdots & A_{2N} \\ \vdots & \vdots & & \vdots \\ A_{N,1} & A_{N,2} & \cdots & A_{N,N} \end{pmatrix}$$

where each diagonal block A_{ii} is square and nonsingular. Consider the linear system

$$Ax = b$$

with A as above and x and b partitioned commensurately.

(a) Write down the block Jacobi, block Gauss–Seidel, and block SOR iterations for the linear system $Ax = b$. (*Hint*: Write $A = L + D + U$, where $D = \text{diag}(A_{11}, \ldots, A_{NN})$ and L and U are strictly block lower and upper triangular matrices.)

(b) If A is symmetric positive definite, then show that $U = L^T$ and D is positive definite. In this case, using the corresponding results in the scalar cases, prove that, with an arbitrary choice of the initial approximation, block Gauss–Seidel always converges and block SOR converges if and only if $0 < \omega < 2$.

39. Consider the block system arising in the solution of the discrete Poisson equation $u_{xx} + u_{yy} = f$:

$$\begin{pmatrix} T & -I & & & \\ -I & T & -I & 0 & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & -I \\ 0 & & & -I & T \end{pmatrix} x = b$$

where

$$T = \begin{pmatrix} 4 & -1 & & 0 \\ -1 & \ddots & \ddots & \\ & \ddots & \ddots & -1 \\ 0 & & -1 & 4 \end{pmatrix}$$

(a) Show that the block Jacobi iteration in this case is

$$Tx_i^{(k+1)} = x_{i+1}^{(k)} + x_{i-1}^{(k)} + b_i, \qquad i = 1, \ldots, N$$

Write down the block Gauss–Seidel and block SOR iterations for this system.

(b) Show that every block tridiagonal matrix with nonsingular diagonal blocks is consistently ordered and 2-cyclic. Find the permutation matrix P in the case of the matrix in part (a).

40. (a) Prove that $\sqrt{\lambda_1}\|x\|_2 \le \|x\|_A \le \sqrt{\lambda_n}\|x\|_2$, where A is a symmetric positive definite matrix with the eigenvalues $0 < \lambda_1 \le \lambda_2 \le \cdots \le \lambda_n$.

(b) Using the result in (a), prove that $\|x - x_k\|_2 \le 2\sqrt{\kappa}\alpha^k\|x - x_0\|_2$, for the conjugate gradient method.

41. Show that the Jacobi method converges for a 2×2 symmetric positive definite system.

42. For the system of Exercise 39, compute $\rho(B_J)$, $\rho(B_{GS})$, and ω_{opt} with $N = 50, 100, 1000$. Compare the rate of convergence of the SOR iteration using the optimal value ω_{opt} in each case with that of Gauss–Seidel, without actually performing the iterations.

43. Consider the block diagonal system

$$\begin{pmatrix} A & 0 & 0 & \cdots & 0 \\ 0 & A & 0 & \cdots & 0 \\ \vdots & \vdots & A & \cdots & \vdots \\ 0 & 0 & \vdots & A & 0 \\ 0 & 0 & \cdots & 0 & A \end{pmatrix}_{25\times 25} \qquad b = \begin{pmatrix} 1 \\ 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}$$

where

$$A = \begin{pmatrix} 2 & -1 & & 0 \\ -1 & 2 & \ddots & \\ & \ddots & \ddots & -1 \\ 0 & & -1 & 2 \end{pmatrix}$$

Compute $\rho(B_J)$ and $\rho(B_{GS})$ and find how they are related. Solve the system using five iterations of Gauss–Seidel and SOR with optimal value of ω. Compare the rates of convergence.

44. Prove that the choice of

$$\alpha = p^T(Ax - b)/p^T Ap$$

minimizes the quadratic function

$$\begin{aligned} \phi_\alpha &= \phi(x - \alpha p) \\ &= \tfrac{1}{2}(x - \alpha p)^T A(x - \alpha p) - b^T(x - \alpha p) \end{aligned}$$

45. Show that the eigenvectors of A are the direction vectors, obtained by the conjugate gradient method.

46. (a) Apply the incomplete Cholesky factorization algorithm to an unreduced tridiagonal matrix T and show that the result is the usual Cholesky factorization of T. Verify the above statement with

$$T = \begin{pmatrix} 4 & -1 & & 0 & \\ -1 & \ddots & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & -1 \\ & 0 & & -1 & 4 \end{pmatrix}_{5\times 5}$$

(b) Apply the SOR iteration to the matrix T in (a) with $\omega = 1.5$ using $x^{(0)} = (0, 0, 0, 0, 0)^T$, and make a table with the results of the iterations. Choose b randomly.

47. Let $p_0, p_1, \ldots, p_{n-1}$ be the direction vectors generated by the basic conjugate gradient algorithm. Let $r_k = b - Ax_k$, $k = 0, 1, \ldots, n-1$. Then prove that

(a) $r_k \in \text{span}(p_0, \ldots, p_k)$, $k = 0, 1, 2, \ldots, n-1$.

(b) $\text{span}(p_0, \ldots, p_i) = \text{span}(r_0, Ar_0, \ldots, A^i r_0)$, $i = 0, 1, \ldots, n-1$.

(c) $r_0, \ldots, r_{n-1}$ are mutually orthogonal.

48. (Multisplitting) Consider the iteration

$$x^{(k+1)} = Bx^{(k)} + d$$

where B is given by

$$B = \sum_{i=1}^{k} D_i B_i^{-1} C_i, \qquad d = \left(\sum_{i=1}^{k} D_i B_i^{-1}\right) b$$

and

$$A = B_i - C_i, \qquad i = 1, \ldots, k, \qquad \sum_{i=1}^{k} D_i = I \quad (D_i \ge 0)$$

Develop the Jacobi, the Gauss–Seidel, and the SOR methods based on the multisplitting of A.

49. Apply the Jacobi, the Gauss–Seidel, and the SOR (with optimal relaxation factor) methods to the system in the example following Theorem 6.10.5 (Example 6.10.5) and verify the statement about number of iterations made there about different methods.

50. Give a proof of Theorem 6.10.7.

51. Prove Theorem 6.10.8.

MATLAB AND MATCOM PROGRAMS AND PROBLEMS

Note: You will need the programs **lugsel**, **inlu**, **inparpiv**, **incompiv**, **givqr**, **compiv**, **hagcond1**, **invuptr**, **iterref**, **jacobi**, **gaused**, **sucov**, and **nichol** from MATCOM.

1. (a) Write a MATLAB program called **forelm**, based on Algorithm 6.4.1:

   $[y] = \textbf{forelm}\ (L, b)$

   to solve a nonsingular lower triangular system $Ly = b$ using forward elimination.

   (b) Write a MATLAB program called **backsub**, based on Algorithm 3.1.3,

   $[x] = \textbf{backsub}\ (U, b)$

   to solve a nonsingular upper triangular system

   $Ux = b$

   Use randomly generated test matrices and test matrices creating L and U with one or more small diagonal entries.

(*Note:* **forelm** and **backsub** are also in MATCOM or in Appendix B.)

Test Matrices for Problems 2 Through 8

For Problems 2 through 8, use the following matrices as test matrices. When the problem is the linear system problem $Ax = b$, create a vector b such that the solution vector x is a vector with all components equal to 1.

1. Hilbert matrix of order 10
2. Pie matrix of order 10
3. Hankel matrix of order 10
4. Randomly generated matrix of order 10
5. $A = \begin{pmatrix} 0.00001 & 1 & 1 \\ 0 & 0.00001 & 1 \\ 0 & 0 & 0.00001 \end{pmatrix}$
6. Vandermonde matrix of order 10

2. (a) Using **lugsel** (from MATCOM), **backsub**, and **forelm**, write the MATLAB program

   $[x] = \textbf{linsyswp}(A, b)$

   to solve $Ax = b$ using Gaussian elimination without pivoting.

   Compute the growth factor, elapsed time, and flop count for each system.

   (b) Run the program **inlu** from MATCOM and multiply the result by the vector b to obtain the solution vector $x = A^{-1}b$. Compute the flop count.

   (c) Compare the computed solutions and flop counts of (a) and (b).

3. (a) Using **parpiv** and **elmul** from Chapter 5 and **backsub**, write a MATLAB program

   $[x] = \textbf{linsyspp}(A, b)$

   to solve $Ax = b$ using Gaussian elimination with partial pivoting.

   Compute the growth factor, elapsed time, and flop count for each system.

   (b) Run the program **inparpiv** from MATCOM and multiply the result by b to compute the solution vector $x = A^{-1}b$. Compute the flop count for each system.

   (c) Compare the computed solutions, flop counts, and elapsed times of (a) and (b).

4. (a) Using **compiv** from MATCOM, **elmul** from Chapter 5, and **backsub**, write the MATLAB program

   $[x] = \textbf{linsyscp}(A, b)$

   to solve $Ax = b$ using Gaussian elimination with complete pivoting. Compute the flop count, elapsed time, and growth factor for each system.

   (b) Run the program **incompiv** from MATCOM and multiply the result by b to compute the solution vector $x = A^{-1}b$. Compute the flop count for each system.

   (c) Compare the computed solutions, flop counts, and elapsed time of (a) and (b).

5. (a) Implement Algorithm 6.4.2 to solve $Ax = b$ without explicit factorization using partial pivoting:

   $[x] = \textbf{linsyswf}(A, b)$

   (b) Compute A^{-1} using this explicit factorization.

6. (a) Using **housqr** from Chapter 5 (or the MATLAB function *qr*) and **backsub**, write the MATLAB program

   $[x] = \textbf{linsysqrh}(A, b)$

   to solve $Ax = b$ using QR factorization with Householder matrices. Compute the flop count for each system.

   (b) Repeat (a) with **givqr** in place of **housqr**; that is, write a MATLAB program called **linsysqrg** to solve $Ax = b$ using the Givens method for QR factorization.

7. (*The purpose of this exercise is to make a comparative study with respect to accuracy, elapsed time, flop count, and growth factor for different methods for solving* $Ax = b$.) Tabulate the result of Problems 2 through 6 as in Table 6.3. Make one table for each matrix. $\hat{x}$ stands for the computed solution.

8. (a) Write a MATLAB program to find the inverse of A using **housqr** (or MATLAB function *qr*) and **invuptr** (from MATCOM):

   $[A] = \textbf{invqrh}(A)$

   Compute the flop count for each matrix.

TABLE 6.3 Comparison of Different Methods for the Linear System Problems

Method	The Computed Solution $\hat{x}$	Rel. Error $\|\|x - \hat{x}\|\|/\|\|x\|\|$	Residual $\|\|b - A\hat{x}\|\|$	Growth Factor	Elapsed Time
linsyswp					
linsyspp					
linsyscp					
linsyswf					
linsysqrh					
linsysqrg					
$A^{-1}b$					

(b) Repeat (a) using **givqr** and **invuptr**:

$$[A] = \mathbf{invqrg}(A)$$

Compute the flop count for each matrix.

(c) Run **inlu**, **inparpiv**, **incompiv** from MATCOM with each of the data matrices. Make a table for each matrix A to compare the different methods for finding the inverse with respect to accuracy and flop count. Denote the computed inverse by $\hat{A}$. Get A^{-1} by using the MATLAB command $inv(A)$. See Table 6.4.

9. (a) Write a MATLAB program **chlgaus** to find the Cholesky factorization of a symmetric positive definite matrix A using Gaussian elimination without pivoting.

$$[H] = \mathbf{chlgaus}(A)$$

Create a 15×15 lower triangular matrix L with positive diagonal entries, taking some of the diagonal entries small enough to be very close to zero, multiply it by L^T, and take $A = LL^T$ as your test matrix. Compute H. Compute the flop count.

(b) Run the MATLAB program **Chol** on the same matrix in (a), and denote the transpose of the result by $\hat{H}$. Compute the flop count.

(c) Compare the results of (a) and (b). (Note that Chol(A) gives an upper triangular matrix H such that $A = H^T H$).

TABLE 6.4 Comparison of Different Methods for Computing the Inverse

Method	Relative Error $\|\|A^{-1} - (\hat{A})\|\|/\|\|A^{-1}\|\|$	Flop Count
inlu		
inparpiv		
incompiv		
invqrh		
invqrg		

10. Run the program **linsyswp** with the diagonally dominant, symmetric tridiagonal and block tridiagonal matrices encountered in Section 6.3.5 by choosing the right-hand-side vector b so that the solution vector x is known a priori. Compare the exact solution x with the computed solution $\hat{x}$. *(The purpose of this exercise is to verify that to solve a symmetric positive definite system, no pivoting is needed to ensure stability in Gaussian elimination.)*

11. (a) Write a MATLAB program to implement Algorithm 6.7.2 that finds an upper bound of the 2-norm of the inverse of an upper triangular matrix U:

$$[CEBOUND] = \mathbf{norminvtr}(U)$$

Test your result by randomly creating a 10×10 upper triangular matrix with several small diagonal entries, and then compare your result with that obtained by running the MATLAB command

$$\mathbf{norm}(\mathbf{inv}(U))$$

(b) Now compute the condition number of U as follows:

$$\mathbf{norm}(U) * \mathbf{norminvtr}(U)$$

Compare your result with that obtained by running the MATLAB command: **cond**(U). Verify that

$$Cond(U) \geq \frac{max(u_{ii})}{min(u_{ii})}$$

(Use the same test matrix U as in part (a)).

12. (a) *(The purpose of this exercise is to compare different approaches for estimating the condition number of a matrix).* Compute and/or estimate the condition number of each of the following matrices A of order 10: Hilbert, Pie, randomly generated, Vandermonde, and Hankel. Use the following approaches:

(i) Find the QR factorization of A with column pivoting: $Q^T AP = R$. Estimate the 2-norm of the inverse of R by running **norminvtr** on R. Now compute **norm**(R) * **norminvtr**(R). Compute the flop count.

(ii) Compute **norm**(A) * **norm**(**inv**(A)). Compute the flop count.

(iii) Compute an estimate of the norm-1 condition number of A using **hagcond1**.

(b) Compare the results and flop counts.

In Problems 13, 14, 15, and 18, create the vector b randomly.

13. (a) Run the iterative refinement program **iterref** from MATCOM on each of the following 15×15 systems: Hilbert, Pie, Vandermonde, randomly generated, and Hankel. Use the solution obtained from the program **linsyspp** (Problem 3) as the initial approximation $x^{(0)}$.

(b) Estimate the condition number of each of the matrices in (a) using the iterative refinement procedure.

(c) Compare your results on condition number estimation with those obtained in Problem 12.

14. Run the programs **jacobi**, **gaused**, and **sucov** from MATCOM on the 6×6 matrix A of Example 6.10.5 with the same starting vector, $x^{(0)} = (0, 0, \ldots, 0)^T$. Find how many iterations each method will take to converge. Verify the statement of the example that it takes 5 iterations for SOR to converge with ω_{opt}, compared with 18 iterations for Jacobi.
15. Run the programs **jacobi** and **gaused** from MATCOM on Example 6.10.3 and verify the statement of the example.
16. Run the program **nichol** from MATCOM implementing the "No-fill incomplete Cholesky factorization" on the tridiagonal symmetric positive definite matrix T of order 20 arising in the discretization of Poisson's equation. Compare your result with that obtained by running **chol(T)** on T.
17. Write a MATLAB program called **arnoldi** based on the Arnoldi method (Algorithm 6.10.8) using the modified Gram–Schmidt algorithm. (The modified Gram–Schmidt has been implemented in the MATCOM program **mdgrsch** (see Chapter 7).)
18. Using **arnoldi** and a suitable least-squares routine from Chapter 7, write a MATLAB program called **gmres** to implement the GMRES algorithm (Algorithm 6.10.9).

Suggestions for Further Reading

The books on numerical methods in the engineering literature routinely discuss how various engineering applications give rise to linear systems problems. We have used the following two in our discussions and found them useful:

Chapra, Steven C., and Raymond P. Canale. *Numerical Methods for Engineers*, 2nd ed. New York: McGraw-Hill, 1988.

O'Neil, Peter V. *Advanced Engineering Mathematics*, 3d ed. Belmont, Calif.: Wadsworth, 1991.

To know about basic concepts and results on ordinary and partial differential equations, any standard textbook on differential equations can be consulted. We particularly recommend the following.

Braun, M. *Differential Equations and Their Applications*, New York: Springer-Verlag, 1978.

Campbell, S. L. *An Introduction to Differential Equations and Their Applications*, 2nd ed. Belmont, Calif.: Wadsworth, 1990.

Weinberger, H. F. *A First Course in Partial Differential Equations*, New York: Wiley, 1965.

For techniques of discretization of partial differential equations using finite differences and finite elements, see the books by Strang (1986), Strang and Fix (1973), Strikwerda (1989), Reddy (1993), Celia and Gray (1992), Chapra and Canale (1988), and Ames (1992).

Direct methods (such as Gaussian elimination, QR factorization, etc.) for linear systems and related problems, discussions on perturbation analysis and conditioning

of the linear systems problems, iterative refinement, and so on, can be found in any standard numerical linear algebra text. (In particular, the books by Golub and Van Loan (1989) and by Stewart (1973) are highly recommended.) Most numerical analysis texts contain some discussions of these methods, but none of the existing books provides a thorough and in-depth treatment. For discussion on solutions of linear systems with special matrices, such as diagonally dominant, Hessenberg, and positive definite, see Wilkinson (1965, 218–220).

For proofs and analyses of backward error analyses of various algorithms, Wilkinson (1965) is the classic reference. Several recent papers by Nick Higham (1987; 1990) on condition number estimators are interesting to read.

For iterative methods, two (by now) almost classical books on the subject are essential reading:

Varga, Richard. *Matrix Iterative Analysis*, Englewood Cliffs, N.J.: Prentice Hall, 1962.

Young, David. *Iterative Solution of Large Linear Systems*, New York: Academic Press, 1971.

Another important book in the area is *Applied Iterative Methods*, by L. A. Hageman and D. M. Young (1981). The most recent book in the area is *Templates for the Solution of Linear Systems: Building Blocks For Iterative Methods*, by Richard Barrett and others (1994). The book incoporates state-of-the-art computational methods for solving large and sparse nonsymmetric systems. See also a recent excellent paper on the subject by Freund, Golub, and Nachtigal (1992) and the forthcoming books by Saad (1995) and Axelsson (1994).

The book *Introduction to Parallel and Vector Solutions of Linear Systems* and *Numerical Analysis: A Second Course* by James Ortega (1988a), also contain very clear expositions on the convergence of iterative methods.

The conjugate gradient method, originally developed by M. R. Hestenes and E. Stiefel (1952), has received considerable attention in the context of solving large and sparse positive definite linear systems. A considerable amount of work has been done by numerical linear algebraists and researchers in applications areas (such as in optimization).

The following books contain some in-depth discussion:

Luenberger, David G. *Introduction to Linear and Nonlinear Programming*, New York: Addison-Wesley, 1973.

Ortega, James. *Introduction to Parallel and Vector Solutions of Linear Systems*, New York: Plenum Press, 1988a.

Shewchuk, Jonathan R. *An Introduction to the Conjugate Gradient Method without the Agonizing Pain*, Technical Report CMU-CS-94-125 of Carnegie Mellon University, 1994.

An excellent survey paper on the subject by Golub and O'Leary (1989) is also recommended for further reading. Also see another survey paper by Axelsson (1985). For stability analysis of incomplete LU factorization schemes, see Elman (1986).

The interesting survey paper by Young, Jea, and Mai (1988), in the book *Linear Algebra in Signals, Systems, and Control*, is worth reading.

The development of conjugate gradient type methods for nonsymmetric and symmetric indefinite linear systems is an active area of research.

A nice discussion on scaling appears in Forsythe and Moler (1967, Chapter 11). See also a paper by Skeel (1979) for the relationship between scaling and stability of Gaussian elimination and a paper by Edelman and Stewart (1993) on scaling and orthogonality.

A recent paper of Chandrasekaran and Ipsen (1994b) describes sensitivity of individual components of the solution vector when the data is subject to perturbations.

To learn more about Krylov subspace methods such as GMRES, QMR, and conjugate gradient, see the papers by Bai, Hu, and Reichel (1994), Bramley and Sameh (1992), Saad (1981, 1984, 1987), Saad and Schultz (1986), Sorensen (1992), Freund (1993), Freund and Nachtigal (1991), Gutknecht (1992), Elman, Saad, and Saylor (1986), Greenbaum and Strakos (1992), Nachtigal, Reichel, and Trefethen (1992), Parlett (1980a, 1989), Pierce and Plemmons (1988), Saylor, Ashby, and Manteuffel (1990), Walker (1988), Woznicki (1993), Young, Jea, and Mai (1988) and the recent books by Saad (1992) and Barrett et al. (1994).

For applications of Krylov subspace methods to control problems, see Boley and Golub (1984), Choudhary (1994), Datta and Saad (1991), Hu and Reichel (1992), Hodel and Poola (1992), and Saad (1988, 1990).

For multisplitting of matrices and their applications to parallel solutions of linear systems, see the book by Ortega (1988a) and papers by O'Leary and White (1985), Neumann and Plemmons (1987), and White (1990).

For sparse matrix computations on MATLAB, see the recent paper of Gilbert, Moler, and Schreiber (1992). An earlier authoritative book on the subject of solving large sparse positive definite systems is the book by George and Liu (1981). A more recent book on direct methods for sparse matrix computations is the book by Duff, Erisman, and Reid (1986).

For condition estimation of sparse matrices, see Bischof, Lewis, and Pierce (1990), Ferng, Golub, and Plemmons (1991), and Pierce and Plemmons (1992).

For conditioning of structured matrices see Van Dooren (1991) and Gohberg and Koltracht (1993). For more on perturbations analysis and conditioning of linear system problems and various matrix factorizations, see the book by Stewart and Sun (1990) and various papers by Demmel (1987b, 1989), Edelman (1988, 1992), McCarthy and Strang (1973), Higham (1991), Stewart (1977b, 1993a), Sun (1991), Zha (1993), O'Leary (1980), and Drmac, Omladic, and Veselic (1994). See a recent paper of Eiermann and Varga (1993) for discussion on optimal relaxation factor for the SOR iteration method.

CHAPTER SEVEN

Least-Squares Solutions to Linear Systems

7.1 Introduction

In Chapter 6 we discussed several methods for solving the linear system

$$Ax = b$$

where A was assumed to be square and nonsingular. However, in several practical situations, such as in statistical applications, geometric modeling, and signal processing, we need to solve a system where the matrix A is nonsquare and/or singular. In such cases, solutions may not exist at all; in cases where there are solutions, there may be infinitely many. For example, when A is $m \times n$ and $m > n$, we have an *overdetermined* system (that is, the number of equations is greater than the number of unknowns), and an overdetermined system typically has no solution. In contrast, an *underdetermined* system ($m < n$) typically has an infinite number of solutions.

In these cases, the best we can hope for is to find a vector x that will make Ax as close as possible to the vector b. In other words, we seek a vector x such that $\|r(x)\| = \|Ax - b\|$ is minimized. When the Euclidean norm $\|\cdot\|_2$ is used, this solution is referred to as a **least-squares solution** to the system $Ax = b$. The term least-squares solution is justified, because it is a solution that minimizes the Euclidean norm of the residual vector, and, by definition, the square of the Euclidean norm of a vector is just the sum of squares of the components of the vector. The problem of finding least-squares solutions to the linear system $Ax = b$ is known as the **linear least-squares problem** (LSP). The linear least-squares problem is formally defined as follows.

Statement of the Linear Least-Squares Problem
Given a real $m \times n$ matrix A and a real vector b, find a real n-vector x such that the function $\|r(x)\| = \|Ax - b\|_2$ is minimized.

If the least-squares problem has more than one solution, the one having the minimum Euclidean norm is called the **minimum length solution**, or the **minimum norm solution**.

This chapter is devoted to the study of such problems. The organization of the chapter is as follows. In Section 7.2 we show how a very simple business application leads to an overdetermined least-squares problem. In this section we simply formulate the problem as a least-squares problem; in Section 7.8 we present a solution of the problem using normal equations.

In Section 7.3 we prove a theorem on the existence and uniqueness of solution of an overdetermined least-squares problem. Some applications giving rise to overdetermined systems are discussed in Sections 7.2 and 7.5. In Section 7.7 we analyze the sensitivity of the least-squares problems due to perturbations in data. We prove only a simple result here and state other results without proofs. In Section 7.4 we give a geometrical interpretation of the least-squares problem. In Section 7.6 we discuss the relationship between the least-squares problem and the pseudoinverse of a matrix.

Section 7.8 deals with computational methods for both full-rank and rank-deficient overdetermined problems. We discuss the normal equations method and the QR factorization methods using Householder transformations, modified Gram–Schmidt and classical Gram–Schmidt orthogonalizations.

Underdetermined systems problems are considered in Section 7.9. We again discuss here the normal equations and the QR methods for an underdetermined problem. In Section 7.10 an iterative improvement procedure for refining approximate solutions is presented. In Section 7.11 we describe an efficient way of computing the variance–covariance matrix of the least-squares solution, which is $(A^TA)^{-1}$.

The following major tools and concepts developed earlier will be needed for smooth learning and comprehension of this chapter's material: the Cholesky factorization algorithm (Algorithm 6.4.4), the QR factorization algorithms using Householder and Givens matrices (Algorithm 5.4.3, 5.5.4), QR factorization of a nonsquare matrix (Section 5.4), computations of orthonormal bases and orthogonal projections (Section 5.6), QR factorization with column pivoting (Section 5.7), iterative refinement algorithm (Algorithm 6.9.1), and perturbation analysis results of linear systems problems (Sections 6.6 and 6.7).

7.2 A Simple Application Leading to an Overdetermined System

Suppose that the number of units b_i of a product sold by a company in the district i of a town depends upon the population a_{i1} (in thousands) of the district and the per capita income a_{i2} (in thousands of dollars). The following table (taken from Neter, Wasserman, and Kunter (1983)) compiled by the company shows the sales in five districts, as well as the corresponding population and per capita income.

District i	Sales b_i	Population a_{i1}	Per Capita Income a_{i2}
1	162	274	2450
2	120	180	3254
3	223	375	3802
4	131	205	2838
5	67	86	2347

Suppose the company wants to use this table to predict future sales and believes (from past experience) that the following relationship between b_i, a_{i1}, and a_{i2} is appropriate:

$$b_i = x_1 + a_{i1}x_2 + a_{i2}x_3$$

If the data in the table have satisfied this relation, we have

$$\begin{aligned} 162 &= x_1 + 274x_2 + 2450x_3 \\ 120 &= x_1 + 180x_2 + 3254x_3 \\ 223 &= x_1 + 375x_2 + 3802x_3 \\ 131 &= x_1 + 205x_2 + 2838x_3 \\ 67 &= x_1 + 86x_2 + 2347x_3 \end{aligned}$$

or

$$Ax = b$$

where

$$A = \begin{pmatrix} 1 & 274 & 2450 \\ 1 & 180 & 3254 \\ 1 & 375 & 3802 \\ 1 & 205 & 2838 \\ 1 & 86 & 2347 \end{pmatrix}, \quad b = \begin{pmatrix} 162 \\ 120 \\ 223 \\ 131 \\ 67 \end{pmatrix}, \quad x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

This system is an overdetermined system of five equations in three unknowns.

7.3 Existence and Uniqueness

As in the case of solving a linear system, questions naturally arise:

1. Does a least-squares solution to $Ax = b$ always exist?
2. If the solution exists, is it unique?
3. How do we obtain such solutions?

The following theorem answers the questions of existence and uniqueness. *Assume that the system $Ax = b$ is overdetermined, or square—that is, A is of order $m \times n$, where $m \geq n$.* An overdetermined system $Ax = b$ can be represented graphically as shown in Figure 7.1.

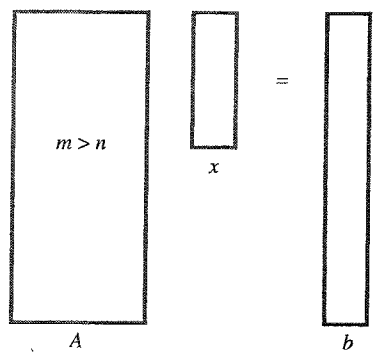


FIGURE 7.1

The underdetermined case will be discussed later in this chapter.

THEOREM 7.3.1 Least-Squares Existence and Uniqueness Theorem There always exists a solution to the linear least-squares problem. This solution is unique if and only if A has full rank; that is, $\text{rank}(A) = n$. If A is rank deficient, then the least-squares problem has infinitely many solutions.

We present a proof here in the full-rank case—that is, in the case when A has full rank. The rank-deficient case is treated later in this chapter and in Chapter 10.

First, we observe the following.

LEMMA 7.3.1 x is a least-squares solution of the system $Ax = b$ if and only if x satisfies

$$A^T Ax = A^T b \tag{7.3.1}$$

Proof: We denote the residual $r = b - Ax$ by $r(x)$ to emphasize that given A and b, r is a function of x. Let y be an n-vector. Then $r(y) = b - Ay = r(x) + Ax - Ay = r(x) + A(x - y)$. So,

$$\|r(y)\|_2^2 = \|r(x)\|_2^2 + 2(x - y)^T A^T r(x) + \|A(x - y)\|_2^2$$

First assume that x satisfies

$$A^T Ax = A^T b$$

that is, $A^T r(x) = 0$. Then from the preceding, we have

$$\|r(y)\|_2^2 = \|r(x)\|_2^2 + \|A(x - y)\|_2^2 \geq \|r(x)\|^2$$

implying that x is a least-squares solution.

Next assume that $A^T r(x) \neq 0$. Set $A^T r(x) = z \neq 0$. Define a vector y such that

$$y = x + cz$$

Then

$$\begin{aligned} r(y) &= r(x) + A(x - y) = r(x) - cAz \\ \|r(y)\|_2^2 &= \|r(x)\|_2^2 + c^2\|Az\|_2^2 - 2cz^T A^T r(x) \\ &= \|r(x)\|_2^2 + c^2\|Az\|_2^2 - 2c\|z\|_2^2 < \|r(x)\|_2^2 \end{aligned}$$

for sufficiently small values of c. This implies that x is not a least-squares solution. ■

Proof of Theorem 7.3.1 (in the full-rank case): We know from Lemma 7.3.1 that x is a least-squares solution of $Ax = b$ if and only if x satisfies

$$A^T Ax = A^T b$$

We now show that in the full-rank case, this solution is unique.

We know from Theorem 6.2.1 that the solution of the system $A^T Ax = A^T b$ is unique if and only if $A^T A$ is nonsingular. So, all we need to do is show that $A^T A$ is nonsingular if and only if A has full rank. To this end, we will prove a stronger result that *$A^T A$ is positive definite if and only if A has full rank.*

If A has full rank, then $x \neq 0$ implies that $y = Ax \neq 0$. Thus, $x^T A^T Ax = y^T y > 0$, implying that $A^T A$ is positive definite. On the other hand, if A does not have full rank, then, for some $x \neq 0$, we will have $Ax = 0$, and, therefore, $x^T A^T Ax = 0$, implying that $A^T A$ is not positive definite.

7.4 Geometric Interpretation of the Least-Squares Problem

Let A be an $m \times n$ matrix with $m > n$. Then A is a linear mapping of $\mathbb{R}^n \to \mathbb{R}^m$. $R(A)$ is a subspace of $\mathbb{R}^m$. Every vector $u \in R(A)$ can be written as $u = Ax$ for some $x \in \mathbb{R}^n$. Let $b \in \mathbb{R}^m$. Because $\|\cdot\|_2$ is a Euclidean norm, $\|b - Ax\|_2$ is the distance between the endpoints of b and Ax. It is clear that this distance is minimal if and only if $b - Ax$ is perpendicular to $R(A)$ (see Figure 7.2). In that case, $\|b - Ax\|_2$ is the distance from the endpoint of b to the "plane" $R(A)$.

From this interpretation, it is easy to understand that a solution of the least-squares problem to the linear system $Ax = b$ always exists. This solution exists because we can project b to the "plane" $R(A)$ to obtain a vector $u \in R(A)$, and there is $x \in \mathbb{R}^n$ such that $u = Ax$. This x is a solution.

Because $b - Ax$ is perpendicular to $R(A)$ and every vector in $R(A)$ is a linear combination of column vectors of A, then $b - Ax$ is orthogonal to every column of A; that is,

$$A^T(b - Ax) = 0$$

or

$$A^T Ax = A^T b$$

7.5 Normal Equations and Polynomial Fitting

DEFINITION 7.5.1 The system of equations

$$A^T Ax = A^T b \tag{7.5.1}$$

is called the **normal equations**.

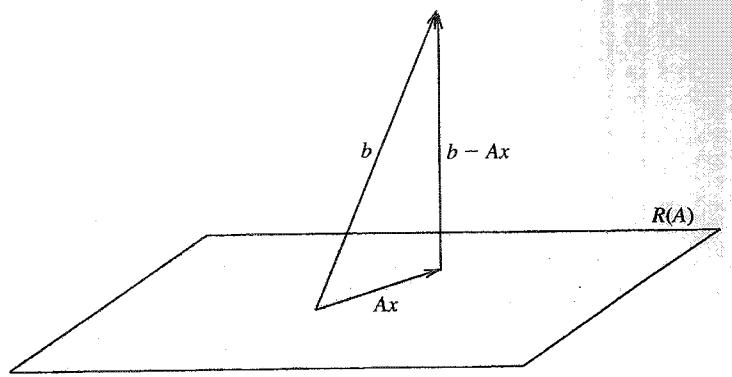


FIGURE 7.2

A well-known example of how normal equations arise is the fitting of a polynomial to a set of experimental data.

The engineers and scientists gather data from experiments. A meaningful representation of the collected data is needed to make meaningful decisions for the future.

Let $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ be a set of paired observations. Suppose that the mth-degree ($m \leq n$) polynomial

$$y(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_m x^m \tag{7.5.2}$$

is the "best fit" for this set of data. One strategy for best fit is to minimize the sum of the squares of the residuals

$$E = \sum_{i=1}^{n} (y_i - a_0 - a_1 x_i - a_2 x_i^2 - \cdots - a_m x_i^m)^2 \tag{7.5.3}$$

We must then have

$$\frac{\partial E}{\partial a_i} = 0, \qquad i = 1, \ldots, m$$

Now,

$$\begin{aligned}
\frac{\partial E}{\partial a_0} &= -2 \sum_{i=1}^{n} (y_i - a_0 - a_1 x_i - a_2 x_i^2 - \cdots - a_m x_i^m) \\
\frac{\partial E}{\partial a_1} &= -2 \sum_{i=1}^{n} x_i (y_i - a_0 - a_1 x_i - a_2 x_i^2 - \cdots - a_m x_i^m) \\
&\vdots \\
\frac{\partial E}{\partial a_m} &= -2 \sum_{i=1}^{n} x_i^m (y_i - a_0 - a_1 x_i - \cdots - a_m x_i^m)
\end{aligned} \tag{7.5.4}$$

Setting these equations to zero, we have

$$\begin{aligned}
a_0 n + a_1 \sum x_i + a_2 \sum x_i^2 + \cdots + a_m \sum x_i^m &= \sum y_i \\
a_0 \sum x_i + a_1 \sum x_i^2 + \cdots + a_m \sum x_i^{m+1} &= \sum x_i y_i \\
&\vdots \\
a_0 \sum x_i^m + a_1 \sum x_i^{m+1} + \cdots + a_m \sum x_i^{2m} &= \sum x_i^m y_i
\end{aligned} \tag{7.5.5}$$

(where $\sum$ denotes the summation from $i = 1$ to n).

Setting $\sum x_i^k = S_k$, $k = 0, 1, \ldots, 2m$, and denoting the entries of the right-hand side, respectively, by $b_0, b_1, \ldots, b_m$, the system of equations can be written as

$$\begin{pmatrix} S_0 & S_1 & \cdots & S_m \\ S_1 & S_2 & \cdots & S_{m+1} \\ \vdots & \vdots & & \vdots \\ S_m & S_{m+1} & \cdots & S_{2m} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_m \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_m \end{pmatrix} \tag{7.5.6}$$

(Note that $S_0 = n$.) This system is a system of $(m + 1)$ equations in $(m + 1)$ unknowns, $a_0, a_1, \ldots, a_m$.

This system is really a system of normal equations. To see this, define

$$V = \begin{pmatrix} 1 & x_1 & \cdots & x_1^m \\ 1 & x_2 & \cdots & x_2^m \\ \vdots & \vdots & & \vdots \\ 1 & x_n & \cdots & x_n^m \end{pmatrix} \tag{7.5.7}$$

Then this system is equal to

$$V^T V a = b = V^T y \tag{7.5.8}$$

where

$$y = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}$$

This is a system of normal equations; furthermore, if the x_i's are all distinct, then the matrix V has full rank.

The matrix V is known as the **Vandermonde matrix**. From our discussion in the previous section, we see that a is the least-squares solution to the system (7.5.8).

Example 7.5.1 Suppose that an electrical engineer has gathered the following experimental data, consisting of the measurement of the current in an electric wire for various voltages.

x = voltage	0	2	5	7	9	13	24
y = current	0	6	7.9	8.5	12	21.5	35

We would like to derive the normal equations for these data corresponding to the best fit of the data to (a) a straight line and (b) a quadratic and would like to see a comparison of the predicted results with the actual result when $x = 5$.

(a) Straight-line fit:

$$V = \begin{pmatrix} 1 & 0 \\ 1 & 2 \\ 1 & 5 \\ 1 & 7 \\ 1 & 9 \\ 1 & 13 \\ 1 & 24 \end{pmatrix}, \qquad y = \begin{pmatrix} 0 \\ 6 \\ 7.9 \\ 8.5 \\ 12.0 \\ 21.5 \\ 35.0 \end{pmatrix}$$

The normal equations are:

$$V^T V a = V^T y = b$$

$$\begin{pmatrix} 7 & 60 \\ 60 & 904 \end{pmatrix} a = 10^3 \begin{pmatrix} 0.0906 \\ 1.3385 \end{pmatrix}$$

The solutions of these equations are

$$a_0 = 0.6831$$
$$a_1 = 1.4353$$

The value of $a_0 + a_1 x$ at $x = 5$ is $0.6831 + 1.4345 \times 5 = 7.8596$.

(b) Quadratic fit:

$$V = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 2 & 4 \\ 1 & 5 & 25 \\ 1 & 7 & 49 \\ 1 & 9 & 81 \\ 1 & 13 & 169 \\ 1 & 24 & 576 \end{pmatrix}$$

The normal equations are

$$V^T V a = V^T y = b$$

$$\begin{pmatrix} 7 & 60 & 904 \\ 60 & 904 & 17{,}226 \\ 904 & 17{,}226 & 369{,}940 \end{pmatrix} a = 10^4 \begin{pmatrix} 0.0091 \\ 0.1338 \\ 2.5404 \end{pmatrix}$$

The solution of these normal equations is

$$a = \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} 0.8977 \\ 1.3695 \\ 0.0027 \end{pmatrix}$$

The value of $a_0 + a_1 x + a_2 x^2$ at $x = 5$ is 7.8127.

Remarks: The set of polynomials of degree less than or equal to m forms a vector space of dimension $(m + 1)$. Thus, if $\phi_0, \phi_1, \ldots, \phi_m$ is a basis of this vector-space, then every polynomial $y(x)$ in this space can be written as

$$y(x) = \sum_{j=0}^{m} a_j \phi_i(x)$$

If the basis functions $\phi_i(x)$ are chosen to be $\phi_0(x) = 1$, $\phi_1(x) = x, \ldots, \phi_m(x) = x^m$, then we get $y(x) = a_0 + a_1 x + \cdots + a_m x^m$, as used in our illustration above. Such a choice of the basis functions is, in general, not used in practice; the method of choice is to generate a set of orthogonal polynomials. The least we should do is make a shift so that the basis functions are centered around the midpoint of the interval. For example, if we use a straight line fitting in Example 7.5.1, the choice of $\phi_i(x)$, $i = 0, 1$ could be $\phi_0(x) = 1$, $\phi_1(x) = \frac{1}{7}(t - 7)$.

With this choice, the matrix V will be

$$V = \begin{pmatrix} 1 & -1 \\ 1 & -0.7143 \\ 1 & -0.2857 \\ 1 & 0 \\ 1 & 0.2857 \\ 1 & 0.8571 \\ 1 & 2.4286 \end{pmatrix}$$

Note that the condition number of this new Vandermonde matrix is 1.2537, whereas the condition number of the original Vandermonde matrix is 17.3845.

7.6 Pseudoinverse and the Least-Squares Problem

DEFINITION 7.6.1 The matrix $A^\dagger = (A^T A)^{-1} A^T$, when A is $m \times n$ $(m \geq n)$ and has rank n, *or* the matrix $A^\dagger = A^T (A A^T)^{-1}$, when A is $m \times n$ $(m < n)$ and has rank m, is called the **pseudoinverse** of A. The pseudoinverse is also referred to as the **Moore–Penrose generalized inverse of** A.

From (7.3.1) it therefore follows that the unique least-squares solution for the overdetermined problem is $x = A^\dagger b$.

> **Least-Squares Solution Using the Pseudoinverse**
> The unique least-squares solution x to the full-rank overdetermined least-squares problem $Ax = b$ is given by
>
> $$x = (A^T A)^{-1} A^T b = A^\dagger b$$

Clearly, this definition of the pseudoinverse generalizes the ordinary definition of the inverse of a square matrix A. Note that when A is square and invertible,

$$A^\dagger = (A^T A)^{-1} A^T = A^{-1} (A^T)^{-1} A^T = A^{-1}$$

We shall discuss this important concept in some more detail in Chapter 10. An excellent reference on the subject is the book *Generalized Inverses of Matrices and Its Applications* by C. R. Rao and S. K. Mitra (1971). Having defined the generalized inverse of a rectangular matrix, we now define the condition number of a matrix as $\text{Cond}(A) = \|A\| \, \|A^\dagger\|$.

DEFINITION 7.6.2 If an $m \times n$ matrix A has full rank, then $\text{Cond}(A) = \|A\| \|A^\dagger\|$.

Note: If not explicitly stated, all the norms used in the rest of this chapter are 2-norms, and $\text{Cond}(A)$ is the condition number with respect to 2-norm; that is, $\text{Cond}(A) = \|A\|_2 \, \|A^\dagger\|_2$.

Example 7.6.1

$$A = \begin{pmatrix} 1 & 2 \\ 2 & 3 \\ 4 & 5 \end{pmatrix}, \qquad \text{rank}(A) = 2$$

Thus, A has full rank.

$$A^\dagger = (A^T A)^{-1} A^T = \begin{pmatrix} -1.2857 & -0.5714 & 0.8571 \\ 1 & 0.5000 & -0.5000 \end{pmatrix}$$

$$\text{Cond}_2(A) = \|A\|_2 \, \|A^\dagger\|_2 = 7.6656 \times 2.0487 = 15.7047$$

Variance–Covariance Matrix

In certain applications, the matrix A^TA is called the *information matrix*, because it measures the information contained in the experiment, and the matrix $(A^TA)^{-1}$ is known as the **variance–covariance matrix**. An algorithm for computing the variance–covariance matrix without explicitly computing the inverse is given in Section 7.11.

7.7 Sensitivity of the Least-Squares Problem

In this section we study the sensitivity of a least-squares solution to perturbations in data; that is, we investigate how a least-squares solution changes with respect to small changes in the data. This study is important in understanding the different behaviors of different methods for solving the least-squares problems that are discussed in the next section. We consider two cases: perturbation in the vector b and perturbation in the matrix A. The results in this section are normwise perturbation results. For componentwise perturbation results, see Björck (1994b), and Chandrasekaran and Ipsen (1994b).

Case 1: Perturbation in the Vector b

Here we assume that the vector b has been perturbed to $\tilde{b} = b + \delta b$, but A has remained unchanged.

THEOREM 7.7.1 Least-Squares Right Perturbation Theorem Let x and $\hat{x}$, respectively, be the unique least-squares solutions to the original and the perturbed problems. Then if $\|b_R\| \neq 0$,

$$\frac{\|\hat{x} - x\|}{\|x\|} \leq \text{Cond}(A)\frac{\|\delta b_R\|}{\|b_R\|}$$

Here

$$\text{Cond}(A) = \|A\|\|A^\dagger\|$$

and b_R and δb_R are, respectively, the projections of the vectors b and δb onto $R(A)$.

Proof: Because x and $\hat{x}$ are the unique least-squares solutions to the original and the perturbed problems, we have

$$x = A^\dagger b$$
$$\hat{x} = A^\dagger(b + \delta b)$$

Thus,

$$\hat{x} - x = A^\dagger b + A^\dagger \delta b - A^\dagger b = A^\dagger \delta b$$

Let δb_N denote the projection of δb onto the orthogonal complement of $R(A)$; that is,

$$\delta b = \delta b_R + \delta b_N$$

Since δb_N lies in the orthogonal complement of $R(A) = N(A^T)$, we have $A^T(\delta b_N) =$

0. So

$$\begin{aligned}\hat{x} - x &= A^\dagger \delta b = A^\dagger(\delta b_R + \delta b_N)\\ &= A^\dagger(\delta b_R) + A^\dagger(\delta b_N) = A^\dagger \delta b_R + (A^TA)^{-1}A^T\delta b_N = A^\dagger \delta b_R \qquad (7.7.1)\end{aligned}$$

Again, because x is the unique least-squares solution, we have

$$Ax = b_R$$

from which (again taking norms on both sides) we get

$$\|x\| \geq \frac{\|b_R\|}{\|A\|} \qquad (7.7.2)$$

Taking norms on both sides of (7.7.1) and combining with (7.7.2), we have the theorem. ■

Interpretation of Theorem 7.7.1. Theorem 7.7.1 tells us that *if only the vector b is perturbed, then, as in the case of linear systems,* $\text{Cond}(A) = \|A\|\|A^\dagger\|$ *serves as the condition number in the sensitivity analysis of the unique least-squares solution.* If this number is large, then even with a small relative error in the projection of b onto $R(A)$, we might have a drastic change in the least-squares solution. On the other hand, if this number is small and the relative error in the projection of b onto $R(A)$ is also small, then the least-squares solution will not change much. *Note that it is the smallness of the relative error in the projection of b onto $R(A)$, namely $\|\delta b_R\|/\|b_R\|$, that plays the role here, not merely the smallness of $\|\delta b_R\|$.*

Example 7.7.1 An Insensitive Problem

$$A = \begin{pmatrix} 1 & 2 \\ 0 & 1 \\ 1 & 0 \end{pmatrix}, \qquad b = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \qquad \delta b = 10^{-4}\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

$$P_A = \text{projection of } A \text{ onto } R(A)$$
$$= \begin{pmatrix} 0.8333 & 0.3333 & 0.1667 \\ 0.3333 & 0.3333 & -0.3333 \\ 0.1667 & -0.3333 & 0.8333 \end{pmatrix}$$

(See Example 5.6.2.)

$$b_R = P_A b = \begin{pmatrix} 1.3333 \\ 0.3333 \\ 0.6667 \end{pmatrix}$$

$$\delta b_R = P_A \delta b = 10^{-3}\begin{pmatrix} 0.13333 \\ 0.03333 \\ 0.06667 \end{pmatrix}$$

$$\text{Cond}(A) = 2.4495$$

Because $\|\delta b_R\|/\|b_R\| = 10^{-4}$, a small number, and A is well conditioned, from Theorem 7.7.1, we expect that the least-squares solution will not be perturbed much.

The following computations show this is, indeed, the case.

$$x = A^{\dagger}b = \begin{pmatrix} 0.6667 \\ 0.3333 \end{pmatrix}$$

$$\hat{x} = A^{\dagger}(b + \delta b) = \begin{pmatrix} 0.6667 \\ 0.3334 \end{pmatrix}$$

$$\frac{\|\hat{x} - x\|}{\|x\|} = 10^{-4}$$

Example 7.7.2 A Sensitive Problem

$$A = \begin{pmatrix} 1 & 1 \\ 10^{-4} & 0 \\ 0 & 10^{-4} \end{pmatrix}, \qquad b = \begin{pmatrix} 2 \\ 10^{-4} \\ 10^{-4} \end{pmatrix}$$

$$\delta b = 10^{-3}\begin{pmatrix} 1 \\ 0.1 \\ 0 \end{pmatrix}, \qquad \text{Cond}(A) = O(10^4)$$

$$x = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \qquad \hat{x} = \begin{pmatrix} 1.5005 \\ 0.5005 \end{pmatrix}$$

Case 2: Perturbation in the Matrix *A*

The analysis here is much more complicated than the previous case. We state the result (without proof) and the major consequences of the result. Let the perturbation E of the matrix be small enough that

$$\text{rank}(A) = \text{rank}(A + E)$$

Let x and $\hat{x}$ denote the unique least-squares solutions, respectively, to the original and to the perturbed problem. Let E_A and E_N denote the projections of E onto $R(A)$ and onto the orthogonal complement of $R(A)$, respectively. Then if $b_R \neq 0$, we have the following theorem due to Stewart (1973, 223).

THEOREM 7.7.2 Least-Squares Left Perturbation Theorem Let x and $\hat{x}$ be the unique least-squares solutions to $Ax = b$ and $(A + E)\hat{x} = b$, and let $\text{rank}(A + E)$ be the same as $\text{rank}(A)$. Then

$$\frac{\|\hat{x} - x\|}{\|x\|} \leq 2\,\text{Cond}(A)\frac{\|E_A\|}{\|A\|} + 4(\text{Cond}(A))^2\frac{\|E_N\|}{\|A\|}\frac{\|b_N\|}{\|b_R\|} + O\left(\frac{\|E_N\|}{\|A\|}\right)^2$$

Interpretation of Theorem 7.7.2. Theorem 7.7.2 tells us that *in the case where only the matrix A is perturbed, the sensitivity of the unique least-squares solution, in general, depends upon the squares of the condition number of A. However, if* $\|E_N\|$ *or* $\|b_N\|$ *is zero, then the sensitivity will depend only on* Cond(A). Note that the residual $r = b - Ax$ is zero if $b_N = 0$.

Two Examples with Different Sensitivities

We now present two examples with the same matrix A but with different b to illustrate the different sensitivities of the least-squares problem in different cases. In the first

example, $(\text{Cond}(A))^2$ serves as the condition number of the problem; in the second example, Cond(A) serves as the condition number.

Example 7.7.3 Sensitivity Depending Upon the Square of the Condition Number

$$A = \begin{pmatrix} 1 & 1 \\ 0.0001 & 0 \\ 0 & 0.0001 \end{pmatrix}, \qquad b = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

$$b_R = P_A b = \begin{pmatrix} 1.0001 \\ 0.0001 \\ 0.0001 \end{pmatrix}, \qquad b_N = \begin{pmatrix} -0.0001 \\ 0.9999 \\ 0.9999 \end{pmatrix}, \qquad \frac{\|b_N\|}{\|b_R\|} = 1.4140$$

(using P_A from Example 5.6.3). Let

$$E = 10^{-4}\begin{pmatrix} 0 & -0.0001 \\ 0 & 0.9999 \\ 0 & 0.9999 \end{pmatrix}$$

$$A + E = \begin{pmatrix} 1 & 1 \\ 0.0001 & 0.0001 \\ 0 & 0.0002 \end{pmatrix}, \qquad E_N = 10^{-4}\begin{pmatrix} 0 & -0.0002 \\ 0 & 0.9999 \\ 0 & 0.9999 \end{pmatrix}$$

$$\frac{\|E_N\|}{\|A\|} = \frac{\|E\|}{\|A\|} = 9.999 \times 10^{-5}$$

Although the product of $\|E_N\| / \|A\|$ and $\|b_N\| / \|b_R\|$ is rather small, since $(\text{Cond}(A))^2 = 2 \times 10^8$ is large, we should expect a drastic departure of the perturbed solution from the original solution. This is indeed true, as the following computations show:

$$\hat{x} = 10^3\begin{pmatrix} -4.999 \\ 5 \end{pmatrix}, \qquad x = \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}$$

$$\frac{\|\hat{x} - x\|}{\|x\|} = 9.999 \times 10^3 \qquad \text{(large)}$$

Example 7.7.4 Sensitivity Depending Upon the Condition Number Let

$$E = 10^{-4}\begin{pmatrix} 0 & -0.0001 \\ 0 & 0.9999 \\ 0 & 0.9999 \end{pmatrix} \qquad \text{(same as in Example 7.7.3)}$$

$$A = \begin{pmatrix} 1 & 1 \\ 0.0001 & 0 \\ 0 & 0.0001 \end{pmatrix}, \qquad b = \begin{pmatrix} 2 \\ 0.0001 \\ 0.0001 \end{pmatrix}$$

In this case, $b_R = b$ and

$$b_N = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

(See Example 5.6.3.)

Thus, according to Theorem 7.7.2, the square of Cond(A) does not have any effect; the least-squares solution is affected only by Cond(A). We verify this as follows:

$$x = A^{\dagger} b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

$$\hat{x} = \begin{pmatrix} 1.4999 \\ 0.5000 \end{pmatrix}$$

$$\text{Cond}(A) = 1.4142 \times 10^4$$

$$\frac{\|\hat{x} - x\|}{\|x\|} = 0.5000$$

Residual Sensitivity. We have just seen that the sensitivities of the least-squares solutions due to perturbations in the matrix A are different for different right-hand side vectors b; however, the following theorem shows that the *residual sensitivity always depends upon the condition number of the matrix A*. We state the result in somewhat simplified and crude form. See Golub and Van Loan, (1989, 228–30) for a precise statement and proof of a result on residual sensitivity.

THEOREM 7.7.3 Least-Squares Residual Sensitivity Theorem Let r and $\hat{r}$ denote the residuals, respectively, for the original and the perturbed least-squares problems; that is,

$$r = b - Ax$$

$$\hat{r} = b - (A + E)(\hat{x})$$

Then

$$\frac{\|\hat{r} - r\|}{\|b\|} \le \epsilon(1 + 2\,\text{Cond}(A)) + O(\epsilon^2)$$

where

$$\epsilon = \frac{\|E\|}{\|A\|}$$

Interpretation of Theorem 7.7.3. This result tells us that the sensitivity of the residual depends at most on the condition number of A, when A is perturbed.

Example 7.7.5

$$A = \begin{pmatrix} 1 & 1 \\ 10^{-4} & 0 \\ 0 & 10^{-4} \end{pmatrix}, \qquad b = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

$$E = 10^{-4} \begin{pmatrix} 0 & -0.0001 \\ 0 & 0.9999 \\ 0 & 0.9999 \end{pmatrix}$$

$$x = \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}, \quad \hat{x} = 10^3 \begin{pmatrix} -4.9999 \\ 5 \end{pmatrix}$$

$$r = b - Ax = \begin{pmatrix} -0.0001 \\ 0.9999 \\ 0.9999 \end{pmatrix}$$

$$\hat{r} = \begin{pmatrix} -0.0001 \\ 0.9999 \\ 0 \end{pmatrix}$$

$$\frac{\|\hat{r} - r\|}{\|b\|} = 0.5773,$$

$$\text{Cond}(A) = 1.4142 \times 10^4$$

and

$$\text{Cond}(A) \cdot \frac{\|E\|}{\|A\|} = 1.4142$$

The inequality in Theorem 7.7.3 is now easily verified.

Sensitivity of the Pseudoinverse. The following result, due to Wedin (1973), shows that it is Cond(A) again that plays a role in the sensitivity analysis of the pseudoinverse of a matrix.

THEOREM 7.7.4 Pseudoinverse Sensitivity Theorem Let A be $m \times n$, where $m \ge n$. Let $A^{\dagger}$ and $\tilde{A}^{\dagger}$ be, respectively, the pseudoinverse of A and of $\tilde{A} = A + E$. Then, provided that rank(A) = rank($\tilde{A}$), we have

$$\frac{\|\tilde{A}^{\dagger} - A^{\dagger}\|}{\|\tilde{A}^{\dagger}\|} \le c\,\text{Cond}(A)\frac{\|E\|}{\|A\|}$$

where c is a small number.

Example 7.7.6

$$A = \begin{pmatrix} 1 & 2 \\ 2 & 3 \\ 4 & 5 \end{pmatrix}, \qquad E = 10^{-3}A = \begin{pmatrix} 0.0010 & 0.0020 \\ 0.0020 & 0.0030 \\ 0.0040 & 0.0050 \end{pmatrix}$$

$$A^{\dagger} = \begin{pmatrix} -1.2857 & -0.5714 & 0.8571 \\ 1 & 0.5000 & -0.5000 \end{pmatrix}$$

$$A + E = \tilde{A} = \begin{pmatrix} 1.001 & 2.002 \\ 2.002 & 3.003 \\ 4.004 & 5.005 \end{pmatrix}$$

$$\tilde{A}^{\dagger} = \begin{pmatrix} -1.2844 & -0.5709 & 0.8563 \\ 0.9990 & 0.4995 & -0.4995 \end{pmatrix}$$

$$\frac{\|\tilde{A}^{\dagger} - A^{\dagger}\|}{\|\tilde{A}^{\dagger}\|} = 10^{-3} \approx \frac{\|E\|}{\|A\|}$$

Note that

$$\text{Cond}(A) = 15.7047$$

7.8 Computational Methods for Overdetermined Least-Squares Problems

7.8.1 The Normal Equations Method

One of the most widely used approaches (especially in statistics) for computing the least-squares solution is the *normal equations method*. This method is based upon the solution of the system of normal equations

$$A^T A x = A^T b$$

We assume that A *is* $m \times n$ $(m > n)$ *and that it has full rank*. Because in this case $A^T A$ is symmetric and positive definite, it admits the Cholesky decomposition

$$A^T A = HH^T$$

Therefore, the normal equations approach for solving the least-squares problem can be stated as follows.

ALGORITHM 7.8.1 Least-Squares Solution Using Normal Equations Given an $m \times n$ $(m > n)$ matrix A with $\text{rank}(A) = n$, the following algorithm computes the least-squares solution x from normal equations using Cholesky factorization.

Step 1: Form $c = A^T b$.

Step 2: Compute the Cholesky factor H of $A^T A$, by taking advantage of the symmetry of $A^T A$.

Step 3: Solve the triangular systems in the sequence

$$Hy = c$$
$$H^T x = y$$

Flop count. This method for solving the full-rank least-squares problem requires about $mn^2/2 + n^3/6$ flops: $mn^2/2$ for computing $A^T A$ and $A^T b$, $n^3/6$ for computing the Cholesky factorization of $A^T A$, and n^2 to solve two triangular systems. Thus, the method is quite efficient.

MATCOM Note: Algorithm 7.8.1 has been implemented in MATCOM program LSFRNME.

Example 7.8.1 Normal Equations Solution of the Case Study Problem

$$A = \begin{pmatrix} 1 & 274 & 2450 \\ 1 & 180 & 3254 \\ 1 & 375 & 3802 \\ 1 & 205 & 2838 \\ 1 & 86 & 2347 \end{pmatrix}, \qquad b = \begin{pmatrix} 162 \\ 120 \\ 223 \\ 131 \\ 67 \end{pmatrix}$$

Step 1: Form $A^T A = \begin{pmatrix} 5 & 1120 & 14{,}691 \\ 1120 & 297{,}522 & 3{,}466{,}402 \\ 14{,}691 & 3{,}466{,}402 & 44{,}608{,}873 \end{pmatrix}$

Step 2: Form $A^T b = \begin{pmatrix} 703 \\ 182{,}230 \\ 2{,}164{,}253 \end{pmatrix}$

Step 3: Solve the normal equations:

$$A^T A x = A^T b$$

$$x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 7.0325 \\ 0.5044 \\ 0.0070 \end{pmatrix}$$

Note that the system is *artificially ill-conditioned* because the columns of A are out of scale.

$$\text{Cond}(A^T A) = 3.0891 \times 10^8$$

To see how the computed least-squares solution agrees with the data of the table, we compute

$$(1 \quad 274 \quad 2450)\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = 162.4043$$

(True value = 162)

$$(1 \quad 180 \quad 3254)\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = 120.6153$$

(True value = 120)

$$(1 \quad 375 \quad 3802)\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = 222.8193$$

(True value = 223)

$$(1 \quad 205 \quad 2838)\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = 130.3140$$

(True value = 131)

$$(1 \quad 86 \quad 2347)\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = 66.8471$$

(True value = 67)

Suppose that the company would like to predict, using these results, the sales in a district with the population 220,000 and per capita income of \$2500. Then the best prediction using the given model is

$$(1 \quad 220 \quad 2500)\begin{pmatrix} 7.0325 \\ 0.5044 \\ 0.0070 \end{pmatrix} = 135.5142$$

Note: The columns of A are out of scale; that is why A is ill conditioned. In practice, they should be scaled so that they have the same length. You can easily verify that in this case, $\text{Cond}(A^TA) = 264.68$, so the problem is not really ill-conditioned.

Example 7.8.2

$$A = \begin{pmatrix} 1 & 2 \\ 2 & 3 \\ 3 & 4 \end{pmatrix}, \qquad b = \begin{pmatrix} 3 \\ 5 \\ 9 \end{pmatrix}$$

Here, $\text{rank}(A) = 2$, and $\text{rank}(A, b) = 3$. Thus, the system $Ax = b$ does not have a solution. We therefore calculate the least-squares solution.

1. $c = A^Tb = \begin{pmatrix} 40 \\ 57 \end{pmatrix}$
2. The Cholesky factor of A^TA is

$$H = \begin{pmatrix} 3.7417 & 0 \\ 5.3452 & 0.6547 \end{pmatrix}$$

3. The solutions of the triangular systems are

$$y = \begin{pmatrix} 10.6904 \\ -0.2182 \end{pmatrix}$$

$$x = \begin{pmatrix} 3.3333 \\ -0.3333 \end{pmatrix}$$

The unique least-squares solution is $x = \begin{pmatrix} 3.3333 \\ -0.3333 \end{pmatrix}$.

Note that

$$A^\dagger = (A^TA)^{-1}A^T = \begin{pmatrix} -1.8333 & -0.3333 & 1.1667 \\ 1.3333 & 0.3333 & -0.6667 \end{pmatrix}$$

and

$$A^\dagger b = \begin{pmatrix} 3.3333 \\ -0.3333 \end{pmatrix}$$

Numerical Difficulties with the Normal Equations Method

The normal equations method, although easy to understand and implement, may give rise to numerical difficulties in certain cases.

First, we might lose some significant digits during the explicit formation of A^TA, and the computed matrix A^TA may be far from positive definite; computationally, it may even be singular. Indeed, it has been shown by Stewart (1973, 225–6) that, unless $\text{Cond}(A)$ is less than $10^{t/2}$, where it is assumed that A^TA has been computed exactly and then rounded to t figures, then A^TA may fail to be positive definite or even may be singular. The following simple example illustrates that fact.

Example 7.8.3 Consider

$$A = \begin{pmatrix} 1 & 1 \\ 10^{-4} & 0 \\ 0 & 10^{-4} \end{pmatrix}$$

with $t = 8$. The columns of A are linearly independent. Now, compute

$$A^TA = \begin{pmatrix} 1 + 10^{-8} & 1 \\ 1 & 1 + 10^{-8} \end{pmatrix}$$

Because $t = 8$, we will get

$$A^TA = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

which is singular. Note that $\text{Cond}(A) = 1.4142 \times 10^4 > 10^{t/2} = 10^4$.

Second, the normal equations approach may, in certain cases, introduce more errors than those inherent in the problem. This is seen as follows.

From the perturbation analysis done in Chapter 6, we easily see that, if $\hat{x}$ is the computed least-squares solution obtained by the normal equations method, then

$$\frac{\|\hat{x} - x\|}{\|x\|} \approx \mu\,\text{Cond}(A^TA)$$

$$= \mu(\text{Cond}(A))^2 \qquad \text{(Exercise 5)}$$

Thus, the accuracy of the least-squares solution using normal equations will depend upon the square of the condition number of the matrix A. However, we just saw in the section on perturbation analysis of the least-squares problem that in certain cases such as when the residual is zero, the sensitivity of the problem depends only on the condition number of A (see Theorems 7.7.1 and 7.7.2). *Thus, in these cases, the normal equations method will introduce more errors in the solution than the data warrant.*

A Special Note on the Normal Equations Method

In spite of the drawbacks of the normal equations method, we must stress that the method is still regarded as a useful tool for solving the least-squares problem, at least in the case where the matrix A is well conditioned. In fact, it is routinely used in many practical applications and seems to be quite popular with practicing engineers and statisticians. Note that, in Example 7.8.3, if we were to use an extended precision in our computations, the computed matrix A^TA could be obtained as a symmetric positive definite matrix and the normal equations method would yield an accurate answer, despite ill-conditioning; as the following computations show:

$$A^TA = \begin{pmatrix} 1.00000001 & 1 \\ 1 & 1.00000001 \end{pmatrix}, \qquad b = (2, 0.0001, 0.0001)^T$$

$$C = A^Tb = \begin{pmatrix} 2.00000001 \\ 2.00000001 \end{pmatrix}$$

$$H = \text{the Cholesky factor of } A^TA = \begin{pmatrix} 1.000000005 & 0 \\ 0.999999995 & 0.00014142135581 \end{pmatrix}$$

Solution of $Hy = c$:

$$y = \begin{pmatrix} 2 \\ 0.00014142135651 \end{pmatrix}$$

Solution of $H^Tx = y$:

$$x = \begin{pmatrix} 0.999999999500183 \\ 1.0000000499817 \end{pmatrix} \approx \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

(The exact solution.)

7.8.2 QR Factorization Methods for the Full-Rank Problem

Let

$$Q^TA = R = \begin{pmatrix} R_1 \\ 0 \end{pmatrix}$$

be the QR decomposition of the matrix A. Then, because the length of a vector is preserved by an orthogonal matrix multiplication, we have

$$\begin{aligned} \|Ax - b\|_2^2 &= \left\|Q^TAx - Q^Tb\right\|_2^2 \\ &= \|R_1x - c\|_2^2 + \|d\|_2^2 \end{aligned}$$

where

$$Q^Tb = \begin{pmatrix} c \\ d \end{pmatrix}$$

Thus, $\|Ax - b\|_2^2$ will be minimized if x is chosen so that

$$\boxed{R_1x - c = 0}$$

The corresponding residual then is given by

$$\boxed{\|r\|_2 = \|d\|_2}$$

This observation immediately suggests the following QR approach for solving the least-squares problem:

Least-Squares Using QR

1. Decompose $A_{m\times n} = Q_{m\times m}R_{m\times n}$.
2. Form $Q^T_{m\times m}b_{m\times 1} = \begin{pmatrix} c \\ d \end{pmatrix}$, c is $n \times 1$ and d is $(m - n) \times 1$.
3. Solve the $n \times n$ upper triangular system

   $$R_1x = c$$

   where

   $$R = \begin{pmatrix} R_1 \\ 0 \end{pmatrix}$$

Example 7.8.4

$$A = \begin{pmatrix} 1 & 2 \\ 2 & 3 \\ 3 & 4 \end{pmatrix}, \qquad b = \begin{pmatrix} 3 \\ 5 \\ 9 \end{pmatrix}$$

1. $A = QR$:

$$Q = \begin{pmatrix} -0.2673 & 0.8729 & 0.4082 \\ -0.5345 & 0.2182 & -0.8165 \\ -0.8018 & -0.4364 & 0.4082 \end{pmatrix}$$

$$R = \left(\begin{array}{cc} -3.7417 & -5.3452 \\ 0 & 0.6547 \\ \hline 0 & 0 \end{array}\right) = \begin{pmatrix} R_1 \\ 0 \end{pmatrix}$$

2. $$Q^Tb = \begin{pmatrix} c \\ d \end{pmatrix} = \begin{pmatrix} -10.6904 \\ -0.2182 \\ 0.8165 \end{pmatrix}$$

3. Solve the system $R_1x = c$:

$$\begin{pmatrix} -3.7417 & -5.3452 \\ 0 & 0.6547 \end{pmatrix} x = \begin{pmatrix} -10.6904 \\ -0.2182 \end{pmatrix}$$

The least-squares solution is

$$x = \begin{pmatrix} 3.3332 \\ -0.3333 \end{pmatrix}$$

The norm of the residual is $\|r\|_2 = \|d\| = 0.8165$.

Use of Householder Matrices

Once A is decomposed into QR and the vector $Q^T b$ is formed, the solution x is obtained by back substitution. Suppose that the Householder method is used to compute the QR decomposition of A. Then, following the notation of Chapter 5, we see that the product $Q^T b$ can be formed as:

For $k = 1, 2, \ldots, n$
$\quad b = H_k b$

where H_k, $k = 1, \ldots, n$, are the Householder matrices such that

$$Q^T = H_n H_{n-1} \cdots H_2 H_1$$

Thus, *the matrix Q does not have to be formed explicitly.*

The idea of solving the least-squares problem based on the QR factorization of A using the Householder transformation was first suggested by Golub (1965).

ALGORITHM 7.8.2 The Golub–Householder Method for Full-Rank Least-Squares Problem

Given an $m \times n$ matrix $A (m \geq n)$ with rank $(A) = n$ and an m-vector b, the following algorithm computes the least-squares solution x using Householder matrices H_1 through H_n.

Step 1: Apply Algorithm 5.4.3 to A. Obtain R_1 and the vectors defining the Householder matrices $H_1, H_2, \ldots, H_n$.

Step 2: Form $H_n H_{n-1} \cdots H_2 H_1 b = \begin{pmatrix} c \\ d \end{pmatrix}$, where c is an n-vector, using Algorithm 5.4.2, out of the vectors computed in Step 1.

Step 3: Solve $R_1 x = c$.

Flop count. Because the cost of the algorithm is dominated by the cost of the QR decomposition of A, the overall flop count for the full-rank least-squares solution using the Golub–Householder method is

$$n^2\left(m - \frac{n}{3}\right) \qquad \text{(Exercise 17(c))}$$

Thus, in case $m > n$, the method is more expensive than the normal equations method. Note that the normal equations method requires about $n^2 m/2 + n^3/6$ flops.

Roundoff error and stability. The method is *stable*. It has been shown in Lawson and Hanson (1974, 90) that the computed solution $\hat{x}$ is such that it minimizes

$$\|(A + E)x - (b + \delta b)\|_2$$

where E and δb are small. Specifically,

$$\|E\|_F \leq c\mu \|A\|_F + O(\mu^2)$$
$$\text{and } \|\delta b\|_2 \leq c\mu \|b\|_2 + O(\mu^2)$$

where $c \approx (6m - 3n + 41)n$ and μ is machine precision. That is, *the computed solution is the exact least-squares solution of a nearby problem.*

MATCOM Note: Algorithm 7.8.2 has been implemented in MATCOM program LSFRQRH.

Example 7.8.5 Find the least-squares solution of $Ax = b$ using the Golub–Householder method.

$$A = \begin{pmatrix} 1 & 1 \\ 0.0001 & 0 \\ 0 & 0.0001 \end{pmatrix}, \qquad b = \begin{pmatrix} 2 \\ 0.0001 \\ 0.0001 \end{pmatrix}$$

Step 1: Using Algorithm 5.4.3, we find the vectors u_3 and u_2 such that

$$H_2 H_1 A = \begin{pmatrix} R_1 \\ 0 \end{pmatrix}$$

where

$$H_1 = I - 2\frac{u_3 u_3^T}{u_3^T u_3}, \qquad H_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \hat{H}_2 & \\ 0 & & \end{pmatrix}$$

$$\hat{H}_2 = I - 2\frac{u_2 u_2^T}{u_2^T u_2}$$

$$u_3 = \begin{pmatrix} 2 \\ 0.0001 \\ 0 \end{pmatrix}, \qquad u_2 = 10^{-3}\begin{pmatrix} -0.2414 \\ 0.1000 \end{pmatrix}$$

$$R_1 = \begin{pmatrix} -1 & -1 \\ 0 & 0.0001 \end{pmatrix}$$

Step 2: Using Algorithm 5.4.2, we form

$$H_2 H_1 b = \begin{pmatrix} c \\ d \end{pmatrix}$$

$$H_1 b = \left(I - 2\frac{u_3 u_3^T}{u_3^T u_3}\right) b$$

$$= \begin{pmatrix} 2 \\ 0.0001 \\ 0.0001 \end{pmatrix} - 0.5\begin{pmatrix} 8 \\ 0.0004 \\ 0 \end{pmatrix}$$

$$= \begin{pmatrix} -2 \\ -0.0001 \\ 0.0001 \end{pmatrix}$$

$$\hat{H}_2\begin{pmatrix}-0.0001\\0.0001\end{pmatrix} = \left(I - 2\frac{u_2u_2^T}{u^Tu_2}\right)\begin{pmatrix}-0.0001\\0.0001\end{pmatrix}$$
$$= \begin{pmatrix}-0.0001\\0.0001\end{pmatrix} - 10^{-3}\begin{pmatrix}-0.2414\\0.1000\end{pmatrix}$$
$$= 10^{-3}\begin{pmatrix}0.1414\\0.0000\end{pmatrix}$$

So,

$$H_2H_1b = \begin{pmatrix}-2\\10^{-3}(0.1414)\\0\end{pmatrix} = \begin{pmatrix}c\\d\end{pmatrix}$$

Thus,

$$c = \begin{pmatrix}-2\\0.0001\end{pmatrix} \text{ (in four digit arithmetic)}$$

Step 3: Solve $R_1x = c$.

$$x = \begin{pmatrix}1\\1\end{pmatrix}$$

Use of Givens Rotations

We can, of course, use the Givens rotations to decompose A into QR and then use this decomposition to solve the least-squares problem. However, as we have seen before, the use of Givens rotations will be more expensive than the use of Householder matrices. Recall that computations of Givens rotations require evaluations of square roots; however, there are *square root–free Givens rotations*, introduced by Gentleman (1973), which can be used to solve the least-squares problem. The square root–free Givens rotations are also known as *fast Givens rotations*.

Use of Gram–Schmidt and Modified Gram–Schmidt Algorithms

Another technique, known as the **Gram–Schmidt process**, can be used to decompose A into QR. A properly implemented Gram–Schmidt process, when used to solve the linear least-squares problem, is just slightly more expensive than the Householder method, but it seems to be as numerically effective as the Householder method for solving the least-squares problem. We first state these algorithms for QR factorization and then show how the modified Gram–Schmidt algorithm can be used to solve the least-squares problem.

ALGORITHM 7.8.3 Classical Gram–Schmidt (CGS) for QR Factorization Given $A = (a_1, a_2, \ldots, a_n)$ of order $m \times n$ with rank$(A) = n$, the following algorithm computes a matrix $Q_{m\times n} = (q_1, \ldots, q_n)$ with orthonormal columns and an upper triangular matrix $R = (r_{ij})_{n\times n}$ such that $A = Q_{m\times n}R_{n\times n}$.

For $k = 1, 2, \ldots, n$ do
  For $i = 1, 2, \ldots, k-1$ do
    $r_{ik} = q_i^T a_k$

$$q_k = a_k - \sum_{i=1}^{k-1} r_{ik}q_i$$
$$r_{kk} = \|q_k\|_2$$
$$q_k \equiv \frac{q_k}{r_{kk}}$$

The algorithm, as outlined here, is known to have serious numerical difficulties. *During the computations of the q_k's, cancellation can take place and, as a result, the computed q_k's can be far from orthogonal.* (We provide details later in this section.)

The algorithm, however, can be modified to have better numerical properties. The following algorithm, known as the **modified Gram–Schmidt algorithm**, computes the QR factorization of A, in which, at the kth step, the kth column of Q and the kth row of R are computed (note that the Gram–Schmidt algorithm computes the kth columns of Q and R at the kth step).

ALGORITHM 7.8.4 Modified Gram–Schmidt (MGS) for QR Factorization

Set $q_k = a_k$, $k = 1, 2, \ldots, n$.
  For $k = 1, 2, \ldots, n$ do
    $r_{kk} = \|q_k\|_2$
    $q_k \equiv \dfrac{q_k}{r_{kk}}$
    For $j = k+1, \ldots, n$ do
      $r_{kj} \equiv q_k^T q_j$
      $q_j = q_j - r_{kj}q_k$

This algorithm is the *row-oriented modified Gram–Schmidt method.* The column-oriented version can be developed similarly (Exercise 17). The two versions are numerically equivalent.

Remark: The modified Gram–Schmidt algorithm is not as satisfactory as the Householder or the Givens method for computing the QR factorization of A. It can be shown (Björck 1994a or 1994b) that the computed Q by MGS, denoted by $\tilde{Q}$, satisfies $\tilde{Q}^T\tilde{Q} = I + E$, where $\|E\| \approx \mu\,\text{Cond}(A)$, whereas the Householder method produces $\tilde{Q}^T\tilde{Q} = I + E$, where $\|E\| \approx \mu$. (For more details, see the discussions in the next section.)

Flop count. The flop count for the MGS is mn^2, compared to $mn^2 - n^3/3$ needed by the Householder method. (*Note that MGS works with the full-length column vector at each step, whereas the Householder method deals with successively shorter columns.*)

MATCOM Note: Algorithms 7.8.3 and 7.8.4 have been implemented in MATCOM programs CLGRSCH and MDGRSCH, respectively.

Example 7.8.6 Find the QR factorization of

$$A = \begin{pmatrix} 1 & 2 \\ 2 & 3 \\ 6 & 7 \end{pmatrix}$$

using modified Gram–Schmidt process (Algorithm 7.8.4).

$k = 1$:

$$r_{11} = \|q_1\|_2 = \|a_1\|_2 = 6.4031$$

$$q_1 = \frac{a_1}{r_{11}} = (0.1562, 0.3123, 0.9370)$$

$$r_{12} = q_1^T q_2 = q_1^T a_2 = 7.8087$$

$$q_2 \equiv q_2 - r_{12}q_1 = a_2 - r_{12}q_1 = \begin{pmatrix} 0.7805 \\ 0.5610 \\ -0.3171 \end{pmatrix}$$

$k = 2$:

$$r_{22} = 1.0121$$

$$q_2 \equiv \frac{q_2}{r_{22}} = (0.7711, 0.5543, -0.3133)^T$$

Thus,

$$Q = (q_1, q_2) = \begin{pmatrix} 0.1562 & 0.7711 \\ 0.3123 & 0.5543 \\ 0.9370 & -0.3133 \end{pmatrix}_{3\times 2}$$

$$R = \begin{pmatrix} 6.4031 & 7.8087 \\ 0 & 1.0121 \end{pmatrix}_{2\times 2}$$

Verify that $A = QR$. Note that if you had used Householder QR (Algorithm 5.4.3), the results were

$$Q = \begin{pmatrix} -0.1562 & -0.7711 & -0.6172 \\ 0.3123 & -0.5543 & 0.7715 \\ -0.9370 & 0.3133 & -0.5143 \end{pmatrix}_{3\times 3}$$

$$R = \begin{pmatrix} -6.4031 & -7.8087 \\ 0 & -1.0121 \\ 0 & 0 \end{pmatrix}_{3\times 2}$$

Modified Gram–Schmidt versus Classical Gram–Schmidt Algorithms

Mathematically, the CGS and MGS algorithms are equivalent. However, as we remarked earlier, their numerical properties are different. For example, consider the computation of q_2 by the CGS method, given q_1 with $\|q_1\|_2 = 1$. We have

$$q_2 = a_2 - r_{12}q_1$$

where $r_{12} = q_1^T a_2$. Then it can be shown (Björck 1994b) that

$$\|\mathrm{fl}(q_2) - q_2\| < (1.06)(2m + 3)\mu \, \|a_2\|_2$$

Because $q_1^T q_2 = 0$, it follows that

$$|q_1^T \, \mathrm{fl}(q_2)| < (1.06)(2m + 3)\mu \, \|a_2\|_2$$

This shows that in CGS, two computed vectors, q_1 and q_2, can be far from being orthogonal. On the other hand, it can be shown (Björck 1994b) that in MGS the loss of orthogonality depends upon the condition number of the matrix A. Specifically, it has been shown that

$$\left\|I - \hat{Q}^T\hat{Q}\right\|_2 \le \frac{c_1\mu \, \mathrm{Cond}_2(A)}{1 - c_2\mu \, \mathrm{Cond}_2(A)}$$

assuming that $c_2\mu \, \mathrm{Cond}_2(A) < 1$, where c_1 and c_2 are small constants.

Since in MGS the loss of orthogonality depends upon the condition number, we could use column pivoting to maintain orthogonality as much as possible. Thus, as far as finding the QR factorization of A is concerned, neither algorithm can be recommended over the Householder or the Givens method. *With CGS the orthogonality of Q can be completely lost; with MGS, the orthogonality of Q may not be acceptable when A is ill-conditioned.*

For example, for the 4×3 matrix of A of the type given in Example 7.8.5, the computed $\hat{Q}$ by *CGS* is such that $|q_3^T q_2| = \frac{1}{2}$, and for *MGS*, $|q_3^T q_1| = (\frac{2}{3})^{1/2} \cdot 10^{-4}$. See Björck (1994b).

To compare the computations with the different Gram–Schmidt processes, we also computed QR factorizations of the 5×5 *Hilbert matrix* using the Gram–Schmidt, the modified Gram–Schmidt, and the Householder methods (which is known to be very stable) and tabulated the results of $\|I - \hat{Q}^T\hat{Q}\|$ in extended precision. The results are displayed in Table 7.1.

Remark: The table clearly shows the superiority of the Householder method over both the Gram–Schmidt and modified Gram–Schmidt methods; of the latter two methods MGS is clearly preferred over CGS.

Use of the MGS Method in the Least-Squares Solution

We have just seen that the MGS is superior to the CGS for QR factorization, but it is still not fully satisfactory. However, as far as the least-squares problem is concerned, it is a different story. There seems to be a growing tendency to use MGS in solutions of least-squares problems. Björck and Paige (1992) have recently shown that *MGS*

TABLE 7.1 Comparison of QR Factorization with Different Methods

Method	$\|I - \hat{Q}^T\hat{Q}\|$
Gram–Schmidt	$1.178648780488241 \times 10^{-7}$
Modified Gram–Schmidt	$4.504305729523455 \times 10^{-12}$
Householder	$4.841449989971538 \times 10^{-16}$

is numerically equivalent to Householder QR factorization applied to A augmented with a square matrix of zero elements on top. However, while using MGS in the solution of a least-squares problem, special care has to be taken. If $c = Q^T b$ is computed using Q obtained from MGS and this c is used to solve $Rx = c$, then the computed least-squares solution may not be accurate. The MGS should be applied to the augmented matrix (A, b) to compute the factorization:

$$(A, b) = (Q_1, q_{n+1})\begin{pmatrix} R & z \\ 0 & \rho \end{pmatrix}$$

From this it follows that

$$\begin{aligned} Ax - b &= (A, b)\begin{pmatrix} x \\ -1 \end{pmatrix} \\ &= (Q_1, q_{n+1})\begin{pmatrix} R & z \\ 0 & \rho \end{pmatrix}\begin{pmatrix} x \\ -1 \end{pmatrix} \\ &= Q_1(Rx - z) - \rho q_{n+1} \end{aligned}$$

If q_{n+1} is orthogonal to Q_1, then $\|Ax - b\|_2$ will be a minimum when $Rx = z$. Thus, the least-squares solution can be obtained by solving

$$Rx = z$$

and

the residual r is given by $r = \rho q_{n+1}$.

Details can be found in Björck (1994a or 1994b). The preceding discussion leads to the following least-squares algorithm.

ALGORITHM 7.8.5 Least-Squares Solution by MGS

Step 1: Apply MGS to $A_{m\times n}$ to obtain $Q = (q_1, \ldots, q_n)$ and R.

Step 2: For $k = 1, \ldots, n$ do

$$\delta_k = q_k^T b$$
$$b \equiv b - \delta_k q_k$$

Step 3: Solve $Rx = (\delta_1, \ldots, \delta_n)^T$.

Example 7.8.7 Consider solving the least-squares problem using the MGS with

$$A = \begin{pmatrix} 1 & 1 \\ 0.0001 & 0 \\ 0 & 0.0001 \end{pmatrix}, \qquad b = \begin{pmatrix} 2 \\ 0.0001 \\ 0.0001 \end{pmatrix}$$

The exact solution is $x = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$.

$$Q = \begin{pmatrix} 1 & 0 \\ 0.0001 & -.7071 \\ 0 & .7071 \end{pmatrix}, \qquad R = \begin{pmatrix} 1 & 1 \\ 0 & 0.0001 \end{pmatrix}$$

If we now form $c = Q^T b$ and solve $Rx = c$, we obtain $x = \begin{pmatrix} 2 \\ 0 \end{pmatrix}$. On the other hand, if we obtain x using the preceding algorithm, we get $(\delta_1, \delta_2) = (2, 0.0001)$, and the solution of $Rx = (\delta_1, \delta_2)^T$ is $x \approx \begin{pmatrix} 1 \\ 1 \end{pmatrix}$.

Roundoff property and flop count. It can be shown (Björck and Paige 1992) that the MGS algorithm for the least-squares problem is numerically equivalent to the Householder method applied to

$$\begin{pmatrix} 0 & 0 \\ A & b \end{pmatrix}$$

that is,

$$H_n H_{n-1} \cdots H_2 H_1 \begin{pmatrix} 0 & 0 \\ A & b \end{pmatrix} = \begin{pmatrix} R & c_1 \\ 0 & c_2 \end{pmatrix}$$

From this equivalence, it follows that *the MGS method is backward stable for the solution of the least-squares problem.* The method is slightly more expensive than the Householder method. *It requires about mn^2 flops, compared to the $mn^2 - n^3/3$ flop needed by the Householder method.*

MATCOM Note: Algorithm 7.8.5 has been implemented in MATCOM program LSFRMGS.

7.8.3 The QR Factorization Method for the Rank-Deficient Case

In this section we consider the rank-deficient least-squares problem. As stated in Theorem 7.3.1, there are infinitely many solutions in this case. There are instances where rank-deficiency is actually desirable, because it provides a rich family of solutions that might be used for optimizing some other aspects of the original problem.

In case the $m \times n$ matrix A, $m \geq n$, has rank $r < n$, the R matrix in a QR factorization of A is rank-deficient. However, we have seen that the use of QR factorization with column pivoting can, *theoretically*, reveal the rank of A. Recall that Householder's method with column pivoting yields

$$Q^T AP = \begin{pmatrix} R_{11} & R_{12} \\ 0 & 0 \end{pmatrix}$$

where P is a permutation matrix, R_{11} is an $r \times r$ nonsingular upper triangular matrix, and R_{12} is $r \times (n - r)$. This factorization can obviously be used to solve the rank-deficient least-squares problem as follows.

Let

$$P^T x = y$$

$$AP = \tilde{A}$$

and

$$Q^T b = \begin{pmatrix} c \\ d \end{pmatrix}$$

Then

$$\|Ax - b\|_2^2 = \left\|Q^T APP^T x - Q^T b\right\|_2^2 \qquad \text{(2-norm remains invariant under orthogonal matrix multiplication)}$$

$$= \left\|\begin{pmatrix} R_{11} & R_{12} \\ 0 & 0 \end{pmatrix}\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} - \begin{pmatrix} c \\ d \end{pmatrix}\right\|_2^2$$

$$= \|(R_{11}y_1 + R_{12}y_2 - c)\|_2^2 + \|d\|_2^2$$

Thus, $\|Ax - b\|_2^2$ will be minimized if y is chosen so that

$$\boxed{R_{11}y_1 = c - R_{12}y_2}$$

Moreover, the norm of the residual in this case will be given by

$$\boxed{\|r\|_2 = \|b - Ax\|_2 = \|d\|_2}$$

This observation suggests the following QR factorization algorithm for rank-deficient least-squares solutions.

ALGORITHM 7.8.6 Least-Squares Solutions for the Rank-Deficient Problem Using QR

Step 1: Using QR decomposition with *column pivoting*, decompose A into

$$Q^T AP = \begin{pmatrix} R_{11} & R_{12} \\ 0 & 0 \end{pmatrix}$$

Step 2: Form $Q^T b = \begin{pmatrix} c \\ d \end{pmatrix}$.

Step 3: Choose an arbitrary vector y_2

Step 4: Solve the $r \times r$ nonsingular upper triangular system

$$R_{11}y_1 = c - R_{12}y_2$$

Step 5: Recover x from

$$x = Py, \qquad y = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$$

Computational Note: Again, Step 1 and Step 2 should be implemented without computing explicitly the matrix Q.

Minimum Norm Solution

Because y_2 is arbitrary, the preceding method will yield infinitely many least-squares solutions, as Theorem 7.3.1 states. The solution obtained by setting $y_2 = 0$ is called the **basic solution**. In case $R_{12} = 0$, the basic solution is the **minimum-norm** least-squares solution; that is, among the infinite number of least-squares solutions, it has the minimum norm. In case $R_{12} \neq 0$, the basic solution can not be the minimum-norm solution (Exercise 20). In such a case, the minimum norm solution is obtained by taking y_2 as a solution to $\min y_2 \|Sy_2 - t\|_2$, where

$$S = \begin{pmatrix} R_{11}^{-1}R_{12} \\ -I_{(n-r)\times(n-r)} \end{pmatrix}$$

$$t = \begin{pmatrix} R_{11}^{-1}c \\ 0 \end{pmatrix}$$

Alternatively, we can use the *complete QR factorization with column pivoting*. Recall from Chapter 5 that the complete QR factorization of an $m \times n$ matrix A is given by

$$Q^T AW = \begin{pmatrix} T & 0 \\ 0 & 0 \end{pmatrix}$$

If pivoting is used, then

$$AP = Q\begin{pmatrix} T & 0 \\ 0 & 0 \end{pmatrix}W^T$$

where P is a permutation matrix.

The minimum norm solution will then be given by

$$x = PW\begin{pmatrix} T^{-1}c \\ 0 \end{pmatrix}$$

where c is a vector consisting of the first r elements of $Q^T b$ (Exercise 21(a)).

Remarks:

1. A note on the use of column pivoting: We have shown that the column pivoting is useful for the rank-deficient least-squares problem. *However, even in the full-rank case, the use of column pivoting is suggested.*
2. *For the rank-deficient least-squares problem, the most reliable approach is the use of singular-value decomposition* (see Section 7.8.4 and Chapter 10).

Roundoff property. It can be shown (Lawson and Hanson 1974, 95) that for the minimum-length least-squares problem, the computed vector $\hat{x}$ is *close* to the exact solution of a perturbed problem. That is, there exists a matrix E and vectors $\bar{x}$ and δb such that $\bar{x}$ is the minimum-length least-squares solution of $(A + E)x \approx b + \delta b$, where

$$\|E\|_F \leq (6m + 6n - 6r - 3s + 84)s\mu\,\|A\|_F + O(\mu^2)$$

and $\|\delta b\| \le (6m - 3r + 40)r\mu \|b\| + O(\mu^2)$, where r is the rank of A and $s = \min(m, n)$. Moreover,

$$\|\bar{x} - \hat{x}\| \le (6n - 6r + 43)r\mu \|\hat{x}\| + O(\mu^2)$$

Note: In the preceding result we have assumed that R_{22} in

$$R = \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix}$$

is *zero*. But in practical computations it will not be identically zero. In that case, if $\hat{R}_{22}$ is the computed version of R_{22}, then we have

$$\|E\|_F \le \left\|\hat{R}_{22}\right\|_F + (6m + 6n - 6r - 3s + 84)s\mu \|A\|_F + O(\mu^2)$$

MATCOM Note: Algorithm 7.8.6 has been implemented in MATCOM program LSRDQRH using Householder QR with column pivoting.

Example 7.8.8

$$A = \begin{pmatrix} 1 & 0 \\ 2 & 0 \\ 0 & 0 \end{pmatrix}, \qquad b = \begin{pmatrix} 3 \\ 6 \\ 0 \end{pmatrix}, \qquad \text{rank}(A) = 1$$

Step 1: $AP = QR, P = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$

$$Q = \begin{pmatrix} -0.4472 & -0.8944 & 0 \\ -0.8944 & -0.4472 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$R = \begin{pmatrix} -2.2361 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}$$

Step 2: $Q^T b = \begin{pmatrix} -6.7082 \\ 0 \\ 0 \end{pmatrix}, c = (-6.7082)$

Step 3: Choose $y_2 = 0$

Step 4: Solve $R_{11}y_1 = c - R_{12}y_2$:

$$y_1 = \frac{c}{R_{11}} = \frac{-6.7082}{-2.2361} = 3$$

The minimum-norm least-squares solution is

$$x = y = \begin{pmatrix} 3 \\ 0 \end{pmatrix}$$

Example 7.8.9

$$A = \begin{pmatrix} 0.6667 & 0.3333 \\ -1.3333 & -0.6667 \\ -1.3333 & -0.6667 \end{pmatrix}, \qquad b = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

Step 1:

$$AP = QR, P = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \qquad R = \begin{pmatrix} -2 & -1 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}$$

Step 2:

$$Q^T b = \begin{pmatrix} c \\ d \end{pmatrix} = \begin{pmatrix} 1.000 \\ -1.4142 \\ 0 \end{pmatrix}$$

$$c = 1.000$$

Step 3: Since $R_{12} = -1 \neq 0$, we compute y_2 as the least-squares solution with S and t.

$$S = \begin{pmatrix} 0.5000 \\ -1.000 \end{pmatrix}, \qquad t = \begin{pmatrix} -0.5000 \\ 0.000 \end{pmatrix}$$

$$y_2 = -0.2000$$

Step 4: y_1 is computed by solving

$$R_{11}y_1 = c - R_{12}y_2$$

$$y_1 = -0.4000$$

Step 5: The minimum norm least-squares solution is

$$x = y = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} -0.4000 \\ -0.2000 \end{pmatrix}$$

7.8.4 Least-Squares Solution Using SVD

Acting on the remark in the previous section, for the sake of curious readers and those who know how to compute the singular value decomposition (SVD) of a matrix (for instance, using some software package such as MATLAB, LAPACK, or LINPACK), we state the following results, which show how the SVD can be used to solve a least-squares problem. A full treatment is given in Chapter 10.

Let

$$A = U\Sigma V^T$$

be the SVD of A, where A is $m \times n$ with $m \ge n$ and $\Sigma = \text{diag}(\sigma_1, \ldots, \sigma_n)$. Let

$$b' = U^T b = \begin{pmatrix} b_1' \\ \vdots \\ b_m' \end{pmatrix}$$

Least-Squares Solutions Using SVD

A family of the least-squares solutions is given by

$$x = Vy$$

where

$$y = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}; \qquad y_i = \begin{cases} \dfrac{b_i'}{\sigma_i}, & \sigma_i \neq 0 \\ \text{arbitrary}, & \sigma_i = 0 \end{cases}$$

In the rank-deficient case, an expression for the minimum-norm solution is

$$x = \sum_{i=1}^{r} \frac{u_i^T b}{\sigma_i} v_i$$

where $r = \text{rank}(A)$ and u_i and v_i are the columns of U and V, respectively. (*Note that rank(A) = the number of nonzero singular values of A.*)

Example 7.8.10

$$A = \begin{pmatrix} 1 & 1 \\ 0.0001 & 0 \\ 0 & 0.0001 \end{pmatrix}, \qquad b = \begin{pmatrix} 2 \\ 0.0001 \\ 0.0001 \end{pmatrix}$$

$A = U\Sigma V^T$ gives

$$U = \begin{pmatrix} 1 & 0 & -0.0001 \\ 0 & -0.7071 & 0.7071 \\ 0 & 0.7071 & 0.7071 \end{pmatrix}$$

$$\Sigma = \begin{pmatrix} 1.4142 & 0 \\ 0 & 0.0001 \\ 0 & 0 \end{pmatrix}$$

$$V = \begin{pmatrix} 0.7071 & -0.7071 \\ 0.7071 & 0.7071 \end{pmatrix}$$

$$b' = U^T b = \begin{pmatrix} 2 \\ 0 \\ 0 \end{pmatrix}, \qquad y = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} 1.4142 \\ 0 \end{pmatrix}, \qquad x = Vy = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

7.9 Underdetermined Systems

Let A be $m \times n$ and $m < n$. Then the system

$$Ax = b$$

has less equations than unknowns. Such a system is called an *underdetermined system*. An underdetermined system can be illustrated graphically as shown in Figure 7.3.

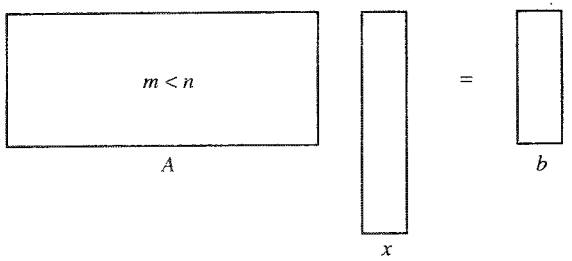


FIGURE 7.3

Underdetermined systems arise in a variety of practical applications. Unfortunately, underdetermined systems are not widely discussed in the literature. An excellent source is the survey paper by Cline and Plemmons (1976). An underdetermined system has either no solution or an infinite number of solutions. This can be seen from the following theorem.

THEOREM 7.9.1 Let $Ax = b$ be an underdetermined system. Then every solution x of the system can be represented as

$$x = x_R + x_N$$

where x_R is any solution; that is,

$$Ax_R = b$$

and x_N is in the null space of A, or

$$Ax_N = 0$$

Note: *A full-rank underdetermined system has an infinite number of solutions.* In this case, the complete solution set is given by

An Expression for the Solution Set of the Full-Rank Underdetermined System

$$x = A^T(AA^T)^{-1}b + (I - A^T(AA^T)^{-1}A)y, \qquad \text{where } y \text{ is arbitrary}$$

In the following, we describe two approaches for computing the *minimum-norm solution* assuming that A has full rank; for example, $\text{rank}(A) = m$.

7.9.1 The Minimum-Norm Solution of the Full-Rank Underdetermined Problem Using Normal Equations

THEOREM 7.9.2 Let A be $m \times n$ $(m < n)$ and have full rank. Then the minimum-norm solution x to the underdetermined system

$$Ax = b \tag{7.9.1}$$

is given by

$$x = A^T(AA^T)^{-1}b \tag{7.9.2}$$

Proof: Because A has full rank, AA^T is nonsingular, and clearly x is a solution of the system (7.9.1) given by (7.9.2). To see that x is indeed the minimum-norm solution, let's assume that y is another solution and define

$$z = y - x$$

Then

$$\begin{aligned}\|y\|_2^2 &= \|x + z\|_2^2 \\ &= \|x\|_2^2 + \|z\|_2^2 + 2(x^T z) \\ &= \|x\|_2^2 + \|z\|_2^2 + 2(b^T(AA^T)^{-1}A(y - x)) \\ &= \|x\|_2^2 + \|z\|_2^2\end{aligned}$$

(because $A(y - x) = Ay - Ax = b - b = 0$). This shows that

$$\|y\|_2 \geq \|x\|_2^2$$

Thus, x is the minimum-norm solution. ■

Note: The matrix $A^T(AA^T)^{-1}$ is the pseudoinverse of A.

Theorem 7.9.2 immediately suggests the following algorithm.

ALGORITHM 7.9.1 Minimum-Norm Solution for the Full-Rank Underdetermined Problem Using Normal Equations

Step 1: Solve

$$AA^T y = b$$

Step 2: Form the minimum norm solution x:

$$x = A^T y$$

MATCOM Note: Algorithm 7.9.1 has been implemented in MATCOM program MNUDNME.

Example 7.9.1

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \end{pmatrix}, \qquad b = \begin{pmatrix} 6 \\ 9 \end{pmatrix}$$

Step 1:

$$AA^T = \begin{pmatrix} 14 & 20 \\ 20 & 29 \end{pmatrix}$$

$$y = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$$

Step 2:

$$x = A^T y = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

Remark: The same difficulties with the normal equations that were pointed out in the overdetermined case may arise in the underdetermined case as well. For example, consider the explicit formation of AA^T, with t digits, when

$$A = \begin{pmatrix} 1 & \epsilon & 0 \\ 1 & 0 & \epsilon \end{pmatrix}$$

If ϵ is such that $10^{-t} < \epsilon < 10^{-t/2}$, then

$$AA^T = \begin{pmatrix} 1 + \epsilon^2 & 1 \\ 1 & 1 + \epsilon^2 \end{pmatrix}$$

is singular.

7.9.2 The QR Approach for the Full-Rank Underdetermined Problem

A QR factorization of A can, of course, be used to solve an underdetermined system and, in particular, to compute the minimum norm solution.

Because A in this case has more columns than rows, we decompose A^T instead of A into QR. Thus, we have

$$Q^T A^T = \begin{pmatrix} R_1 \\ 0 \end{pmatrix}$$

(Note that A^T is $n \times m$, $n > m$.) So, the system $Ax = b$ becomes

$$(R_1^T, 0^T)Q^T x = b$$

Let $y = Q^T x$. Then we have

$$(R_1^T, 0^T)y = b$$

Partitioning y conformally, we get

$$(R_1^T, 0^T)\begin{pmatrix} y_R \\ y_N \end{pmatrix} = b$$

or

$$R_1^T y_R = b$$

The minimum-norm solution is obtained by setting

$$y_N = 0$$

So, the minimum-norm solution is given by

$$x = Qy = (Q_1, Q_2)\begin{pmatrix} y_R \\ 0 \end{pmatrix} = Q_1 y_R$$

The preceding discussion suggests the following algorithm.

ALGORITHM 7.9.2 Minimum-Norm Solution for the Full-Rank Underdetermined Problem Using QR

Step 1: Find the QR factorization of A^T:

$$Q^T A^T = \begin{pmatrix} R_1 \\ 0 \end{pmatrix}$$

Step 2: Partition $Q = (Q_1, Q_2)$

Step 3: Solve the $m \times m$ nonsingular lower triangular system:

$$R_1^T y_R = b$$

Step 4: Form $x = Q_1 y_R$.

A note on implementation. Note that if we use the Householder method to compute the QR factorization of A, the product $Q_1 y_R$ can be computed from the factored form of Q using Algorithm 5.4.2.

Flop count. Using Householder orthogonalization, $m^2 n - m^3/3$ flops will be required to implement the above algorithm.

Roundoff property. It has been shown (Lawson and Hanson 1974, 93) that the computed vector $\hat{x}$ is close to the exact minimum-length solution of a perturbed problem; that is, there exist a matrix E and a vector $\bar{x}$ such that $\bar{x}$ is the minimum-length solution of

$$(A + E)x = b$$

where

$$\|E\|_F \le (6n - 3m + 41)m\mu \, \|A\|_F + O(\mu^2), \text{ and}$$

$$\|\hat{x} - \bar{x}\| \le (6n - 3m + 40)m\mu \, \|\hat{x}\| + O(\mu^2)$$

MATCOM Note: Algorithm 7.9.2 has been implemented in MATCOM program MNUDQRH.

Example 7.9.2

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \end{pmatrix}, \qquad b = \begin{pmatrix} 6 \\ 9 \end{pmatrix}$$

Step 1:

$$Q = \begin{pmatrix} -0.2673 & 0.8729 & 0.4082 \\ -0.5345 & 0.2182 & -0.8165 \\ -0.8018 & -0.4364 & -0.4082 \end{pmatrix}$$

$$Q^T A^T = \begin{pmatrix} -3.7417 & -5.3452 \\ 0.0000 & 0.6547 \\ 0.0000 & 0.0000 \end{pmatrix}$$

Step 2:

$$Q_1 = \begin{pmatrix} -0.2673 & 0.8729 \\ -0.5345 & 0.2182 \\ -0.8018 & -0.4364 \end{pmatrix}$$

$$R_1 = \begin{pmatrix} -3.7417 & -5.3452 \\ 0.0000 & 0.6547 \end{pmatrix}$$

Step 3:

$$y_R = \begin{pmatrix} -1.6036 \\ 0.6547 \end{pmatrix}$$

Step 4: The minimum-norm solution is

$$x = Q_1 y_R = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

7.9.3 Minimum-Norm Solution by MGS

A recent new result (see Björck and Paige 1992) is that the modified Gram–Schmidt (MGS) process can be used to compute the minimum norm solution to an underdetermined system, in a numerically stable way. The backward stability follows from the numerical equivalence of the MGS with the special Householder factorization as mentioned earlier. For details, see Björck (1994a) or Björck and Paige (1992).

7.10 Iterative Refinement

It is natural to wonder if a computed least-squares solution x can be improved cheaply in an iterative manner, as was done in the case of a linear system. A natural analog of the iterative refinement procedure for the linear system problem described in Section 6.9 will be the following. The scheme was proposed by Golub (1965).

ALGORITHM 7.10.1 Linear System Analog Iterative Refinement For Least-Squares Solution

Let $x^{(1)}$ be an approximate solution of the least-squares problem. Then

For $k = 1, 2, \ldots,$ do until convergence occurs:

Step 1: $r^{(k)} = b - Ax^{(k)}$ (Compute the residual in double precision.)

Step 2: Compute $c^{(k)}$ such that $\|Ac^{(k)} - r^{(k)}\|_2$ is minimum.

Step 3: Correct the solution:

$$x^{(k+1)} = x^{(k)} + c^{(k)}$$

MATCOM Note: Algorithm 7.10.1 has been implemented in MATCOM program LSITRN1.

An analysis of this refinement procedure by Golub and Wilkinson (1966) reveals that the method is satisfactory only when the true residual vector $r = b - Ax$ is sufficiently small. A successful procedure used widely in practice now follows. The method is based upon an interesting observation made by Golub that the least-squares solution x and the corresponding residual vector r satisfy the linear system

$$\begin{pmatrix} I & A \\ A^T & 0 \end{pmatrix}\begin{pmatrix} r \\ x \end{pmatrix} = \begin{pmatrix} b \\ 0 \end{pmatrix}$$

Note that this system is equivalent to

$$Ax + r = b$$

and

$$A^T r = 0$$

which means that x is the solution of the normal equation

$$A^T Ax = A^T b$$

Thus, we can apply the iterative refinement procedure described in Chapter 6 to this augmented linear system, yielding the following scheme, which is due to Björck (1967; 1968).

ALGORITHM 7.10.2 Iterative Refinement for Least-Squares Solutions

Set $r^{(0)} = 0$, $x^{(0)} = 0$.

For $k = 1, 2, \ldots,$ do

Step 1: Compute $\begin{pmatrix} r_1^{(k)} \\ r_2^{(k)} \end{pmatrix} = \begin{pmatrix} b \\ 0 \end{pmatrix} - \begin{pmatrix} I & A \\ A^T & 0 \end{pmatrix}\begin{pmatrix} r^{(k)} \\ x^{(k)} \end{pmatrix}$ (using accumulation of inner products in double precision).

Step 2: Solve the system $\begin{pmatrix} I & A \\ A^T & 0 \end{pmatrix}\begin{pmatrix} c_1^{(k)} \\ c_2^{(k)} \end{pmatrix} = \begin{pmatrix} r_1^{(k)} \\ r_2^{(k)} \end{pmatrix}$

Step 3: Update the solution and the residual: $\begin{pmatrix} r^{(k+1)} \\ x^{(k+1)} \end{pmatrix} = \begin{pmatrix} r^{(k)} \\ x^{(k)} \end{pmatrix} + \begin{pmatrix} c_1^{(k)} \\ c_2^{(k)} \end{pmatrix}$

Remark: Higham (1991) has recently shown that the above iterative refinement procedure with single precision residuals, although not as effective as the square linear system problems (Algorithm 6.9.1), still improves a componentwise measure of backward stability. Thus, even if Step 1 in Algorithm 7.10.2 is not performed in double precision, the results may still be satisfactory. See also Björck (1994b).

MATCOM Note: Algorithm 7.10.2 has been implemented in MATCOM program LSITRN2.

Implementation of Step 2

Because the matrix

$$\begin{pmatrix} I & A \\ A^T & 0 \end{pmatrix}$$

is of order $m + n$, the preceding scheme would be quite expensive when m is large. Fortunately, we can implement the scheme rather cheaply. Observe that if

$$Q^T A = \begin{pmatrix} R_1 \\ 0 \end{pmatrix}$$

is the QR decomposition of A, then the system

$$\begin{pmatrix} I & A \\ A^T & 0 \end{pmatrix}\begin{pmatrix} c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} r_1 \\ r_2 \end{pmatrix}$$

can be transformed into

$$Q^T c_1 + \begin{pmatrix} R_1 \\ 0 \end{pmatrix} c_2 = Q^T r_1$$

$$(R_1^T, 0) Q^T c_1 = r_2$$

This result shows that the preceding augmented system can be solved by solving two triangular systems and computing two matrix–vector multiplications, as follows:

1. Form $Q^T r_1 = \begin{pmatrix} r_1' \\ r_2' \end{pmatrix}$.
2. Solve for c_2': $R_1^T c_2' = r_2$.
3. Solve for c_2: $R_1 c_2 = r_1' - c_2'$.
4. Form $c_1 = Q\begin{pmatrix} c_2' \\ r_2' \end{pmatrix}$.

Flop count. With the preceding formulation, each iteration will require only $4mn - n^2$ flops, assuming that the Householder method has been used and that Q has not been formed explicitly. Note that for the matrix–vector multiplications in steps 1 and 4, Q does not need to be formed explicitly; these products can be obtained if Q is known only in implicit form.

Roundoff error. It can be shown (Björck 1994b) that the solution $x^{(s)}$ obtained at iteration step s by Algorithm 7.10.2 satisfies

$$\frac{\|x^{(s)} - x\|_2}{\|x^{(s-1)} - x\|_2} < c\mu \operatorname{Cond}(A), \qquad s = 2, 3, \ldots$$

where $c = c(m, n)$ is an error constant.

An Interpretation of the Result and Remarks

The preceding result tells us that the iterative refinement procedure is quite satisfactory. It is even more satisfactory for least-squares problems with large residuals. Note that for these problems, $(\text{Cond}(A))^2$ serves as the condition number. On the other hand, this result shows that error at an iterative refinement step depends upon the condition number of A. The procedure "may give solutions to full single precision accuracy even when the initial solution may have no correct significant figures" (Björck 1994b). For a well-conditioned matrix, the convergence may occur even in one iteration. Björck and Golub (1967) have shown that taking A as the last six columns of the inverse of the 8×8 Hilbert matrix, three-digit accuracy per step for both the solution and the residual can be obtained.

Example 7.10.1

$$A = \begin{pmatrix} 1 & 2 \\ 2 & 3 \\ 3 & 4 \end{pmatrix}, \qquad b = \begin{pmatrix} 3 \\ 5 \\ 9 \end{pmatrix}$$

$$r^{(0)} = 0, \qquad x^{(0)} = 0, \qquad k = 0$$

$$\begin{pmatrix} r_1^{(0)} \\ r_2^{(0)} \end{pmatrix} = \begin{pmatrix} b \\ 0 \end{pmatrix} = \begin{pmatrix} 3 \\ 5 \\ 9 \\ 0 \\ 0 \end{pmatrix}$$

Step 1: Solve the system:

$$\begin{pmatrix} I & A \\ A^T & 0 \end{pmatrix} \begin{pmatrix} c_1^{(0)} \\ c_2^{(0)} \end{pmatrix} = \begin{pmatrix} r_1^{(0)} \\ r_2^{(0)} \end{pmatrix}$$

$$\begin{pmatrix} c_1^{(0)} \\ c_2^{(0)} \end{pmatrix} = \begin{pmatrix} 0.3333 \\ -0.6667 \\ 0.3333 \\ 3.3333 \\ -0.3333 \end{pmatrix}$$

Step 2: Update the solution and the residual:

$$\begin{pmatrix} r^{(1)} \\ x^{(1)} \end{pmatrix} = \begin{pmatrix} r^{(0)} \\ x^{(0)} \end{pmatrix} + \begin{pmatrix} c_1^{(0)} \\ c_2^{(0)} \end{pmatrix} = \begin{pmatrix} 0.3333 \\ -0.6667 \\ 0.3333 \\ 3.3333 \\ -0.3333 \end{pmatrix}$$

The computations of $c_1^{(0)}$ and $c_2^{(0)}$ are as follows.

$$r_1^{(0)} = b, \; r_2^{(0)} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

$$\begin{pmatrix} r_1' \\ r_2' \end{pmatrix} = Q^T b = \begin{pmatrix} -10.6904 \\ -0.2182 \\ 0.8165 \end{pmatrix}$$

Thus,

$$r_1' = \begin{pmatrix} -10.6904 \\ 0.2182 \end{pmatrix}$$

$$r_2' = (0.8165)$$

$$c_2' = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

$$c_2^{(0)} = \begin{pmatrix} 3.3333 \\ -0.3333 \end{pmatrix}$$

$$c_1^{(0)} = \begin{pmatrix} 0.3333 \\ -0.6667 \\ 3.3333 \end{pmatrix}$$

Note that

$$x^{(1)} = \begin{pmatrix} 3.3333 \\ -0.3333 \end{pmatrix}$$

is the same least-squares solution as obtained by the QR and normal equation methods.

7.11 Computing the Variance-Covariance Matrix $(A^TA)^{-1}$

In statistical and image-processing applications we often need to compute the variance–covariance matrix

$$X = (A^TA)^{-1}$$

In order to see how this matrix arises in statistical analysis, consider the classical linear regression model:

$$b = Ax + \epsilon$$

with the properties

1. $E(\epsilon) = 0$; and
2. $\text{cov}(\epsilon) = E(\epsilon\epsilon^T) = \sigma^2 I$.

Here b is the **response vector**, A is the **design matrix**, and ϵ is the **error term**. The parameters x and σ^2 are unknown.

Regression analysis is concerned with prediction of one or more response (dependent) variables from the values of a group of predictor (independent) variables. Assuming that A has full rank, then the least-squares estimate of x in the preceding model is, clearly, given by

$$\hat{x} = (A^TA)^{-1}A^Tb$$

Also,

$$\hat{\epsilon} = b - \hat{b} = (I - A(A^TA)^{-1}A^T)b$$

(note that $A^T\hat{\epsilon} = 0$ and $\hat{b}^T\hat{\epsilon} = 0$). A common estimate of σ^2 is

$$\bar{\sigma}^2 = \frac{\|b - A\hat{x}\|_2^2}{m - n}$$

where A is $m \times n$. For details, see Johnson and Wichern (1988) and Björck (1994b).

Because vital information might be lost in computing A^TA explicitly, we wonder if it is possible to compute X from the QR factorization of A. We show how this can be done.

Consider the QR factorization of A:

$$Q^TA = \begin{pmatrix} \tilde{R} \\ 0 \end{pmatrix}$$

Then

$$A^TA = (\tilde{R})^T\tilde{R}$$

and, thus,

$$X = (A^TA)^{-1} = (\tilde{R})^{-1}(\tilde{R})^T$$

We can then easily compute $(\tilde{R})^{-1}$, because $\tilde{R}$ is an upper triangular matrix. Note that because X is symmetric, only one-half of its entries need to be computed, and X can overwrite R. However, computing $(A^TA)^{-1}$ this way will require explicit multiplication of $(\tilde{R})^{-1}(\tilde{R})^{-T}$.

This explicit multiplication can easily be avoided if the computations are reorganized. Thus, if x_1 through x_n are the successive columns of X, then from

$$X = (\tilde{R})^{-1}(\tilde{R})^{-T}$$

we have

$$\tilde{R}(x_1, \ldots, x_n) = (\tilde{R})^{-T}$$

Because the last column of $\tilde{R}^{-T}$ is just $(1/r_{nn})e_n$, the last column, x_n, is easily computed by solving the upper triangular system

$$\tilde{R}x_n = \frac{1}{r_{nn}}e_n \tag{7.11.1}$$

By symmetry we also get the last row of X.

Suppose now that we have already determined $x_{ij} = x_{ji}$, $j = n, \ldots, k+1$, $i \le j$. We now determine x_{ik}, $i \le k$.

For x_{kk} we have

$$x_{kk}r_{kk} + \sum_{j=k+1}^{n} r_{kj}x_{jk} = \frac{1}{r_{kk}}$$

from which we obtain

$$x_{kk} = \frac{1}{r_{kk}}\left(\frac{1}{r_{kk}} - \sum_{j=k+1}^{n} r_{kj}x_{kj}\right) \tag{7.11.2}$$

(Note that $x_{kj} = x_{jk}$, $j = k+1, \ldots, n$, have already been computed.) For $i = k-1, \ldots, 1$, we have

$$x_{ik} = -\frac{1}{r_{ii}}\left(\sum_{j=i+1}^{k} r_{ij}x_{jk} + \sum_{j=k+1}^{n} r_{ij}x_{kj}\right) \tag{7.11.3}$$

Thus, (7.11.1)–(7.11.3) determine all the entries of X.

ALGORITHM 7.11.1 Computing the Variance–Covariance Matrix $X = (A^TA)^{-1}$

Step 1: Compute the last column x_n by solving the system (7.11.1).

Step 2: Compute x_{kk} from (7.11.2) for $k = n-1, n-2, \ldots, 2, 1$

Step 3: Compute x_{ik} from (7.11.3), $i = k-1, \ldots, 1$; $k = 2, 3, \ldots, n$

Flop count. The computation of $X = (A^TA)^{-1}$ using these formulas requires about $n^3/3$ flops.

MATCOM Note: Algorithm 7.11.1 has been implemented in MATCOM program VARCOVAR.

Example 7.11.1

$$A = \begin{pmatrix} 1 & 5 & 7 \\ 2 & 3 & 4 \\ 1 & 1 & 1 \end{pmatrix}$$

$A = QR$ gives

$$Q = \begin{pmatrix} -0.4082 & 0.9045 & 0.1231 \\ -0.8165 & -0.3015 & -0.4924 \\ -0.4082 & -0.3015 & 0.8616 \end{pmatrix}$$

$$R = \begin{pmatrix} -2.4495 & -4.8990 & -6.5320 \\ 0 & 3.3166 & 4.8242 \\ 0 & 0 & -0.2462 \end{pmatrix}$$

The last column is

$$x_3 = \begin{pmatrix} 4 \\ -24 \\ 16.5 \end{pmatrix} \quad \text{(using (7.11.1))}$$

$$x_{22} = 35 \quad \text{(using (7.11.2))}$$

$$\left.\begin{array}{l} x_{12} = -6 \\ x_{11} = 1.5 \end{array}\right] \quad \text{(using (7.11.3))}$$

Then

$$X = (A^T A)^{-1} = \begin{pmatrix} 1.5 & -6 & 4 \\ -6 & 35 & -24 \\ 4 & -24 & 16.5 \end{pmatrix}$$

7.12 Review and Summary

1. *Existence and uniqueness:* The least-squares solution to the problem $Ax = b$ always exists. In the overdetermined case, it is unique if and only if A has full rank (Theorem 7.3.1).

2. *Overdetermined problems:* We discussed two methods: the normal equations method and the QR factorization method. The normal equations method is simple to implement, but there are numerical difficulties with this method in certain cases. The QR factorization methods are more expensive than the normal equations method but are more reliable numerically. If we insist on using normal equations, the use of extended precision is recommended. We can use Householder or Givens transformations or the modified Gram–Schmidt method to achieve the QR factorization of A needed for solution of the least-squares problem.

The Householder QR factorization is the most efficient and numerically viable; the modified Gram–Schmidt is only slightly more expensive than the Householder method and is not as reliable as the Householder or the Givens method numerically for QR factorization. However, with some reorganization it can be made competitive with the Householder method to compute the least-squares solution. Indeed, popularity of using the MGS method in solving the least-squares problem is growing. Some numerical experiments even suggest the superiority of this method over other QR methods (Björck 1994a or 1994b). Rice first noted the superiority of this algorithm by performing numerical experiments.

For the rank-deficient overdetermined problem, there are infinitely many solutions. Here we have discussed only the QR factorization method with column pivoting. It is reasonably accurate, but the *most numerically viable approach to deal with rank deficiency is the SVD approach, to be discussed in Chapter 10.*

3. *Underdetermined systems:* For the underdetermined problem there are infinitely many solutions in the full-rank case. We have discussed the normal equations method and the QR factorization method for the minimum-norm solution. The normal equations to be solved here are:

$$AA^T y = b; \qquad x = A^T y$$

whereas those of the overdetermined problem are

$$A^T A x = A^T b$$

The numerical difficulties with the normal equations are the same as those of the overdetermined problem.

The roundoff error analyses for the underdetermined and the rank-deficient overdetermined problems are somehow complicated, because solutions in these cases are not unique unless imposing the requirement of having the minimum-norm solution. The backward roundoff analyses of the QR factorization methods using Householder transformations in these cases show that the computed minimum-norm solution in each case is nearby the minimum-norm solution of a slightly perturbed problem. This is in contrast with the results obtained by the Householder–QR factorization method for the full-rank overdetermined problem, where it is shown that the computed solution is the exact solution of a slightly perturbed problem.

4. *Perturbation analysis:* The results of perturbation analyses are different for different cases of the perturbations in the data.

If only b is perturbed, then $\text{Cond}(A) = \|A\| \, \|A^\dagger\|$ serves as the condition number for the unique least-squares solution (Theorem 7.7.1).

If A is perturbed, then the sensitivity of the unique least-squares solution, in general, depends upon the square of the condition number (Theorem 7.7.2). In certain

TABLE 7.2 Comparisons for Different Least-Squares Methods

Problem	Method	Flop Count	Numerical Properties
Overdetermined Full rank	Normal Equations	$\frac{mn^2}{2} + \frac{n^3}{6}$	1. Difficulties exist with the formation of $A^T A$. 2. It produces more errors in the solution than there are warranted by data, in certain cases.
Overdetermined Full rank	Householder–QR	$mn^2 - \frac{n^3}{3}$	*Stable:* The computed solution is the exact solution of a nearby problem.
Overdetermined Full rank	MGS–QR	mn^2	It is as stable as the Householder–QR.
Overdetermined Rank deficient	Householder–QR with column pivoting	$2mnr - r^2(m+n) + \frac{2r^3}{3}$, where $r = \text{rank}(A)$	*Mildly stable:* The computed minimum-norm solution is close to the minimum-norm solution of a perturbed problem.
Underdetermined Full rank	Normal equations	$\frac{m^2 n}{2} + \frac{m^3}{6}$	The same difficulties exist as in the case of the overdetermined problem.
Underdetermined Full rank	Householder–QR	$m^2 n - \frac{m^3}{3}$	This has the same properties as the rank-deficient overdetermined problem.
Underdetermined Full rank	Modified Gram–Schmidt	$m^2 n$	This is as stable as the Householder–QR.

cases, such as when the residual is zero, the sensitivity depends only on the condition number of A.

5. *Iterative refinement:* As in the case of the linear system problem, it is possible to improve the accuracy of a computed least-squares solution in an iterative fashion. An algorithm that is a natural analog to the one for the linear system (Section 6.9) is satisfactory only when the residual vector $r = b - Ax$ is sufficiently small. A widely used algorithm due to Björck is presented in Section 7.10. This algorithm requires solution of an augmented system of order $m + n$ (where A is $m \times n$). It shows how to solve the system in a rather inexpensive way using QR factorization of A. The solution obtained by this iterative refinement algorithm is quite satisfactory.

6. *Computing the covariance matrix:* An efficient computation of the matrix $(A^T A)^{-1}$ is important in statistical and image processing applications. In Section 7.11, we show how to compute this matrix using QR factorization of A and without finding the inverse explicitly.

Table 7.2 (on the preceding page) compares the methods of this chapter.

EXERCISES

Section 7.3

1. Let A be $m \times n$, $m \geq n$. Then, using QR factorization of A, prove that $A^T A$ is positive definite if and only if A has rank n.
2. Show that the residual vector $r = b - Ax$ is orthogonal to all vectors in $R(A)$.
3. Prove that x is a least-squares solution if and only if

$$Ax = b_R \quad \text{and} \quad b - Ax = b_N$$

where b_R and b_N are, respectively, the range-space and column-space components of the vector b.

4. Prove Theorem 7.3.1, both in the full-rank and rank-deficient cases, based on QR factorization of A.

Sections 7.5–7.7

5. Let A have linearly independent columns. Then prove that $(\text{Cond}_2(A))^2 = \text{Cond}_2(A^T A)$.
6. Using least squares, fit a straight line and a quadratic to

x	0	1	3	5	7	9	12
y	10	12	18	15	20	25	36

Compare your results. Compute the condition number of the associated Vandermonde matrix in each case.

7. Find the generalized inverse and the condition number of each of the following matrices.

$$A = \begin{pmatrix} 0.0001 & 1 \\ 1 & 1 \end{pmatrix}, \qquad A = \begin{pmatrix} 1 & 1 \\ 0.0001 & 0 \\ 0 & 0.0001 \end{pmatrix}$$

$$A = \begin{pmatrix} 7 & 6.990 \\ 4 & 4 \end{pmatrix}, \qquad A = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 4 & 5 \\ 0 & 7 & 8 \end{pmatrix}$$

8. Let A and $\tilde{A}$ have full rank. Let x and $\tilde{x}$ be, respectively, the unique least-squares solutions to the problems

$$Ax = b$$

and

$$\tilde{A}\tilde{x} = b$$

where $\tilde{A} = A + E$. Then prove that

$$\frac{\|\tilde{x} - x\|}{\|\tilde{x}\|} \leq \text{Cond}(A)\frac{\|E\|}{\|A\|}\left(1 + \frac{\|b\|}{\|A\|\,\|\tilde{x}\|}\right) + (\text{Cond}(A))^2\frac{\|E\|}{\|A\|}\left(1 + \frac{\|E\|}{\|A\|}\right)$$

(*Hint*: Apply the perturbation analysis of the linear systems with normal equations.)

9. Verify the inequality of Exercise 8 with the following data:

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}, \qquad b = \begin{pmatrix} 3 \\ 7 \\ 11 \end{pmatrix}$$

$$E = 10^{-4}A$$

10. Verify the inequality of Theorem 7.7.4 in each of the following cases.

(a) $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}, E = 10^{-4}A$

(b) $A = \begin{pmatrix} 1 & 1 \\ 10^{-4} & 0 \\ 0 & 10^{-4} \end{pmatrix}, E = 10^{-4}A$

(c) $A = \begin{pmatrix} 1 & 1 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}, E = 10^{-3}A$

11. Work out a proof of Theorem 7.7.2.

Sections 7.7 and 7.8

12. Develop an algorithm to solve the normal equations whose system matrix is positive semidefinite and singular, based on *complete pivoting* of the system matrix. Apply your algorithm to the normal equations with

$$A = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \quad \text{and} \quad b = \begin{pmatrix} 2 \\ 2 \end{pmatrix}$$

13. Let $A = \begin{pmatrix} 1 & 1 \\ 2 & 3 \\ 0 & 1 \end{pmatrix}, \qquad b = \begin{pmatrix} 2 \\ 5 \\ 1 \end{pmatrix}$

(a) Find the unique least-squares solution x using

(i) $x = A^{\dagger}b$

(ii) The normal equations method

(iii) The Householder and the Givens QR factorization methods

(iv) The CGS and MGS methods

(b) Find Cond(A).

(c) Show that for this problem, the sensitivity of the least-squares problem, when only A is perturbed, depends upon Cond(A).

(d) Let $E = 10^{-4}A$ and let $\tilde{x} = \tilde{A}^{\dagger}b$, where $\tilde{A} = A + E$. Find $\dfrac{\|\tilde{x} - x\|}{\|x\|}$ and verify the inequality of Theorem 7.7.2 for this problem.

(e) Find r and $\hat{r}$ and verify the inequality of Theorem 7.7.3.

14. Construct your own example where the sensitivity of the least-squares problem will depend upon the square of the condition number of the matrix. (Show all your work.)

15. Consider the following well-known ill-conditioned matrix

$$A = \begin{pmatrix} 1 & 1 & 1 \\ \epsilon & 0 & 0 \\ 0 & \epsilon & 0 \\ 0 & 0 & \epsilon \end{pmatrix}, \qquad |\epsilon| \ll 1$$

(a) Choose an ϵ small, so that rank(A) = 3. Then compute Cond$_2(A)$ to check that A is ill-conditioned.

(b) Find the least-squares solution with A as given above and

$$b = \begin{pmatrix} 3 \\ \epsilon \\ \epsilon \\ \epsilon \end{pmatrix}$$

using

(i) The normal equations method

(ii) The QR factorization method with the Householder, CGS, and MGS methods

(c) Change b to

$$\tilde{b} = \begin{pmatrix} 3 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Keep A unchanged. Find an upper bound for the relative error in the least-squares solution.

(d) Change A to $\tilde{A} = A + E$, where $E = 10^{-3}A$. Keep b unchanged. Find an upper bound for the relative error in the least-squares solution.

(e) Find the maximum departure from orthogonality of the computed columns of the Q matrix using the CGS and MGS methods.

(f) Compute the least-squares solution of the problem in (b) using the SVD.

16. (Square Root–Free Cholesky) Given a symmetric positive definite matrix A, develop an algorithm for finding the Cholesky decomposition of A without any square roots:

$$A = LDL^T$$

where L is a unit lower triangular matrix and D is a diagonal matrix with positive diagonal entries. Apply your algorithm to solve the full-rank least-squares problem based on solving normal equations.

17. (a) Derive a column-oriented version of the modified Gram–Schmidt process.

(b) Show that the flop count for both the Gram–Schmidt and the modified Gram–Schmidt are mn^2, where A is $m \times n$, $(m \geq n)$.

(c) Show that the flop count for the Householder method is $n^2(m - n/3)$.

18. (a) Construct an example of the least-squares problem where Gram–Schmidt will yield a poor result but modified Gram–Schmidt will do fairly well.

(b) Apply the CGS and MGS methods of QR factorization to the matrix

$$A = \begin{pmatrix} 8 & 21 \\ 13 & 34 \\ 21 & 35 \\ 34 & 39 \end{pmatrix}$$

using $t = 4$. Show that in both cases the loss of orthogonality between q_1 and q_2 is unacceptable.

19. Show that the flop count for QR factorization with column pivoting using the Householder method is $2mnr - r^2(m + n) + 2r^3/3$, where $r = \text{rank}(A)$.

20. Show that for the QR method with column pivoting in the rank-deficient case, the basic solution cannot be the minimum-norm solution unless R_{12} is zero.

21. (a) Show that the minimum-norm solution to $Ax = b$, with rank(A) = r, obtained by complete QR factorization with column pivoting is given by

$$x = PW\begin{pmatrix} T^{-1}c \\ 0 \end{pmatrix}$$

where c is a vector consisting of the first r elements of Q^Tb.

(b) Find the minimum norm solution to the least-squares problem with

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}, \qquad b = \begin{pmatrix} 3 \\ 3 \\ 3 \end{pmatrix}$$

Section 7.9

22. Develop an algorithm based on the QR factorization with column pivoting to compute the minimum norm solution to the underdetermined system

$$Ax = b$$

where A is $m \times n$, $m < n$. Give a flop count for the algorithm. Apply your algorithm to

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 6 \\ 15 \end{pmatrix}$$

23. Develop an algorithm, based on complete pivoting factorization of A, to solve the underdetermined system $Ax = b$, where A is $m \times n$, $m < n$. Your algorithm should compute a solution when it exists, or else indicate that a solution does not exist.
    (a) Give the flop count for your algorithm.
    (b) Test your algorithm with data of Problem 22.
24. Using Theorem 7.9.1, prove that the minimum norm solution to an underdetermined system can be obtained by projecting any solution to the system onto $R(A^\dagger)$. That is, if $P_A^\dagger$ is the orthogonal projection onto $R(A^\dagger)$, then the minimum norm solution x is given by

$$x = P_A^\dagger y$$

where y is any solution. Using the above formula, compute the minimum norm solution to the system

$$\begin{pmatrix} 1 & 10^{-4} & 0 & 0 \\ 1 & 0 & 10^{-4} & 0 \\ 1 & 0 & 0 & 10^{-4} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

25. Give a geometric interpretation of Theorem 7.9.1.

Section 7.10

26. (a) Apply three iterations of Algorithm 7.10.1 to each of the following problems:

$$\text{(i)}\ A = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}, \quad b = \begin{pmatrix} 3 \\ 5 \\ 9 \end{pmatrix}$$

$$\text{(ii)}\ A = \begin{pmatrix} 1 & 1 \\ 10^{-4} & 0 \\ 0 & 10^{-4} \end{pmatrix}, \quad b = \begin{pmatrix} 2 \\ 10^{-4} \\ 10^{-4} \end{pmatrix}$$

$$\text{(iii)}\ A = \begin{pmatrix} 1 & 1 \\ 10^{-4} & 0 \\ 0 & 10^{-4} \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

    (b) What is the relationship of Algorithm 7.10.1 with Algorithm 7.10.2?
27. Apply Algorithm 7.10.2 to each part of Exercise 26, with and without explicitly solving the augmented system in step 2, and compare the results.
28. Apply Algorithm 7.10.2 to the least-squares problem with the 7×7 Hilbert matrix and $b = (2.5929, 1.7179, 1.3290, 1.0956, 0.9365, 0.8199, 0.7301)^T$.
    Tell how many digits of accuracy per iteration step both in the solution and the residual were obtained. Note that $x = (1, 1, \ldots, 1)^T$.

Section 7.11

29. In many applications, only the diagonal entries of the variance–covariance matrix $X = (A^T A)^{-1}$ are needed. Show that these diagonal entries are just the squares of the 2-norms of the rows of R^{-1}.

MATLAB PROGRAMS AND PROBLEMS

Note: You will need the programs **housqr**, **givqr**, **clgrsch**, **mdgrsch**, **lsfrqrh**, **lsfrmgs**, **lsfrnme**, **mnudnme**, **mnudqrh**, **lsitrn1**, **lsitrn2**, **varcovar**, and **backsub** from MATCOM.

1. Consider the following set of data points:

x	0	1	2	3	4	5	6	7	8	9
y	1	2.9	6.8	12	20.5	30.9	42.9	51.5	73	90.5

   Using the MATLAB command **vander** and the operation '\', compute the least-square fit of the data to polynomials of degree 1 through 4. Plot the original data point and the least-squares fits using the MATLAB command **plot** and **polyval**, and compare the results.
2. Let

$$A = \begin{pmatrix} 1 & 1 \\ 10^{-3} & 0 \\ 0 & 10^{-3} \\ 0 & 0 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

$$E_A = \begin{pmatrix} 0 & -0.0001 \\ 0 & 0.0009 \\ 0 & 0.0003 \\ 0 & 0.0001 \end{pmatrix}, \quad \delta b = \begin{pmatrix} 0.0001 \\ 0.0001 \\ 0.0001 \\ 0.0001 \end{pmatrix}$$

   Using MATLAB commands such as **pinv**, **cond**, **norm**, **orth**, and **null**, verify the inequalities of Theorems 7.7.1–7.7.4 on different sensitivities of least-squares problems.
3. (Implementation of the Least-Squares QR Algorithm Using Givens Rotations) Using **givqr** and **backsub**, write a MATLAB program to implement the QR algorithm using Givens rotations for the full-rank overdetermined least-squares problem:

$$[x] = \textbf{lsfrqrg}(A, b)$$

   Test your program with $A = \text{hilb}(7)$, and create a vector b so that the least-squares solution $x = (1, 1, \ldots, 1)^T$.

Test Data for Problems 4 and 5

For Problems 4 and 5 use the following sets of test data.

1. A randomly generated matrix of order 10
2. Hilbert matrix of order 10
3. $\begin{pmatrix} 1 & 1 & 1 \\ 10^{-3} & 0 & 0 \\ 0 & 10^{-3} & 0 \\ 0 & 0 & 10^{-3} \end{pmatrix}$

(*continued*)

For Problem 5, generate b so that the least-squares solution x in each case has all entries equal to 1. For example, for data matrix in test data 3,

$$b = \begin{pmatrix} 3 \\ 10^{-3} \\ 10^{-3} \\ 10^{-3} \end{pmatrix}$$

4. (The purpose of this exercise is to compare the accuracy, and flop count, and timing for different methods for QR factorization of a matrix.)
   (a) Compute the QR factorization for each matrix A of the data set using
      (i) $[Q, R] = qr(A)$ from MATLAB or $[Q, R] =$ **housqr**(A) from Chapter 5 (Householder QR)
      (ii) $[Q, R] =$ **givqr**(A) (Givens QR)
      (iii) $[Q, R] =$ **clgrsch**(A) from MATCOM (classical Gram–Schmidt)
      (iv) $[Q, R] =$ **mdgrsch**(A) from MATCOM (modified Gram–Schmidt)
   (b) Using the results of (a), make a table like Table 7.3 for each matrix A. $\hat{Q}$ and $\hat{R}$ stand for the computed Q and R.
   (c) Write your conclusions.
5. (The purpose of this exercise is to compare the accuracy, flop count, and timing for different least-squares methods for full-rank overdetermined problems.)
   (a) Compute the least-squares solution $\hat{x}$ for each data set using each of the following.
      (i) $[x] =$ **lsfrmgs**(A, b) (least squares using modified Gram–Schmidt)
      (ii) $[x] =$ **lsfrqrh**(A, b) (least squares using Householder QR)
      (iii) $[x] =$ **lsfrqrg**(A, b) (least squares using Givens QR)
      (iv) $[x] =$ **lsfrnme**(A, b) (least squares using normal equations)
      (v) $[x] =$ **pinv**(A) * b (least squares using generalized inverse)

Note: **lsfrmgs**, **lsfrqrh**, and **lsfrnme** are all available in MATCOM. **pinv** is a MATLAB command for computing the generalized inverse of a matrix.

   (b) Using the results of (a), make one table like Table 7.4 for each data set. ($\hat{x}$ stands for computed x.) Note also that the vector x has all entries equal to 1.
   (c) Write your conclusions.

TABLE 7.3 Comparison of Different QR Factorization Methods

Method	$\|(\hat{Q})^T\hat{Q} - I\|_2$	$\dfrac{\|A - \hat{Q}\hat{R}\|_2}{\|A\|_2}$	Flop Count	Elapsed Time
housqr				
givqr				
clgrsch				
mdgrsch				

TABLE 7.4 Comparison of Different Methods for Full-Rank Overdetermined Least-Squares Problem

Method	$\|x - \hat{x}\|_2/\|x\|_2$	$\|A\hat{x} - b\|_2$	Flop Count	Elapsed Time
lsfrmgs				
lsfrqrh				
lsfrqrg				
lsfrnme				
Generalized inverse				

6. Using **housqr** from MATCOM or $[Q, R] = qr(A)$ from MATLAB and **backsub** from Chapter 6, write a MATLAB program called **lsrdqrh**(A, b) to compute the minimum-norm least-squares solution x to the rank-deficient overdetermined problem $Ax = b$, and the corresponding residual r, based on the Householder QR factorization of A:

   $[x, r] =$ **lsrdqrh**(A, b)

   (This program should implement Algorithm 7.8.6 of the book.)

Test Data: A is a 20×2 matrix with all entries equal to 1 and b is a vector with all entries equal to 2.

7. Using the MATLAB function

   $[U, S, V] =$ **svd**(A)

   to compute the singular-value decomposition of A, write a MATLAB program called **lsrdsvd** to compute the minimum-norm least-squares solution x to the rank-deficient overdetermined system $Ax = b$:

   $[x] =$ **lsrdsvd**(A, b)

   Use the same test data as in Problem 6 and compare the results with respect to accuracy, flop count, and elapsed time.
8. Run the programs **mnudnme** (minimum norm solution to the underdetermined full-rank problem using normal equations) and **mnudqrh** (minimum norm solution to the underdetermined full-rank problem using Householder QR factorization) from MATCOM on the following sets of data to compute the minimum-norm solution x to the full-rank underdetermined problem $Ax = b$, and compare the results with respect to accuracy, elapsed time, and flop count.

$$A = (1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6)$$

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{pmatrix}$$

$$A = \begin{pmatrix} 10 & 10 & 10 & 10 & 10 & 10 & 10 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Construct b for each A so that the minimum-norm solution x has all its entries equal to 1.

9. Run the programs **lsitrn1** (based on Algorithm 7.10.1) and **lsitrn2** (based on Algorithm 7.10.2) from MATCOM on the 8×8 Hilbert matrix A, constructing b randomly. Compare the algorithms on this data with respect to number of iterations, flop count, and elapsed time, both for the solution and the residual.
10. (a) Compute $(A^T A)^{-1}$ for each of the following matrices A as follows:
    (i) Compute explicitly $(A^T A)^{-1}$ using MATLAB command **inv**.
    (ii) Run the program **varcovar** from MATCOM.
    (b) Compare the results with respect to accuracy, flop count, and elapsed time.

Test data:

$A = 8 \times 8$ Hilbert matrix

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 10^{-3} & 0 & 0 \\ 0 & 10^{-3} & 0 \\ 0 & 0 & 10^{-3} \end{pmatrix}$$

$A = 8 \times 8$ randomly generated matrix

Suggestions for Further Reading

The techniques of least-squares solutions are covered in any numerical linear algebra text and in some numerical analysis texts. The emphasis in most books is on the overdetermined problems. For a thorough treatment of the subject we refer the readers to the book by Golub and Van Loan (1989). The books by Stewart (1973), by Watkins (1991), and by Gil, Murray, and Wright (1991) also contain detailed discussions on perturbation analyses. See also a paper of Stewart (1977a) and the book by Stewart and Sun (1990). For componentwise perturbation analysis, see Björck (1994b) and Chandrasekaran and Ipsen (1994b). For perturbation analysis of the underdetermined problems see Golub and Van Loan (1989, 258–259). For applications of Gaussian elimination in the solution of least-squares problems, see Plemmons (1974).

For applications of least-squares problems to numerical solutions of systems of differential equations, see Campbell (1985).

For more on underdetermined problems, see the papers by Cline and Plemmons (1976) and Arioli and Laratta (1985).

A complete book devoted to the subject is the (almost classical) book by C. L. Lawson and R. J. Hanson (1974). There is a nice recent monograph by Björck (1994b) that contains a thorough discussion on least-squares problems. These two books are essential reading for those interested in further study on the subject. The

book by Lawson and Hanson, in particular, gives the proofs of the roundoff error analyses of the various algorithms described in the present book.

Any book on regression analysis in statistics will contain applications of least-squares problems in statistics. We have, in particular, used the book *Applied Linear Regression Models*, by John Neter, William Wasserman, and Michael Kunter (1983). A classical survey paper of Golub (1969) contains an excellent exposition of the numerical linear algebra techniques for the least-squares and the singular value decomposition problems arising in statistics and elsewhere. A paper by Stewart (1987) is also interesting to read. See also the book by Thisted (1988), and a paper by Efron (1988) for computational aspects of statistical regression.

It is highly recommended that the readers consider these papers and other books representing the most fundamental contributions in this area. For an extensive set of references, see the bibliography of the recent monograph of Björck (1994b).

Several important aspects of least-squares problems have not been covered in this book. They include the *total least-squares problems*, *constrained least-squares problems*, and *sparse least-squares problems*. See Björck (1994b) for these topics. The book *The Total Least Squares Problems: Computational Aspects and Analysis* by S. Van Huffel and J. Vandewalle, *Frontier in Applied Mathematics* series SIAM, Philadelphia, 1991, is also suggested for further reading on total least-squares problems. See also a recent paper by Fierro and Bunch (1994) on collinearity and total least squares.

Math. Seminal

CHAPTER EIGHT

Numerical Matrix Eigenvalue Problems

8.1 Introduction

This chapter is devoted to the study of numerical matrix eigenvalue problems. These problems are very important practical problems and arise in a variety of applications, including engineering, statistics, and economics.

Because the eigenvalues of a matrix A are the zeros of the characteristic polynomial $\det(A - \lambda I)$, one would naively think of computing the eigenvalues of A by finding its characteristic polynomial and then computing its zeros by a standard root-finding method. Unfortunately, this is not a practical approach.

A standard practical algorithm for finding the eigenvalues of a matrix is the **QR iteration method** with a single or double shift. Several applications do not need the knowledge of the whole spectrum. A few selected eigenvalues, usually a few largest or smallest ones, suffice. A classical method, based on implicit powering of A, known as the power method is useful for this purpose.

The symmetric eigenvalue problem enjoys certain remarkable special properties; one such property is that *the eigenvalues of a symmetric matrix are well conditioned.* A method based on the exploitation of some of the special properties of the symmetric eigenvalue problem, known as the **Sturm sequence-bisection method**, is useful for finding eigenvalues of a symmetric matrix in an interval (note that the eigenvalues of a symmetric matrix are all real).

The QR iteration method, unfortunately, cannot be used to compute the eigenvalues and eigenvectors of a very large and sparse matrix. The sparsity gets lost, and the storage becomes an issue. An almost classical method by Lanczos has been rejuvenated by numerical linear algebraists recently. The **symmetric Lanczos method** is useful in particular to extract a few extremal eigenvalues and the associated eigenvectors of a very large and sparse matrix.

The organization of this chapter is as follows. Some of the basic theoretical results on eigenvalues and eigenvectors are stated (and proved sometimes) in Section 8.2.

Section 8.3 is devoted to the discussions of how the eigenvalue problem arises in some practical applications, such as stability analyses of a system of differential and difference equations, vibration analysis, transient behavior of an electrical circuit, buckling problems, principal component analysis in statistics, and so on.

In Section 8.4 some classical results on eigenvalue locations, such as Geršgorin's disk theorems, are stated and proved. Section 8.5 describes the power method, the inverse power method, the Rayleigh–Quotient iteration, and so on, for finding a selected number of eigenvalues and the corresponding eigenvectors.

In Section 8.6 the difficulties of computing the eigenvalues of a matrix via the characteristic polynomial and the Jordan canonical form are highlighted.

The eigenvalue and eigenvector sensitivity are discussed in Sections 8.7 and 8.8. The most important result in this section is the Bauer–Fike theorem. Section 8.9 is the most important section of this chapter. The QR iteration method with and without shifts and their implementations are described in this section.

The Hessenberg inverse iteration and computations of eigenvectors from the real Schur form are described in Section 8.10. The methods based on specific properties of symmetric matrices are discussed in Section 8.11. These include the Sturm sequence-bisection method and symmetric QR iteration method with the Wilkinson shift, and the divide-and-conquer method.

The chapter concludes with a brief introduction of the symmetric Lanczos algorithm for finding the extremal eigenvalues of a large and sparse symmetric matrix.

The following major tools and concepts developed earlier will be needed for smooth learning and comprehension of this chapter's material: norm properties of matrices (Section 1.7), the QR factorization of an arbitrary and a Hessenberg matrix using both Householder and Givens matrices (Sections 5.4.1 and 5.5.1), solutions of linear systems (Section 6.4), the condition number and properties (Sections 6.6 and 6.7), and reduction to Hessenberg and Tridiagonal forms (Sections 5.4.3 and 5.5.3).

8.2 Some Basic Results Involving Eigenvalues and Eigenvectors

So far we have been dealing with real vectors and matrices. In most practical applications the data are real. However, a real matrix can have complex eigenvalues and complex eigenvectors. This chapter and the next two chapters will deal with eigenvalues and eigenvectors. We, therefore, quickly introduce the concepts of complex vectors and the complex space, and some associated definitions.

A complex n-vector x is an ordered arrangement of n complex numbers in a column:

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

The set of all complex n-vectors is denoted by $\mathbb{C}^n$. Fortunately, most of the concepts defined for $\mathbb{R}^n$ in Chapter 1 generalize very easily to $\mathbb{C}^n$.

A **complex $m \times n$ matrix** is a rectangular array of m rows and n columns. The set of all $m \times n$ matrices is denoted by $\mathbb{C}^{m \times n}$. The **complex conjugate** of a complex matrix A, denoted by $\bar{A}$, is the matrix whose each entry is the complex conjugate of the corresponding entry of A.

The transpose of the complex conjugate of A is denoted by A^*; that is,

$$A^* = (\bar{A})^T$$

$\bar{x}$, and x^* of a complex vector x are similarly defined. A square complex matrix A is **Hermitian** if $A^* = A$. A square complex matrix A is **unitary** if $A^*A = AA^* = I$. A Hermitian matrix A is **positive definite** if $x^*Ax > 0$ for all nonzero vectors x. It is **positive semidefinite** if $x^*Ax \geq 0$ for all $x \neq 0$.

With these basic definitions, we proceed to state (and prove sometimes) some basic results on eigenvalues and eigenvectors. Some of these results and definitions are in Chapter 1. Some will be repeated here for the sake of convenience and immediate reference. Keeping applications in mind, we will be discussing the methods for computing eigenvalues and eigenvectors of real matrices only.

8.2.1 Eigenvalues and Eigenvectors

Let A be an $n \times n$ matrix. Then λ is an **eigenvalue** of A if there exists a nonzero vector x such that

$$Ax = \lambda x \quad \text{right}$$

or

$$(A - \lambda I)x = 0$$

The vector x is a **right eigenvector** (or just an eigenvector) of A associated with the eigenvalue λ. (λ, x) will be called an eigenpair of A. The vector y given by

$$y^T A = \lambda y^T \quad \text{left}$$

is called a **left eigenvector** of A associated with the eigenvalue λ. It is customary to call a right eigenvector just an eigenvector.

The homogeneous system $(A - \lambda I)x = 0$ has a nontrivial solution if and only if

$$\det(A - \lambda I) = 0$$

$P_A(\lambda) = \det(A - \lambda I)$ is a polynomial in λ of degree n and is called the **characteristic polynomial of** A. Thus *the n eigenvalues of A are the n roots of its characteristic polynomial.*

The sum of the eigenvalues of a matrix A is called the *trace* of A. It is denoted by **trace** (A) or **Tr**(A).

Recall also from Chapter 1 that if $A = (a_{ij})$, then $\text{trace}(A) = a_{11} + \cdots + a_{nn}$ = sum of the diagonal entries of A.

The equivalence of these two definitions follows by considering the coefficient of λ in the characteristic polynomial $\det(A - \lambda I)$.

Observations

From the definitions of eigenvalues and eigenvectors, we immediately observe the following:

1. If (λ, x) is an eigenpair of a nonsingular matrix A, then $(\frac{1}{\lambda}, x)$ is the corresponding eigenpair of A^{-1}.
2. If (λ, x) is an eigenpair of A, then $(\lambda - \sigma, x)$ is an eigenpair of $A - \sigma I$, where σ is a scalar.

DEFINITION 8.2.1 The set of eigenvalues of a matrix A is called the **spectrum** of A.

We now present some basic results on eigenvalues and eigenvectors.

THEOREM 8.2.1 The eigenvalues of a triangular matrix are its diagonal entries.

Proof: If $A = (a_{ij})$ is triangular, then the matrix $(A - \lambda I)$ is also triangular, with $a_{11} - \lambda, a_{22} - \lambda, \ldots, a_{nn} - \lambda$ as its diagonal entries. So,

$$\det(A - \lambda I) = \prod_{i=1}^{n} (a_{ii} - \lambda)$$

This shows that the eigenvalues of A are $a_{11}, a_{22}, \ldots, a_{nn}$. ■

Theorem 8.2.1 can be easily generalized.

THEOREM 8.2.2 Let

$$A = \begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1k} \\ & A_{22} & \cdots & A_{2k} \\ & 0 & \ddots & \vdots \\ & & & A_{kk} \end{pmatrix}$$

where each A_{ii} is a square matrix. Then the spectrum of A is equal to the union of the spectrum of $A_{11}, \ldots, A_{kk}$.

Proof: We have

$$\det(A - \lambda I) = \det(A_{11} - \lambda I)\det(A_{22} - \lambda I)\ldots\det(A_{kk} - \lambda I)$$

Thus, the set of the zeros of the characteristic polynomial of A is equal to the union of the sets of the zeros of the characteristic polynomials of $A_{11}, \ldots, A_{kk}$. ■

THEOREM 8.2.3 Two similar matrices have the same eigenvalues. That is, the eigenvalues of A and TAT^{-1} are identical for all nonsingular matrices T.

Proof: We show that A and TAT^{-1} have the same characteristic polynomial, and, therefore, have the same eigenvalues.

$$\begin{aligned}\det(TAT^{-1} - \lambda I) &= \det(T(A - \lambda I)T^{-1}) \\ &= \det T \cdot \det(A - \lambda I) \cdot \det(T^{-1}) \\ &= \det T \cdot \det(T^{-1}) \cdot \det(A - \lambda I) \\ &= \det(T \cdot T^{-1}) \cdot \det(A - \lambda I) \\ &= \det(A - \lambda I)\end{aligned}$$

Thus, TAT^{-1} and A have the same characteristic polynomial and, therefore, have the same eigenvalues. ■

Note: The converse is *not* true. That is, two matrices having the same eigenvalues are not necessarily similar. Here is a simple example. Take

$$A = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \quad B = I_{2\times 2} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

A and B have the same eigenvalues, but they cannot be similar, because, $TBT^{-1} = B$ for all nonsingular matrices T.

THEOREM 8.2.4 The eigenvectors associated with the distinct eigenvalues of a matrix are linearly independent.

Proof: We prove the theorem by induction on m. Let $\lambda_1, \lambda_2, \ldots, \lambda_m$ be the m distinct eigenvalues of A and $x_1, x_2, \ldots, x_m$ be the corresponding eigenvectors. Assume that theorem holds for $m - 1$. We then prove that it also holds for m. Consider

$$c_1x_1 + c_2x_2 + \cdots + c_mx_m = 0$$

where c_i, $i = 1, \ldots, m$ are constants. We then show that $c_1 = c_2 = \cdots = c_m = 0$.

Multiplying by $(\lambda_1 I - A) \cdots (\lambda_{m-1} I - A)$ to the left, we obtain

$$(\lambda_1 I - A) \cdots (\lambda_{m-1} I - A)(c_1x_1 + \cdots + c_mx_m) = 0 \tag{8.2.1}$$

Because the matrices $(\lambda_i I - A)$ and $(\lambda_j I - A)$ commute with each other, we have for $i = 1, \cdots, m - 1$

$$(\lambda_1 I - A) \cdots (\lambda_{m-1} I - A)c_ix_i = c_i \prod_{\substack{j=1 \\ i\neq j}}^{m-1} (\lambda_j I - A)(\lambda_i I - A)x_i = 0 \qquad (\text{since } \lambda_i x_i = Ax_i)$$

Thus (8.2.1) becomes

$$(\lambda_1 I - A)(\lambda_2 I - A) \cdots (\lambda_{m-1} I - A)c_mx_m = 0$$

or

$$c_m(\lambda_1 - \lambda_m)(\lambda_2 - \lambda_m) \cdots (\lambda_{m-1} - \lambda_m)x_m = 0$$

or $c_m = 0$, because $\lambda_i - \lambda_m \neq 0$, $i = 1, \ldots, m - 1$. By induction hypothesis, $c_1x_1 + \cdots + c_{m-1}x_{m-1} = 0$ implies that $c_1 = c_2 = \cdots = c_{m-1} = 0$. ■

THEOREM 8.2.5 An arbitrary matrix A is similar to a diagonal matrix D if and only if A has a complete set of n linearly independent eigenvectors.

Proof: Let $x_1, \ldots, x_n$ be the eigenvectors corresponding to the eigenvalues $\lambda_1, \ldots, \lambda_n$. Define

$$X = (x_1, \ldots, x_n)$$

Then,

$$\begin{aligned}AX &= A(x_1, \ldots, x_n) = (\lambda_1x_1, \ldots, \lambda_nx_n) \\ &= (x_1, \ldots, x_n)\begin{pmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_n \end{pmatrix} = XD\end{aligned}$$

where

$$D = \operatorname{diag}(\lambda_1, \ldots, \lambda_n)$$

Since $x_1, \ldots, x_n$ are linearly independent, the matrix X is nonsingular. So,

$$X^{-1}AX = D$$

The converse can be proved by reversing the arguments. ■

From Theorems 8.2.4 and 8.2.5, we immediately have the following result.

COROLLARY 8.2.1 If all the eigenvalues of A are distinct, then A is similar to a diagonal matrix.

Note that the converse does *not* hold. An identity matrix is a diagonal matrix but its eigenvalues are not distinct.

Remark: Recall that an $n \times n$ matrix A is called nondefective if it has n linearly independent eigenvectors. Thus, Theorem 8.2.5 can be restated as follows:
A matrix A is nondefective if and only if it is similar to a diagonal matrix.

8.2.2 The Schur Triangularization Theorem and Its Applications

THEOREM 8.2.6 For any complex matrix A there exists a unitary matrix U such that

$$U^*AU = T$$

is an upper triangular matrix. The eigenvalues of A are the diagonal entries of T.

The theorem is of significant theoretical importance. As we shall see, several important results on eigenvalues and eigenvectors can be derived using this theorem. We give a proof of this theorem later in the chapter (Theorem 8.9.1).

THEOREM 8.2.7 Let $\lambda_1, \ldots, \lambda_n$ be the eigenvalues of a matrix A. Then the eigenvalues of A^m are $\lambda_1^m, \lambda_2^m, \ldots, \lambda_n^m$. In general, if $p(x)$ is a polynomial of degree n,

$$p(x) = c_0x^n + c_1x^{n-1} + \cdots + c_n$$

then the eigenvalues of $p(A)$ are $p(\lambda_1), p(\lambda_2), \ldots, p(\lambda_n)$.

Proof: We prove the first part only. The second part is left as an exercise. From $U^*AU = T$ we have

$$T^m = (U^*AU)(U^*AU)\cdots(U^*AU) = U^*A(UU^*)A(UU^*)\cdots(UU^*)AU$$

$$(m \text{ times})$$

$$= U^*A^mU$$

But T^m is also a triangular matrix with $\lambda_1^m, \ldots, \lambda_n^m$ as its diagonal entries. Because A^m and T^m are similar, they have the same eigenvalues. ■

THEOREM 8.2.8 Let A be a Hermitian matrix. Then

1. There exists an unitary matrix U such that

$$U^*AU = D$$

is a diagonal matrix.
2. The eigenvalues of A are real.
3. The eigenvectors of A may be chosen to be orthonormal.

Proof of 1: By the Schur triangularization theorem we have

$$U^*AU = T$$

where T is triangular. We now show that T is diagonal. We have

$$(U^*AU)^* = T^*$$

or

$$U^*A^*U = T^*$$

or

$$U^*AU = T^* \qquad (\text{since } A^* = A)$$

or

$$T = T^*$$

This shows that T is Hermitian. Because T is triangular and Hermitian, it must be a diagonal matrix.

Proof of 2: Because the eigenvalues of A are the diagonal entries of T and *the diagonal entries of a Hermitian matrix must be real*, it follows that the eigenvalues of A are real.

Remark: Note that if A is real, then $A^* = A$ implies that A is symmetric ($A^T = A$). *The eigenvalues of a real symmetric matrix are also real.*

Proof of 3: Denote the columns of U by u_1 through u_n. Then from

$$U^*AU = D$$

or

$$AU = UD$$

or

$$A(u_1, u_2, \ldots, u_n) = (u_1, u_2, \ldots, u_n) \cdot \operatorname{diag}(d_1, d_2, \ldots, d_n)$$

we see that u_i is an eigenvector of A corresponding to the eigenvalue d_i. Since U is unitary, the eigenvectors $u_1, u_2, \ldots, u_n$ are orthonormal. ■

Let A be real and symmetric, then, because its eigenvalues are real and eigenvectors can be chosen to be real, the unitary matrix U in Theorem 8.2.8 can be orthogonal.

COROLLARY 8.2.2 If A is a real symmetric matrix, then there is an orthogonal matrix U such that $U^TAU = D$, where D is diagonal.

THEOREM 8.2.9 The eigenvalues of a Hermitian positive definite matrix are positive; conversely, if a Hermitian matrix has all its eigenvalues positive, it must be positive definite.

Proof: Let (λ, x) be an eigenpair of A. Then from $Ax = \lambda x$, we have $x^*Ax = \lambda x^*x$. Because A is Hermitian, λ is real. Because A is positive definite, $x^*Ax > 0$. Also, since $x \neq 0$,

$$x^*x = |x_1|^2 + \cdots + |x_n|^2 > 0$$

where

$$x = (x_1, x_2, \ldots, x_n)^T$$

Thus,

$$\lambda = \frac{x^*Ax}{x^*x} > 0$$

To prove the converse, we note that, since A is Hermitian, by Theorem 8.2.8 we have

$$U^*AU = D = \operatorname{diag}(\lambda_1, \ldots, \lambda_n)$$

Define for any nonzero vector x, $y = U^*x$. Then $y \neq 0$. Now

$$x^*Ax = x^*UDU^*x = y^*Dy = \Sigma\lambda_i|y_i|^2 > 0$$

for any $x \neq 0$, which proves that A is positive definite. ■

8.2.3 The Jordan Canonical Form

We have seen that an arbitrary $n \times n$ matrix A is similar to a diagonal matrix if and only if it has a complete set of n linearly independent eigenvectors. Thus, in particular, a Hermitian matrix, a real symmetric matrix, and a matrix having distinct eigenvalues are similar to a diagonal matrix. That is, these matrices are nondefective. The next theorem shows that an arbitrary matrix is always similar to a block diagonal matrix.

THEOREM 8.2.10 The Jordan Canonical Form For an $n \times n$ matrix A, there exists a nonsingular matrix X such that
(arbitrary)

$$X^{-1}AX = \begin{pmatrix} J_1 & & & 0 \\ & \cdot & & \\ & & \cdot & \\ 0 & & & \cdot \\ & & & & J_m \end{pmatrix}$$

Jordan canonical form

where

$$J_i = \begin{pmatrix} \lambda_i & 1 & & & 0 \\ & \lambda_i & 1 & & \\ & & \cdot & \cdot & \\ & & & \cdot & \cdot \\ 0 & & & & 1 \\ & & & & \lambda_i \end{pmatrix}$$

Jordan block

where the λ_i are eigenvalues of A.

Proof: See Horn and Johnson (1985, pp. 121–129). ■

The preceding diagonal block matrix is called the **Jordan canonical form** of A. The matrices $J_1, \ldots, J_m$ are Jordan matrices. If J_i is of degree v_i, then

$$v_1 + \cdots + v_m = n$$

DEFINITION 8.2.2 The polynomials

$$\det(J_i - \lambda I) = (\lambda_i - \lambda)^{v_i}$$

are called the **elementary divisors** of A. If $v_1 = v_2 = \cdots = v_m = 1$, then the elementary divisors are called **linear elementary divisors**.

If the ith column of X is denoted by x_i then for the vectors $x_1, x_2, \ldots, x_{v_1}$, we have

$$Ax_1 = \lambda_1 x_1$$
$$Ax_{i+1} = \lambda_1 x_{i+1} + x_i, \quad i = 1, 2, \ldots, v_1 - 1$$

Similar relations hold for other vectors. These vectors are called **principal vectors** or the **generalized eigenvectors** of A.

Notes:

1. When all the eigenvalues of A are distinct, each Jordan block has order 1; therefore, elementary divisors are linear. In fact, if there are m Jordan blocks, there are m linearly independent eigenvectors.
2. If there are some nonlinear elementary divisors, then A has eigenvalues whose multiplicities are higher than the number of independent eigenvectors associated with them.
3. For a real symmetric or complex Hermitian matrix, the elementary divisors are linear.
4. The number of Jordan blocks corresponding to an eigenvalue is the **geometric multiplicity** of the eigenvalue. The **algebraic multiplicity** of an eigenvalue is the sum of the orders of all the Jordan blocks associated with it. The geometric multiplicity of an eigenvalue is less than or equal to its algebraic multiplicity.

8.3 The Eigenvalue Problems Arising in Practical Applications

The problem of finding eigenvalues and eigenvectors arises in a wide variety of practical applications. It arises in almost all branches of science and engineering. As we have seen before, the mathematical models of many engineering problems are systems of differential and difference equations, and the solutions of these equations are often expressed in terms of the eigenvalues and eigenvectors of the matrices of these systems. Furthermore, many important characteristics of physical and engineering systems, such as *stability*, can often be determined only by knowing the nature and location of the eigenvalues. We give a few representative examples in this section.

8.3.1 Stability Problems for Differential and Difference Equations

A homogeneous linear system of differential equations with constant coefficients of the form

$$\frac{dx_1(t)}{dt} = a_{11}x_1(t) + a_{12}x_2(t) + \cdots + a_{1n}x_n(t)$$
$$\frac{dx_2(t)}{dt} = a_{21}x_1(t) + a_{22}x_2(t) + \cdots + a_{2n}x_n(t)$$
$$\vdots$$
$$\frac{dx_n(t)}{dt} = a_{n1}x_1(t) + a_{n2}x_2(t) + \cdots + a_{nn}x_n(t)$$

or in matrix form

$$\dot{x}(t) = Ax(t) \tag{8.3.1}$$

where

$$A = (a_{ij})_{n\times n}$$

and

$$\dot{x}(t) = \frac{d}{dt}\begin{pmatrix} x_1(t) \\ \vdots \\ x_n(t) \end{pmatrix}$$

arises in a wide variety of physical and engineering systems. The solution of this system is intimately related to the eigenvalue problem for the matrix A.

To see this, assume that (8.3.1) has a solution $x(t) = ve^{\lambda t}$, where v is not dependent on t. Then, from (8.3.1), we must have

$$\lambda v e^{\lambda t} = A v e^{\lambda t} \tag{8.3.2}$$

that is

$$Av = \lambda v \tag{8.3.3}$$

showing that λ is an eigenvalue of A and v is corresponding eigenvector. Thus the eigenpair (λ, v) of A can be used to compute a solution $x(t)$ of (8.3.1). If A has n linearly independent eigenvectors (which will happen, as we saw in the last section, when the eigenvalues of A are all distinct), then the general solution of the system can be written as

$$x(t) = c_1 v_1 e^{\lambda_1 t} + c_2 v_2 e^{\lambda_2 t} + \cdots + c_n v_n e^{\lambda_n t} \tag{8.3.4}$$

where $\lambda_1, \ldots, \lambda_n$ are the eigenvalues of A and $v_1, v_2, \ldots, v_n$ are the corresponding eigenvectors.

In the general case, the general solution of (8.3.1) with $x(0) = x_0$ is given by

$$x(t) = e^{At} x_0 \tag{8.3.5}$$

where

$$e^{At} = I + At + \frac{A^2 t^2}{2!} + \cdots$$

The eigenvalues λ_i and the corresponding eigenvectors now appear in the computation of e^{At}.

For example, if A has the Jordan canonical form

$$V^{-1}AV = \operatorname{diag}(J_1, J_2, \ldots, J_k) \qquad \text{with} \tag{8.3.6}$$

$$J_i = \begin{pmatrix} \lambda_i & 1 & & 0 \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ 0 & & & \lambda_i \end{pmatrix} \tag{8.3.7}$$

then

$$e^{At} = V \operatorname{diag}(e^{J_1 t}, e^{J_2 t}, \ldots, e^{J_k t}) V^{-1} \tag{8.3.8}$$

If $p = v_i - 1$, where v_i is the order of J_i, then

$$e^{J_i t} = e^{\lambda_i t}\begin{pmatrix} 1 & t & t^2/2 & \cdots & t^p/p! \\ 0 & 1 & t & \cdots & t^{p-1}/(p-1)! \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & t \\ 0 & \cdots & \cdots & 0 & 1 \end{pmatrix} \tag{8.3.9}$$

Thus, the system of differential equations (8.3.1) is completely solved by knowing the eigenvalues and eigenvectors of the system matrix A.

Furthermore, as we said before, many interesting and desirable properties of physical and engineering systems can be studied just by knowing the location or the nature of the eigenvalues of the system matrix A. *Stability is one such property.* The stability is defined with respect to an equilibrium solution.

DEFINITION 8.3.1 An **equilibrium solution** of the system

$$\dot{x}(t) = Ax(t), \qquad x(0) = x_0 \tag{8.3.10}$$

is the vector x_e satisfying

$$Ax_e = 0$$

Clearly, $x_e = 0$ is an equilibrium solution and it is the unique equilibrium solution if and only if A is nonsingular.

A mathematical definition of stability is now in order.

DEFINITION 8.3.2 An equilibrium solution x_e is said to be **stable** if, for every $\epsilon > 0$, there exists a real number $\delta > 0$ such that $\|x(t) - x_e\| < \epsilon$ whenever $\|x_0 - x_e\| < \delta$.

System (8.3.1) is stable if the equilibrium solution $x_e = 0$ is stable.

> In other words, the stability guarantees that the system remains close to the origin if the initial position is chosen close to it.

In many situations, stability is not enough.

DEFINITION 8.3.3 An equilibrium solution x_e is asymptotically stable if it is stable and there exists a $\delta > 0$ such that $\|x(t) - x_e\| \to 0$ as $t \to \infty$, whenever $\|x_0 - x_e\| < \delta$.

DEFINITION 8.3.4 System (8.3.1) is asymptotically stable if the equilibrium solution $x_e = 0$ is asymptotically stable.

In other words, asymptotic stability not only guarantees the stability but also ensures that every solution $x(t)$ converges to the origin whenever the initial position is chosen close to it.

Because an asymptotically stable system is necessarily stable, but the converse is not true, the following convention is normally adopted.

DEFINITION 8.3.5 A system is called **marginally stable** if it is stable but not asymptotically stable.

DEFINITION 8.3.6 A system that is not stable is called **unstable**.

Mathematical Criteria for Asymptotic Stability

THEOREM 8.3.1 Stability Theorem for a Homogeneous System of Differential Equations

1. A necessary and sufficient condition for the equilibrium solution $x_e = 0$ of the homogeneous system (8.3.1) to be asymptotically stable is that the eigenvalues of the matrix A all have negative real parts.
2. An equilibrium solution is unstable if at least one eigenvalue has a positive real part.

Proof: It is enough to prove that $x(t) \to 0$ as $t \to \infty$. Because the general solution of the system $\dot{x}(t) = Ax(t)$ is given by $x(t) = e^{At}x(0)$, the proof follows from (8.3.6)–(8.3.9).

Note that if $\lambda_j = \alpha_j + i\beta_j$, $j = 1, 2, \ldots, n$, then $e^{\lambda_j t} = e^{\alpha_j t}e^{i\beta_j t}$, and $e^{\lambda_j t} \to 0$, when $t \to \infty$, if and only if $\alpha_j < 0$. ■

Stability of a Nonhomogeneous System

Many practical situations give rise to a mathematical model of the form

$$\dot{x}(t) = Ax(t) + b \tag{8.3.11}$$

where b is a constant vector. The stability of such a system is also governed by the eigenvalues of A. In fact, the stability of any solution of (8.3.11) is equivalent to the stability of the equilibrium solution $x_e = 0$ of the homogeneous system (8.3.10). This can be seen as follows.

Let $\bar{x}(t)$ be an equilibrium solution of (8.3.11). Define

$$z(t) = x(t) - \bar{x}(t)$$

Then

$$\begin{aligned}\dot{z}(t) &= \dot{x}(t) - (\dot{\bar{x}})(t)\\ &= Ax(t) + b - A\bar{x}(t) - b\\ &= A(x(t) - \bar{x}(t)) = Az(t)\end{aligned}$$

Thus, $x(t) \to \bar{x}(t)$ if and only if $z(t) \to 0$. Theorem (8.3.2) therefore follows from Theorem (8.3.1).

THEOREM 8.3.2 Stability Theorem for a Nonhomogeneous System of Differential Equations

1. An equilibrium solution of (8.3.11) is asymptotically stable if and only if all the eigenvalues of A have negative real parts.
2. An equilibrium solution is unstable if at least one eigenvalue has positive real part.

Remarks: Because the stability of an equilibrium solution depends upon the eigenvalues of the matrix A of the system, it is usual to refer to *stability of the system* itself, or just the stability of the matrix A.

Stability of a System of Difference Equations

Like the system of differential equations (8.3.11), many practical systems are modeled by a system of difference equations: $x_{k+1} = Ax_k + b$.

A well-known mathematical criterion for asymptotic stability of such a system is given in the following theorem. We leave the proof to the reader.

THEOREM 8.3.3 Stability Theorem for a System of Difference Equations An equilibrium solution of

$$x_{k+1} = Ax_k + b \tag{8.3.12}$$

is asymptotically stable if and only if all the eigenvalues of A are inside the unit circle. The equilibrium solution is unstable if at least one eigenvalue has magnitude greater than 1.

Summarizing, to determine the stability and asymptotic stability of a system modeled by a system of first-order differential or difference equations, all we need to know is if the eigenvalues of A are in the left half plane or inside the unit circle, respectively. Explicit knowledge of the eigenvalues is *not* needed.

Example 8.3.1 A European Arms Race Consider the arms race of 1909–1914 between two European alliances.

Alliance 1: France and Russia
Alliance 2: Germany and Austria–Hungary

The two alliances went to war against each other. Let's try to explain this historical fact through the notion of stability. First, consider the following crude (but simple) mathematical model of war between two countries:

$$\frac{dx_1}{dt} = k_1x_2 - \alpha_1x_1 + g_1$$

$$\frac{dx_2}{dt} = k_2x_1 - \alpha_2x_2 + g_2$$

where

$x_i(t)$ = war potential of the country i, $i = 1, 2$

$g_i(t)$ = the grievances that country i has against the other, $i = 1, 2$

g_i, α_i, and k_i, $i = 1, 2$, are all positive constants. $\alpha_i x_i$ denotes the cost of armaments of the country i. This mathematical model is due to L. F. Richardson and is known as the *Richardson model.*

Note that this simple model is realistic in the sense that the rate of change of the war potential of one country depends upon the war potential of the other country, the grievances that one country has against its enemy country, and the cost of the armaments the country can afford. Although the first two factors cause the rate to increase, the last factor has certainly a slowing effect (which is why a minus sign is associated with that term).

In matrix form, this model can be written as:

$$\dot{x}(t) = Ax(t) + g$$

where

$$A = \begin{pmatrix} -\alpha_1 & k_1 \\ k_2 & -\alpha_2 \end{pmatrix}, \qquad x(t) = \begin{pmatrix} x_1(t) \\ x_2(t) \end{pmatrix}$$

$$g = \begin{pmatrix} g_1 \\ g_2 \end{pmatrix}$$

The eigenvalues of A are

$$\lambda = \frac{-(\alpha_1 + \alpha_2) \pm \sqrt{(\alpha_1 - \alpha_2)^2 + 4k_1k_2}}{2}$$

Thus, the equilibrium solution $x(t)$ is asymptotically stable if $\alpha_1\alpha_2 - k_1k_2 > 0$ and unstable if $\alpha_1\alpha_2 - k_1k_2 < 0$. This result occurs because when $\alpha_1\alpha_2 - k_1k_2 > 0$, both the eigenvalues will have negative real parts; if it is negative, one eigenvalue will have positive real part.

For the European arms race, the estimates of α_1, α_2 and k_1, k_2 were made under some realistic assumptions and were found to be as follows:

$$\alpha_1 = \alpha_2 = 0.2$$
$$k_1 = k_2 = 0.9$$

(For details of how these estimates were obtained, see the book by M. Braun (1978). The main assumptions are that both the alliances have roughly the same strength and α_1 and α_2 are the same as Great Britain, which is usually taken to be the reciprocal of the lifetime of the British Parliament (5 years).

With these values of α_1, α_2 and k_1, k_2, we have

$$\alpha_1\alpha_2 - k_1k_2 = \alpha_1^2 - k_1^2 = -0.7700$$

Thus, the equilibrium is unstable. In fact, the two eigenvalues are -1.1000 and 0.7000.

For a general model of Richardson's theory of arms races and the role of eigenvalues there, see Luenberger (1979, 209–14).

Convergence of Iterative Schemes for Linear Systems

In Chapter 6 (Theorem 6.10.1) we saw that the iteration

$$x_{k+1} = Bx_k + d \tag{8.3.13}$$

for solving the linear system

$$Ax = b$$

converges to the solution x for an arbitrary choice of the initial approximation x_1 if and only if the spectral radius $\rho(B) < 1$.

We thus see again that only an implicit knowledge of the eigenvalues of B is needed to determine if an iterative scheme based upon (8.3.13) is convergent.

8.3.2 Vibration Problem, Buckling Problem, and Simulating Transient Current of an Electrical Circuit

Problems such as analysis of vibration and buckling of structures, and simulation of the transient current of electrical circuits often give rise to a system of second-order differential equations of the form

$$B\ddot{y} + Ay = 0 \tag{8.3.14}$$

where

$$\ddot{y} = \begin{pmatrix} \ddot{y}_1(t) \\ \ddot{y}_2(t) \\ \vdots \\ \ddot{y}_n(t) \end{pmatrix} \tag{8.3.15}$$

The solution of such a system leads to the solution of an eigenvalue problem of the type:

$$Ax = \lambda Bx \tag{8.3.16}$$

This can be seen as follows:

Assume that solutions are of the form $y = xe^{i\omega t}$. Then from (8.3.14) we must have

$$\omega^2 Bx = Ax$$

Writing $\lambda = \omega^2$, this becomes

$$Ax = \lambda Bx \tag{8.3.17}$$

Such an eigenvalue problem is called a **generalized eigenvalue problem.**

The number λ is called an **eigenvalue**, and x is the corresponding **eigenvector** of the generalized eigenvalue problem (8.3.17).

Very often, in practice, the matrices A and B are symmetric and positive definite. The eigenvalues of the generalized eigenvalue problem in this case are all real and positive, and eigenvectors can be taken to be linearly independent. In such a case, an explicit solution of the system can be written down in terms of the eigenvalues and eigenvectors.

Eigenvalue–Eigenvector Solution of $B\ddot{y} + Ay = 0$ with A, B Positive Definite

Let $x_1, x_2, \ldots, x_n$ be the n independent eigenvectors corresponding to the eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$, of (8.3.17) with A and B positive definite. Then the general solution of (8.3.14) is given by

$$y = \sum_{k=1}^{n} \left(c_k e^{i\sqrt{\lambda_k}\, t} + d_k e^{-i\sqrt{\lambda_k}\, t} \right) x_k,$$

where c_k and d_k are arbitrary constants (Wilkinson 1965, 34–35).

In vibration problems, the matrices B and A are called the **mass** and **stiffness** matrices, respectively, and are denoted by M and K, giving rise to the *symmetric generalized eigenvalue problem*:

$$Kx = \lambda Mx$$

DEFINITION 8.3.7 The quantities $\omega_i = \sqrt{\lambda_i}$, $i = 1, \ldots, n$, are called **natural frequencies**, and $x_1, \ldots, x_n$ are called the **modes** of vibration of the system.

The frequencies can be used to determine the **periods** T_p for the vibrations. Thus

$$T_{p_i} = \frac{2\pi}{\omega_i}$$

is the period of vibration for the ith mode.

As we will see, the behavior of a vibrating system can be analyzed by knowing the natural frequencies and the modes. We will give a simple example to illustrate this. An entire chapter (Chapter 9) is devoted to the generalized eigenvalue problem.

Other structural problems such as the *buckling problem* of a beam involve *boundary value problems* for the second-order differential equations. The solutions of such problems using finite differences also lead to eigenvalue problems. We also illustrate this situation in this section.

We next describe how the frequencies and the modes can be used to predict the *phenomenon of resonance* in vibration engineering.

All machines and structures, such as bridges, buildings, and aircraft, possessing mass and stiffness experience vibration to some degree, and their design requires consideration of their oscillatory behavior. Free vibration takes place when a system oscillates due to the forces inherent in the system and without any external forces. Under free vibration such a system vibrates at one or more of its natural frequencies, which are properties of the dynamic system and depend on the associated mass and stiffness distribution. For forced vibration, systems oscillate under the excitation of external forces. When such excitation is oscillatory, the system is also forced to vibrate at the excitation frequency.

Phenomenon of Resonance

If the excitation frequency coincides or becomes close to one of the natural frequencies of the system, dangerously large oscillations may result, and a condition of resonance is encountered. This is the kind of situation an engineer would very much like to avoid. The collapse of the Tacoma Narrows (also known as Galloping Gerty) Bridge* over Puget Sound in the state of Washington in 1940 and that of the Broughton Suspension Bridge in England have been attributed to such a phenomenon.

This commonly accepted explanation for the Tacoma Bridge collapse has recently been challenged by scientists who believe there may be something more to it. See the recent interesting paper by Lazer and McKenna (1990) and the references therein.

In both of these cases, a frequency of the exciting force was equal to one of the natural frequencies of the bridge at the time of collapse. In the case of the Broughton Bridge, the exciting force was set up by soldiers marching in cadence over the bridge. In the case of the Tacoma Bridge, the force was the wind. Because of what happened in Broughton, soldiers are no longer permitted to march in cadence over a bridge. Buildings can be constructed with active devices so that the force due to wind can be controlled.

Another important property of dynamic systems is damping, which is present in all systems due to energy dissipation by friction and other resistances. However, small damping values have very little effect on the natural frequencies and are normally not included in the estimation of natural frequencies. Damping becomes important in limiting the amplitude of oscillation at resonance; the major effect of damping being reducing amplitude with time.

For a continuous elastic body the number of independent coordinates, or degrees of freedom, is infinite. However, under almost all situations, the system may be treated as dynamically equivalent to one having a finite number of degrees of freedom.

In summary, the behavior of a vibrating system can be analyzed by knowing the frequencies and the modes, and the eigenvalues and the eigenvectors of the matrix of the mathematical model of the system are related to these quantities. Specifically, *the frequencies are the square roots of the eigenvalues and the modes are the eigenvectors.*

Example 8.3.2 Vibration of a Building Consider a two-story building (Figure 8.1(a)) with a rigid floor. It is assumed that the weight distribution of the building can be represented in the form of concentrated weight at each floor level, as shown in Figure 8.1(b), and the stiffnesses of supporting columns are represented by the spring constants k_i.

The equations of motion for this system can be written as

$$m_1\ddot{y}_1 + (k_1 + k_2)y_1 - k_2 y_2 = 0$$
$$m_2\ddot{y}_2 - k_2 y_1 + k_2 y_2 = 0$$

*For a complete story of the collapse of the Tacoma bridge, see Braun (1978, 167–9).

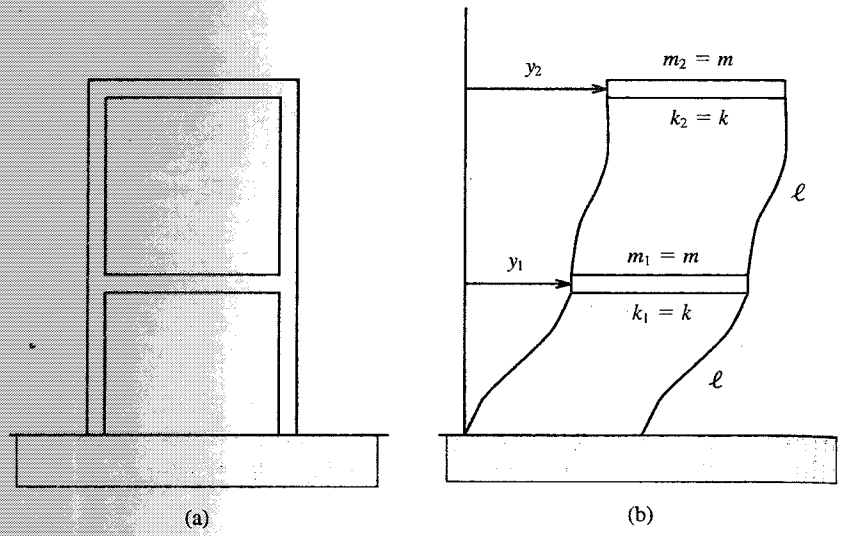


FIGURE 8.1

or

$$\begin{pmatrix} m_1 & 0 \\ 0 & m_2 \end{pmatrix}\begin{pmatrix} \ddot{y}_1 \\ \ddot{y}_2 \end{pmatrix} + \begin{pmatrix} k_1 + k_2 & -k_2 \\ -k_2 & k_2 \end{pmatrix}\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = 0 \tag{8.3.18}$$

Taking $m_1 = m_2 = m$, $k_1 = k_2 = k$, we have

$$\begin{pmatrix} m & 0 \\ 0 & m \end{pmatrix}\begin{pmatrix} \ddot{y}_1 \\ \ddot{y}_2 \end{pmatrix} + \begin{pmatrix} 2k & -k \\ -k & k \end{pmatrix}\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = 0 \tag{8.3.19}$$

Defining the mass matrix

$$M = \begin{pmatrix} m & 0 \\ 0 & m \end{pmatrix}$$

and the stiffness matrix

$$K = \begin{pmatrix} 2k & -k \\ -k & k \end{pmatrix}$$

Equation (8.3.19) becomes

$$M\ddot{y} + Ky = 0, \qquad \text{where } y = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \tag{8.3.20}$$

Assuming that a solution y is of the form

$$y = xe^{i\omega t}$$

multiplying Equation (8.3.20) by M^{-1}, and setting $A = M^{-1}K$ and $\lambda = \omega^2$, the equations of motion (8.3.20) give rise to the following eigenvalue problem.

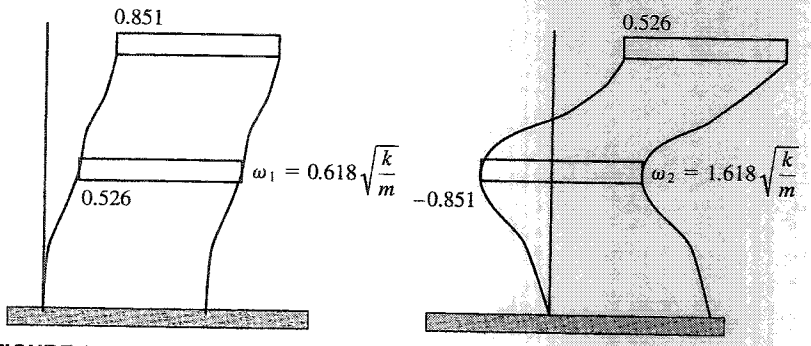


FIGURE 8.2

$$\begin{pmatrix} \frac{2k}{m} & -\frac{k}{m} \\ -\frac{k}{m} & \frac{k}{m} \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \lambda\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \tag{8.3.21}$$

The eigenvalues λ_1 and λ_2 and the corresponding eigenvectors representing two normal modes for this 2×2 problem are easily calculated. The eigenvalues are

$$\lambda_1 = \frac{k}{m}(0.3820), \qquad \lambda_2 = \frac{k}{m}(2.6180)$$

The corresponding eigenvectors are

$$\begin{pmatrix} 0.5257 \\ 0.8507 \end{pmatrix}, \qquad \begin{pmatrix} -0.8507 \\ 0.5257 \end{pmatrix}$$

The vibration of the building at these two normal modes is shown in Figure 8.2.

Buckling Problem (A Boundary Value Problem)

Consider a thin, uniform beam of length l. An axial load P is applied to the beam at one of the ends, as shown in Figure 8.3.

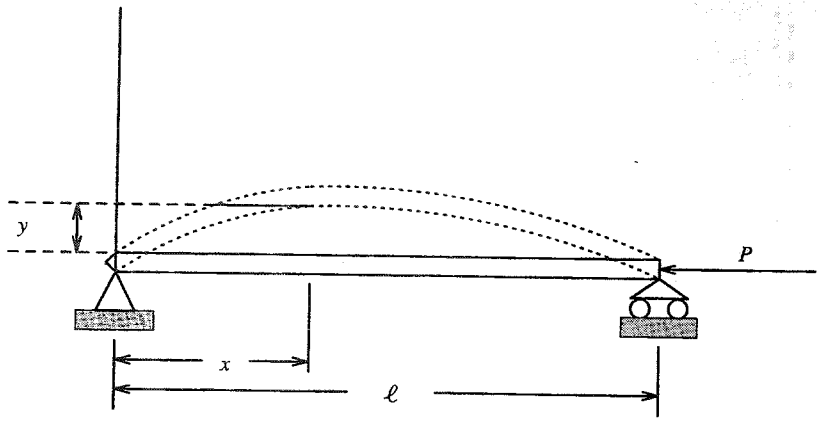


FIGURE 8.3

We are interested in the stability of the beam—that is, how and when the beam buckles. We show how this problem gives rise to an eigenvalue problem and what role the eigenvalues play.

Let y denote the vertical displacement of a point of the beam that is at a distance x from the left support. Suppose that both the ends of the beam are simply supported—that is, $y(0) = y(l) = 0$. From beam theory, the basic relationship between the curvature d^2y/dx^2 and the internal moment M for the beam is

$$\frac{d^2y}{dx^2} = \frac{M}{EI}$$

where E is the modulus of elasticity and I is the area moment of inertia of column cross section. Writing the bending moment at any section as $M = Py$, the equation gives the governing differential equation, which is the *bending moment equation*:

$$EI\frac{d^2y}{dx^2} = -Py \tag{8.3.22}$$

The constant EI is called the **flexural rigidity** of the beam.

The bending moment equation and the boundary conditions

$$y(0) = y(l) = 0 \tag{8.3.23}$$

constitute a boundary value problem.

We solve this boundary value problem by approximating d^2y/dx^2 with an appropriate finite difference scheme, as we did in Chapter 6.

Let the interval $[0, l]$ be partitioned into n subintervals of equal length h, with $x_0, x_1, \ldots, x_n$ as the points of division; that is,

$$0 = x_0 < x_1 < x_2 < \cdots < x_j < \cdots < x_{n-1} < x_n = l$$

and

$$h = \frac{l}{n}$$

Let

$$\left.\frac{d^2y}{dx^2}\right|_{x=x_i} \approx \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} \tag{8.3.24}$$

Substituting this approximation of d^2y/dx^2 into the bending moment equation (8.3.22) and taking into account the boundary conditions (8.3.23), we obtain the following **symmetric tridiagonal matrix eigenvalue problem**:

$$\begin{pmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & \ddots & \ddots & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & \ddots & -1 \\ 0 & \cdots & 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \lambda \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \tag{8.3.25}$$

where

$$\lambda = \frac{Ph^2}{EI} \tag{8.3.26}$$

Each value of λ determines a load

$$P = \frac{\lambda EI}{h^2} \tag{8.3.27}$$

called a **critical load**. These critical loads are the ones that are of practical interest, because they determine the possible onset of the buckling of the beam.

In general, the smallest value of P is of primary importance, because the bending associated with larger values of P may not be obtained without failure occurring under the action of the lowest critical value P.

Simulating Transient Current for an Electric Circuit (Chapra and Canale 1988)

Given an electric circuit consisting of several loops, suppose we are interested in the transient behavior of the electric circuit. In particular, we want to know the oscillation of each loop with respect to the other. First consider the circuit of the loop shown in Figure 8.4.

Voltage drop across a capacitor is

$$V_C = \frac{q}{C}; \qquad q = \text{charge on the capacitor}$$

$$C = \text{capacitance}$$

Voltage drop across an inductor is

$$V_L = L\frac{di}{dt}; \qquad L = \text{inductance}$$

Voltage drop across a resistor is

$$V_R = iR; \qquad R = \text{resistance}$$

Kirchhoff's voltage law states that the algebraic sum of voltage drops around a closed-loop circuit is zero. We then have, for this circuit,

$$L\frac{di}{dt} + Ri + \frac{q}{C} = 0$$

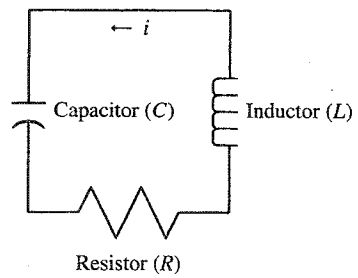


FIGURE 8.4

or

$$L\frac{di}{dt} + iR + \frac{1}{C}\int_{-\infty}^{0} i\,dt = 0$$

(because $V_C = \frac{q}{C} = \frac{1}{C}\int_{-\infty}^{t} i\,dt$).

Now consider the network with four loops shown in Figure 8.5. Kirchhoff's voltage law applied to each loop gives the following.

Loop 1:

$$-L_1\frac{di_1}{dt} - \frac{1}{C_1}\int_{-\infty}^{t}(i_1 - i_2)\,dt = 0 \tag{8.3.28}$$

Loop 2:

$$-L_2\frac{di_2}{dt} - \frac{1}{C_2}\int_{-\infty}^{t}(i_2 - i_3)\,dt + \frac{1}{C_1}\int_{-\infty}^{t}(i_1 - i_2)\,dt = 0 \tag{8.3.29}$$

Loop 3:

$$-L_3\frac{di_3}{dt} - \frac{1}{C_3}\int_{-\infty}^{t}(i_3 - i_4)\,dt + \frac{1}{C_2}\int_{-\infty}^{t}(i_2 - i_3)\,dt = 0 \tag{8.3.30}$$

Loop 4:

$$-L_4\frac{di_4}{dt} - \frac{1}{C_4}\int_{-\infty}^{t} i_4\,dt + \frac{1}{C_3}\int_{-\infty}^{t}(i_3 - i_4)\,dt = 0 \tag{8.3.31}$$

The system of ordinary differential equations given by (8.3.28)–(8.3.31) can be differentiated and rearranged to give

$$L_1\frac{d^2i_1}{dt^2} + \frac{1}{C_1}(i_1 - i_2) = 0 \tag{8.3.32}$$

$$L_2\frac{d^2i_2}{dt^2} + \frac{1}{C_2}(i_2 - i_3) - \frac{1}{C_1}(i_1 - i_2) = 0 \tag{8.3.33}$$

$$L_3\frac{d^2i_3}{dt^2} + \frac{1}{C_3}(i_3 - i_4) - \frac{1}{C_2}(i_2 - i_3) = 0 \tag{8.3.34}$$

$$L_4\frac{d^2i_4}{dt^2} + \frac{1}{C_4}i_4 - \frac{1}{C_3}(i_3 - i_4) = 0 \tag{8.3.35}$$

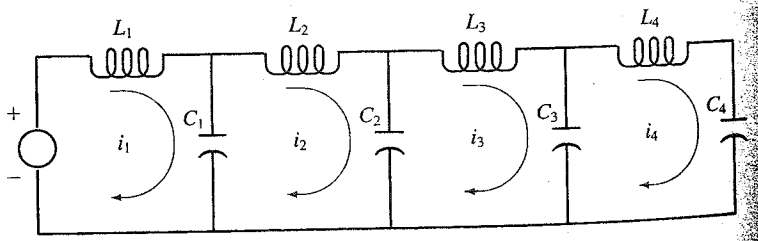


FIGURE 8.5

Assume

$$i_j = A_j \sin(\omega t), \qquad j = 1, 2, 3, 4 \tag{8.3.36}$$

(Recall that i_j is the current at the jth loop.)

From (8.3.32),

$$-L_1A_1\omega^2 \sin \omega t + \frac{1}{C_1}A_1 \sin \omega t - \frac{1}{C_1}A_2 \sin \omega t = 0$$

or

$$\boxed{\left(\frac{1}{C_1} - L_1\omega^2\right)A_1 - \frac{1}{C_1}A_2 = 0} \tag{8.3.37}$$

From (8.3.33),

$$-L_2A_2\omega^2 \sin \omega t + \frac{1}{C_2}A_2 \sin \omega t - \frac{1}{C_2}A_2 \sin \omega t - \frac{1}{C_1}A_1 \sin \omega t + \frac{1}{C_1}A_2 \sin \omega t = 0$$

or

$$\boxed{-\frac{1}{C_1}A_1 + \left(\frac{1}{C_1} + \frac{1}{C_2} - L_2\omega^2\right)A_2 - \frac{1}{C_2}A_3 = 0} \tag{8.3.38}$$

From (8.3.34),

$$-L_3A_3\omega^2 \sin \omega t + \frac{1}{C_3}A_3 \sin \omega t - \frac{1}{C_3}A_4 \sin \omega t - \frac{1}{C_2}A_2 \sin \omega t + \frac{1}{C_2}A_3 \sin \omega t = 0$$

or

$$\boxed{-\frac{1}{C_2}A_2 + \left(\frac{1}{C_2} + \frac{1}{C_3} - L_3\omega^2\right)A_3 - \frac{1}{C_3}A_4 = 0} \tag{8.3.39}$$

From (8.3.35),

$$-L_4A_4\omega^2 \sin \omega t + \frac{1}{C_4}A_4 \sin \omega t - \frac{1}{C_3}A_3 \sin \omega t - \frac{1}{C_3}A_4 \sin \omega t = 0$$

or

$$\boxed{-\frac{1}{C_3}A_3 + \left(\frac{1}{C_3} + \frac{1}{C_4} - L_4\omega^2\right)A_4 = 0 \qquad (8.3.40)}$$

Gathering the equations (8.3.37)–(8.3.40) together, we have

$$\left(\frac{1}{C_1} - L_1\omega^2\right)A_1 - \frac{1}{C_1}A_2 = 0 \qquad (8.3.41)$$

$$-\frac{1}{C_1}A_1 + \left(\frac{1}{C_1} + \frac{1}{C_2} - L_2\omega^2\right)A_2 - \frac{1}{C_2}A_3 = 0 \qquad (8.3.42)$$

$$-\frac{1}{C_2}A_2 + \left(\frac{1}{C_2} + \frac{1}{C_3} - L_3\omega^2\right)A_3 - \frac{1}{C_3}A_4 = 0 \qquad (8.3.43)$$

$$-\frac{1}{C_3}A_3 + \left(\frac{1}{C_3} + \frac{1}{C_4} - L_4\omega^2\right)A_4 = 0 \qquad (8.3.44)$$

or

$$\boxed{\begin{aligned} A_1 - A_2 &= L_1C_1\omega^2A_1 \\ -\frac{C_2}{C_1}A_1 + \frac{C_2}{C_1}A_2 + A_2 - A_3 &= L_2C_2\omega^2A_2 \\ -\frac{C_3}{C_2}A_2 + \left(\frac{C_3}{C_2} + 1\right)A_3 - A_4 &= L_3C_3\omega^2A_3 \\ -\frac{C_4}{C_3}A_3 + \left(\frac{C_4}{C_3} + 1\right)A_4 &= L_4C_4\omega^2A_4 \end{aligned} \qquad (8.3.45)}$$

In matrix form, (8.3.45) is:

$$\begin{pmatrix} 1 & -1 & 0 & 0 \\ -\frac{C_2}{C_1} & (\frac{C_2}{C_1}+1) & -1 & 0 \\ 0 & -\frac{C_3}{C_2} & (\frac{C_3}{C_2}+1) & -1 \\ 0 & 0 & -\frac{C_4}{C_3} & (\frac{C_4}{C_2}+1) \end{pmatrix}\begin{pmatrix} A_1 \\ A_2 \\ A_3 \\ A_4 \end{pmatrix} = \omega^2\begin{pmatrix} L_1C_1 & 0 & 0 & 0 \\ 0 & L_2C_2 & 0 & 0 \\ 0 & 0 & L_3C_3 & 0 \\ 0 & 0 & 0 & L_4C_4 \end{pmatrix}\begin{pmatrix} A_1 \\ A_2 \\ A_3 \\ A_4 \end{pmatrix} \qquad (8.3.46)$$

This problem is an eigenvalue problem. To see it more clearly, consider the special case

$$C_1 = C_2 = C_3 = C_4 = C$$

and

$$L_1 = L_2 = L_3 = L_4 = L$$

and assume

$$\lambda = LC\omega^2$$

Then the problem reduces to

$$\begin{aligned} (1-\lambda)A_1 - A_2 &= 0 \\ -A_1 + (2-\lambda)A_2 - A_3 &= 0 \\ -A_2 + (2-\lambda)A_3 - A_4 &= 0 \\ -A_3 + (2-\lambda)A_4 &= 0 \end{aligned}$$

or

$$\begin{pmatrix} (1-\lambda) & -1 & 0 & 0 \\ -1 & (2-\lambda) & -1 & 0 \\ 0 & -1 & +(2-\lambda) & -1 \\ 0 & 0 & -1 & +(2-\lambda) \end{pmatrix}\begin{pmatrix} A_1 \\ A_2 \\ A_3 \\ A_4 \end{pmatrix} = 0 \qquad (8.3.47)$$

Because $i_j = A_j \sin \omega t$, $j = 1, \ldots, 4$, we have

$$\begin{pmatrix} (1-\lambda) & -1 & 0 & 0 \\ -1 & (2-\lambda) & -1 & 0 \\ 0 & -1 & (2-\lambda) & -1 \\ 0 & 0 & -1 & (2-\lambda) \end{pmatrix}\begin{pmatrix} i_1 \\ i_2 \\ i_3 \\ i_4 \end{pmatrix} = 0 \qquad (8.3.48)$$

or

$$\begin{pmatrix} 1 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{pmatrix}\begin{pmatrix} i_1 \\ i_2 \\ i_2 \\ i_3 \end{pmatrix} = \lambda\begin{pmatrix} i_1 \\ i_2 \\ i_3 \\ i_4 \end{pmatrix} \qquad (8.3.49)$$

The solution of this eigenvalue problem will give us the natural frequencies ($\omega_i^2 = \lambda_i/LC$). Moreover, the knowledge of the eigenvectors can be used to study the circuit's physical behavior, such as the natural modes of oscillation.

These eigenvalues and the corresponding normalized eigenvectors (in four-digit arithmetic) are

$$\lambda_1 = 0.1206, \qquad \lambda_2 = 1, \qquad \lambda_3 = 2.3473, \qquad \lambda_4 = 3.5321$$

$$\begin{pmatrix} 0.6665 \\ 0.5774 \\ 0.4285 \\ 0.2280 \end{pmatrix}, \quad \begin{pmatrix} 0.5774 \\ -0.0000 \\ -0.5774 \\ -0.5774 \end{pmatrix}, \quad \begin{pmatrix} -0.4285 \\ 0.5774 \\ 0.2280 \\ -0.6565 \end{pmatrix}, \quad \begin{pmatrix} -0.2280 \\ 0.5774 \\ -0.6565 \\ 0.4285 \end{pmatrix}$$

From the directions of the eigenvectors, we conclude that for λ_1 all the loops oscillate in the same direction. For λ_3, the second and third loops oscillate in the opposite directions from the first and fourth, and so on. This is shown in Figure 8.6.

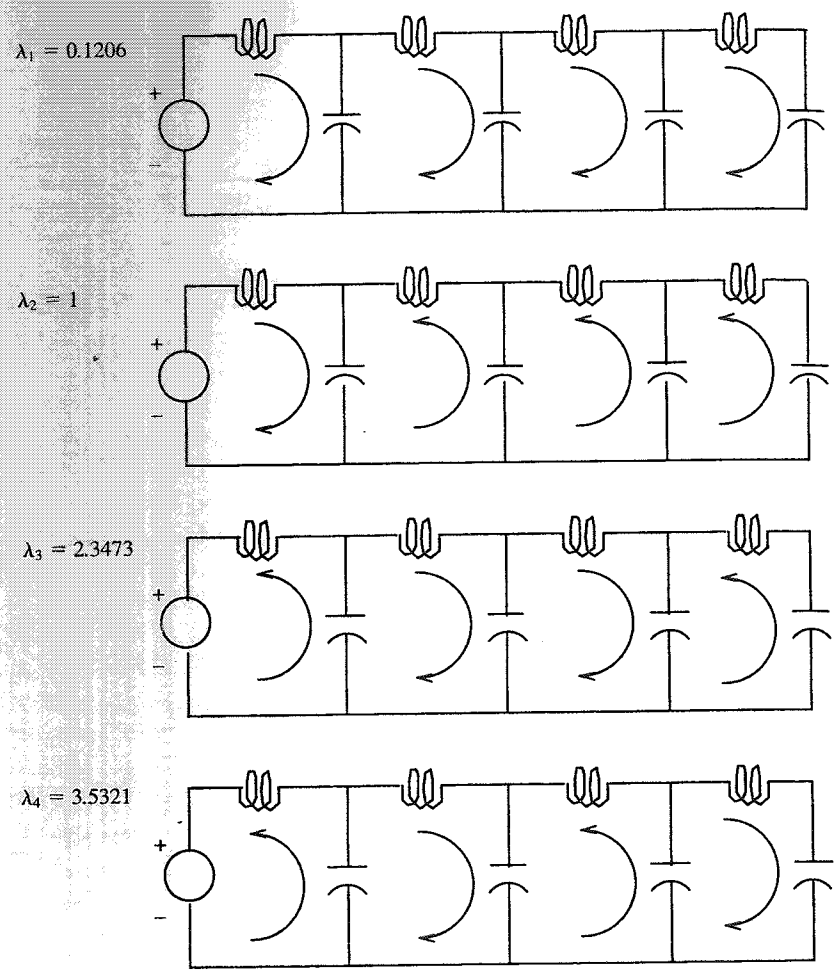


FIGURE 8.6

8.3.3 An Example of the Eigenvalue Problem Arising in Statistics: Principal Components Analysis

Many real-life applications involving statistical analysis (for example, stock market or weather prediction) involve a huge amount of data. The volume and complexities of the data in these cases can make the computations required for analysis practically infeasible. In order to handle and analyze such voluminous data in practice, it is necessary to reduce the data. The basic idea is to choose judiciously k components from a data set consisting of n measurement on p ($p > k$) original variables in such a way that much of the information (if not most) in the original p variables is contained in the k chosen components.

Such k components are called the first k **principal components** in statistics. A knowledge of eigenvalues and eigenvectors of the covariance matrix is needed to find these principal components. Specifically, if Σ is the covariance matrix corresponding to the random vector

$$X = (X_1, X_2, \ldots, X_p),$$

$\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_p \geq 0$ are the eigenvalues, and x_1 through x_p are the corresponding eigenvectors of the matrix Σ, then the ith principal component is given by

$$Y_i = x_i^T X, \qquad i = 1, 2, \ldots, p$$

Furthermore, the proportion of total population variance due to ith principal component is given by the ratio

$$\frac{\lambda_i}{\lambda_1 + \lambda_2 + \cdots + \lambda_p} = \frac{\lambda_i}{\text{trace}(\Sigma)}, \qquad i = 1, \ldots, p_n \tag{8.3.50}$$

Note: The covariance matrix is symmetric positive semidefinite; therefore, its eigenvalues are all nonnegative.

If the first k ratios constitute the most of the total population variance, then the first k principal components can be used in statistical analysis.

Note that in computing the kth ratio, we need to know only the kth eigenvalue of the covariance matrix; the entire spectrum does not need to be computed. To end this section, we remark that many real-life practices, such as computing the index of the Dow Jones Industrial Average, can be better understood and explained through the principal components analysis, as illustrated in the following example.

A Stock-Market Example (Taken from Johnson and Wichern (1992))
Suppose that the covariance matrix for the weekly rates of return for stocks of five major companies (Allied Chemical, DuPont, Union Carbide, Exxon, and Texaco) in a given period of time is given by

$$R = \begin{pmatrix} 1.000 & 0.577 & 0.509 & 0.387 & 0.462 \\ 0.577 & 1.000 & 0.599 & 0.389 & 0.322 \\ 0.509 & 0.599 & 1.000 & 0.436 & 0.426 \\ 0.387 & 0.389 & 0.436 & 1.000 & 0.523 \\ 0.462 & 0.322 & 0.426 & 0.523 & 1.000 \end{pmatrix}$$

The first two eigenvalues of R are:

$$\lambda_1 = 2.857 \tag{8.3.51}$$

$$\lambda_2 = 0.809 \tag{8.3.52}$$

The proportion of total population variance due to the first component is approximately

$$\frac{2.857}{5} = 57\% \tag{8.3.53}$$

The proportion of total population variance due to the second component is

$$\frac{0.809}{5} \cong 16\% \tag{8.3.54}$$

Thus the first two principal components account for 73% of the total population variance. The eigenvectors corresponding to these principal components are:

$$x_1^T = (0.464, 0.457, 0.470, 0.421, 0.421) \tag{8.3.55}$$

and

$$x_2^T = (0.240, 0.509, 0.260, -0.526, -0.582) \tag{8.3.56}$$

These eigenvectors have interesting interpretations. From the expression of x_1 we see that the first component is (roughly) the equally weighted sum of the five stocks. This component is generally called the **market component**. However, the expression for x_2 tells us that the second component represents a contrast between the chemical stocks and the oil-industries stocks. This component is generally called the **industry component**. Thus, we conclude that about 57% of total variations in these stock returns is due to the market activity and 16% is due to industry activity.

The eigenvalue problem also arises in many other important statistical analysis, such as in computing **canonical correlations**. Interested readers are referred to the book by Johnson and Wichern (1992) for further reading.

A Final Comment: Most eigenvalue problems arising in statistics, such as in principal components analysis and canonical correlations, are actually singular-value decomposition problems and should be handled computationally using the singular-value decomposition described in Chapter 10.

8.4 Localization of Eigenvalues

As we have seen in several practical applications, explicit knowledge of eigenvalues may not be required; all that is required is a knowledge of distribution of the eigenvalues in some given regions of the complex plane or estimates of some specific eigenvalues. There are ways such information may be acquired without actually computing the eigenvalues of the matrix. In this section, we shall discuss some of the well-known approaches. We start with a well-known result of Geršgorin, proved in 1931.

8.4.1 The Geršgorin Disk Theorems

THEOREM 8.4.1 Geršgorin's First Theorem Let $A = (a_{ij})_{n\times n}$. Define

$$r_i = \sum_{\substack{j=1\\ j\neq i}}^{n} |a_{ij}|, \qquad i = 1, \ldots, n$$

Then each eigenvalue λ of A satisfies at least one of the following inequalities:

$$|\lambda - a_{ii}| \le r_i, \qquad i = 1, 2, \ldots, n$$

each λ is contained in the disk.

In other words, all the eigenvalues of A can be found in the union of disks $\{z : |z - a_{ii}| \le r_i, i = 1, \ldots, n\}$.

Proof: Let λ be an eigenvalue of A and x be an eigenvector associated with λ. Then from

$$Ax = \lambda x$$

or from

$$(\lambda I - A)x = 0$$

we have

$$(\lambda - a_{ii})x_i = \sum_{\substack{j=1\\ j\neq i}}^{n} a_{ij}x_j, \qquad i = 1, \ldots, n$$

where x_i is the ith component of the vector x. Let x_k be the largest component of x (in absolute value). Then, because $|x_j|/|x_k| \le 1$ for $j \neq k$, we have

$$\begin{aligned} |\lambda - a_{kk}| &\le \sum_{\substack{j=1\\ j\neq k}}^{n} |a_{kj}| \frac{|x_j|}{|x_k|} \\ &\le \sum_{\substack{j=1\\ j\neq k}}^{n} |a_{kj}| \end{aligned}$$

Thus, λ is contained in the disk $\{\lambda : |\lambda - a_{kk}| \le r_k\}$. ■

DEFINITION 8.4.1 The disks $R_i = \{z : |z - a_{ii}| \le r_i, i = 1, \ldots, n\}$ are called **Geršgorin disks** in the complex plane.

Example 8.4.1

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 4 & 9 \\ 1 & 1 & 1 \end{pmatrix}$$

$$r_1 = 5$$
$$r_2 = 12$$
$$r_3 = 2$$

The Geršgorin disks are:

$$R_1: \quad \{z : |z - 1| \le 5\}$$
$$R_2: \quad \{z : |z - 4| \le 12\}$$
$$R_3: \quad \{z : |z - 1| \le 2\}$$

(The eigenvalues of A are 7.3067, $-0.6533 \pm 0.3473i$.) See Figure 8.7.

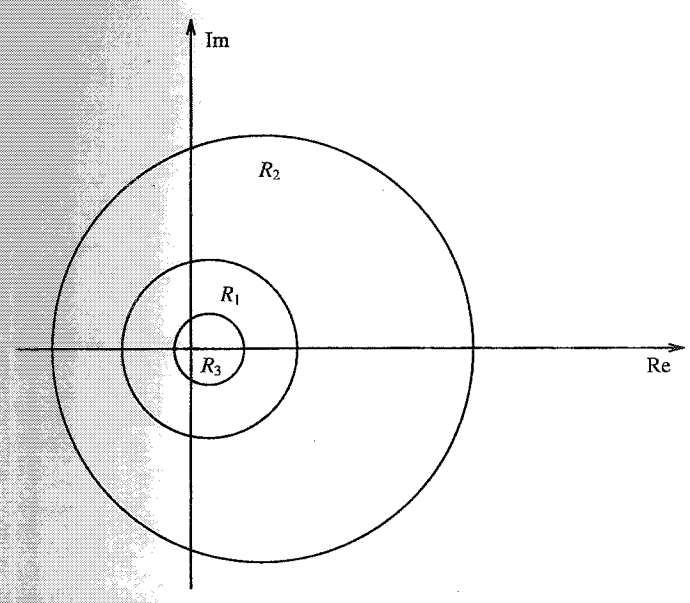


FIGURE 8.7 Geršgorin's Disks: Example 8.4.1.

Although Theorem 8.4.1 tells us only that the eigenvalues of A lie in the union of n Geršgorin disks, the following theorem gives some more specific information. We state the theorem without proof.

THEOREM 8.4.2 Geršgorin's Second Theorem Suppose that r Geršgorin disks are disjoint from the rest, then exactly r eigenvalues of A lie in the union of the r disks.

Proof: Horn and Johnson (1985, 344–345). ■

Note: For generalizations of Geršgorin's theorems, see Horn and Johnson (1985) and Brualdi and Mallendorf (1994).

Example 8.4.2

$$A = \begin{pmatrix} 1 & 0.1 & 0.2 \\ 0.2 & 4 & 0.3 \\ 0.4 & 0.5 & 8 \end{pmatrix}$$

The Geršgorin disks are:

R_1: $\{z : |z - 1| \le 0.3\}$
R_2: $\{z : |z - 4| \le 0.5\}$
R_3: $\{z : |z - 8| \le 0.9\}$

All the three disks are disjoint from each other. Therefore, by Theorem 8.4.2, each disk must contain exactly one eigenvalue of A. This is indeed true. Note that the eigenvalues A are 0.9834, 3.9671, and 8.0495. See Figure 8.8.

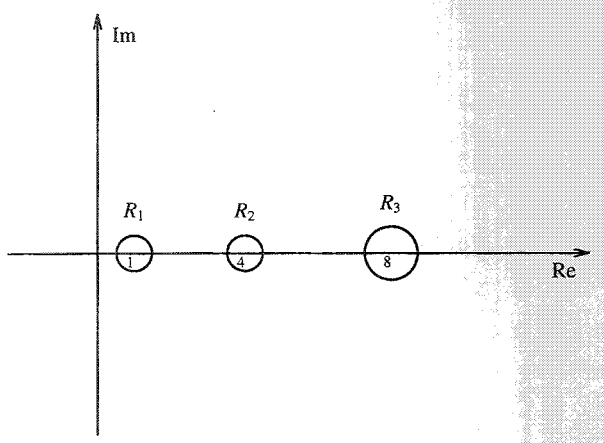


FIGURE 8.8 Geršgorin's Disks: Example 8.4.2.

8.4.2 Eigenvalue Bounds and Matrix Norms

Simple matrix norms can sometimes be used to obtain useful bounds for the eigenvalues, as the following theorems illustrate. Theorem 8.4.3 is a restatement of Theorem 1.7.5.

THEOREM 8.4.3 For any consistent pair of matrix–vector norms, we have

$$|\lambda| \le \|A\|$$

where λ is an eigenvalue of A. In particular, $\rho(A)$, the spectral radius of A, is bounded by $\|A\|$: $\rho(A) \le \|A\|$.

Note: In Example 8.4.1, $\|A\|_\infty = 16$, and $\rho(A) = 7.3067$.

Since the eigenvalues of A^T are the same as those of A, we have

COROLLARY 8.4.1

$$\rho(A) \le \|A^T\|$$

Combining these two results and taking the infinity norm in particular, we obtain Theorem 8.4.4.

THEOREM 8.4.4

$$\rho(A) \le \min\{\max_i \sum_{j=1}^{n} |a_{ij}|, \max_j \sum_{i=1}^{n} |a_{ij}|\}$$

THEOREM 8.4.5 Let $\lambda_1, \lambda_2, \ldots, \lambda_n$ be the eigenvalues of A; then

$$\sum_{i=1}^{n} |\lambda_i|^2 \le \|A\|_F^2$$

Proof: The Schur triangularization theorem (Theorem 8.2.6) tells us that there exists a unitary matrix U such that

$$U^*AU = T, \qquad \text{an upper triangular matrix}$$

Thus,

$$TT^* = U^*AA^*U$$

So, AA^* is unitarily similar to TT^*. Because similar matrices have the same traces,

$$Tr(TT^*) = Tr(AA^*) = \|A\|_F^2$$

Again, $Tr(TT^*) = \sum_{i=1}^{n}\sum_{j=1}^{n} |t_{ij}|^2$. So,

$$\sum_{i=1}^{n}\sum_{j=1}^{n} |t_{ij}|^2 = \|A\|_F^2$$

Also,

$$\sum_{i=1}^{n} |t_{ii}|^2 = \sum_{i=1}^{n} |\lambda_i|^2$$

Thus, $\sum_{i=1}^{n} |\lambda|_i^2 = \sum_{i=1}^{n} |t_{ii}|^2 \le \sum_{i=1}^{n}\sum_{j=1}^{n} |t_{ij}|^2 = \|A\|_F^2$. ■

8.4.3 Inertia of a Matrix

We have seen in Section 8.3.1 that the stability of a physical system modeled by a system of differential equations is determined just by knowing if the eigenvalues of the system matrix A have all negative real parts. A problem more general than the stability problem is the *inertia problem*.

The **inertia** of an $n \times n$ complex matrix A, with respect to the half planes, is defined to be an integer triple, $In(A) = (\pi(A), \nu(A), \delta(A))$, where $\pi(A)$ is the number of eigenvalues of A with positive real parts, $\nu(A)$ is the number of eigenvalues with negative real parts, and $\delta(A)$ is the number of eigenvalues with zero real parts, counting multiplicities.

Thus, A is a **stability matrix** if $In(A) = (0, n, 0)$. A classical result on the inertia of a Hermitian matrix A is Sylvester's law of inertia.

THEOREM 8.4.6 Sylvester's Law of Inertia Theorem Let A be Hermitian and P be a nonsingular matrix. Then

$$In(A) = In(PAP^*)$$

For a proof, see Horn and Johnson (1985, 223–224).

There is extensive literature on the inertia of a non-Hermitian matrix.

The non-Hermitian inertia theorems attempt to determine conditions under which the inertia of a non-Hermitian matrix can be made equal to the inertia of a Hermitian matrix. Discussing the inertia theory of a non-Hermitian matrix is beyond the scope of this book. For an account of inertia theory see Datta (1986) and the forthcoming inertia paper by Datta.

The importance of Sylvester's law of inertia lies in the fact that it tells us that the inertia of a Hermitian matrix A is equal to the inertia of a diagonal matrix D which can be obtained by triangular factorization of A (see the following discussion).

Computing the Inertia of a Symmetric (or Complex Hermitian Matrix)

If A is symmetric (or complex Hermitian), then Sylvester's law of inertia provides an inexpensive and numerically effective method for computing the inertia.

Recall from Section 6.4.7 that a symmetric matrix A admits a triangular factorization:

$$A = UDU^T$$

where U is a product of elementary unit upper triangular and permutation matrices, and D is a symmetric block diagonal with blocks of order 1 or 2. This is known as **diagonal pivoting factorization** (see Chapter 6 and the relevant papers of Bunch (1971), Bunch and Parlett (1971), and Bunch and Kaufman (1977). Thus, by Sylvester's law of inertia $In(A) = In(D)$.

Let D have p blocks of order 1 and q blocks of order 2, with $p + 2q = n$. Assume that none of the 2×2 blocks of D is singular. Suppose that out of p blocks of order 1, p' of them are positive, p'' of them are negative, and p''' of them are zero (that is, $p' + p'' + p''' = p$). Then

$$\pi(A) = p' + q$$
$$\nu(A) = p'' + q$$
$$\delta(A) = p'''$$

The diagonal pivoting factorization can be achieved in a numerically stable way. It requires only $n^3/6$ flops.

Computing the Inertia of a Non-Hermitian Matrix

It is generally believed by numerical analysts that the inertia of a non-Hermitian matrix A be best determined by computing explicitly the eigenvalues of A.

Having said this, let us point out that there exists a computational method due to Carlson and Datta (1979) for determining the inertia of a non-Hermitian matrix. This method does not require eigenvalue computations or solution of a matrix equation. The method requires only $n^3/3$ flops after A has been transformed to a Hessenberg matrix.

The Carlson–Datta inertia method is based on the *implicit* solution of a special matrix equation: Starting from a non-Hermitian matrix A, the method constructs a Hermitian matrix X such that whenever X is nonsingular, $In(A) = In(X)$. The computed matrix X turns out to be a solution of a matrix equation, but a matrix equation is not explicitly solved. Of course, once the Hermitian matrix X is constructed, its inertia can be computed using Sylvester's law of inertia as described in the last section.

The numerical round-off properties of the method have not been fully investigated. As an initial step, the method requires transformation of A to a Hessenberg matrix H, and it seems that if one or more codiagonal entries of H are small or near zero, a large round-off error may be expected in the construction of X. The method has been reformulated by Datta and Datta (1987) so that no division by diagonal

entries is needed. The method works very well (so does the Carlson–Datta method) in practice, even with small codiagonal entries, but this has not been established by rigorous round-off error analysis of either of these methods.

8.5 Computing Selected Eigenvalues and Eigenvectors

8.5.1 Discussions on the Importance of the Largest and Smallest Eigenvalues

We have seen before that in several applications, all we needed to compute were a few largest or smallest eigenvalues and the corresponding eigenvectors. For example, recall that in the buckling problem, it is the smallest eigenvalue that is the most important one.

In *vibration analysis of structures*, a common engineering practice is just to compute the first few smallest eigenvalues (frequencies) and the corresponding eigenvectors (modes), because it has been seen in practice that the larger eigenvalues and eigenvectors contribute very little to the total response of the system. The same remarks hold in the case of control problems modeled by a system of second-order differential equations arising in the finite element–generated reduced-order model of large flexible space structures (Inman 1989).

In statistical applications, such as in *principal component analysis*, only the first few largest eigenvalues are computed. There are other applications where only the dominant and the subdominant eigenvalues and the corresponding eigenvectors play an important role.

8.5.2 The Role of Dominant Eigenvalues and Eigenvectors in Dynamic Systems

Let's discuss briefly the role of dominant eigenvalues and the eigenvectors in the context of dynamic systems.

Consider the homogeneous discrete-time system

$$x_{k+1} = Ax_k, \qquad k = 0, 1, 2, \ldots$$

Let λ_1 be the dominant eigenvalue of A; that is, $|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \cdots \geq |\lambda_n|$, where $\lambda_1, \lambda_2, \ldots, \lambda_n$ are the eigenvalues of A. Suppose that A has a set of independent eigenvectors: $v_1, v_2, \ldots, v_n$. Then the state x_k at any time $k > 0$ is given by

$$x_k = \alpha_1\lambda_1^k v_1 + \alpha_2\lambda_2^k v_2 + \cdots + \alpha_n\lambda_n^k v_n$$

where $x_0 = \alpha_1 v_1 + \cdots + \alpha_n v_n$. Because $|\lambda_1|^k > |\lambda_i|^k$, $i = 2, 3, \ldots, n$, it follows that for large values of k,

$$|\alpha_1\lambda_1^k| \gg |\alpha_i\lambda_i^k|, \qquad i = 2, 3, \ldots, n$$

provided that $\alpha_1 \neq 0$. This means that for large values of k, the state vector x_k will approach the direction of the vector v_1 corresponding to the dominant eigenvalue λ_1. Furthermore, the rate at which the state vector approaches v_1 is determined by the ratio of the second to the first dominant eigenvalue: $|\lambda_2/\lambda_1|$. (A proof is presented later.)

In the case $\alpha_1 = 0$, the second dominant eigenvalue λ_2 and the corresponding eigenvector assume the role of the first dominant eigenpair. Similar conclusions hold for the continuous-time system

$$\dot{x}(t) = Ax(t)$$

For details, see Luenberger (1979).

> In summary, the long term behavior of a homogeneous dynamical system can essentially be predicted from just the first and second dominant eigenvalues of the system matrix and the corresponding eigenvectors.

The second dominant eigenpair is particularly important in the case $\alpha_1 = 0$. In this case, the long-term behavior of the system is determined by this pair.

An Example: The Population Study

Let's take the case of a population system to illustrate this. It is well known (Luenberger 1979, 170) that such a system can be modeled by

$$p_{k+1} = Ap_k, \qquad k = 0, 1, 2, \ldots$$

where p_k is the population vector. If the dominant eigenvalue λ_1 of the matrix A is less than 1 (that is, if $|\lambda_1| < 1$), then it follows from

$$p_k = \alpha_1\lambda_1^k v_1 + \cdots + \alpha_n\lambda_n^k v_n$$

that the population decreases to zero as k becomes large. Similarly, if $|\lambda_1| > 1$, then there is long-term growth in the population. In the latter case, the original population approaches a final distribution that is defined by the eigenvector of the dominant eigenvalue. Moreover, it is the second dominant eigenvalue of A that determines how fast the original population distribution is approaching the final distribution. Finally, if the dominant eigenvalue is 1, then over the long term there is neither growth nor decay in the population.

8.5.3 The Power Method, the Inverse Iteration, and the Rayleigh Quotient Iteration

In this section we briefly describe two well-known classical methods for finding the dominant eigenvalues and the corresponding eigenvectors of a matrix. The methods are particularly suitable for *sparse matrices*, because they rely on matrix vector multiplications only.

The Power Method

The **power method** is frequently used to find the **dominant eigenvalue** and the corresponding eigenvector of a matrix. It is so named because it is based on *implicit* construction of the powers of A.

Let the eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$ of A be such that

$$|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \cdots \geq |\lambda_n|$$

that is, λ_1 is the dominant eigenvalue of A. Let v_1 be the corresponding eigenvector. Let $\max(g)$ denote the element of maximum modulus of the vector g. Assume that A is diagonalizable.

ALGORITHM 8.5.1 Power Method

Step 1: Choose x_0

Step 2: For $k = 1, 2, 3, \ldots$ do until convergence

$$\hat{x}_k = Ax_{k-1}$$
$$x_k = \hat{x}_k / \max(\hat{x}_k)$$

End

THEOREM 8.5.1 $\{\max(\hat{x}_k)\} \to \lambda_1$, and $\{x_k\} \to w_1$, a multiple of v_1, as $k \to \infty$.

Proof: From before, we have

$$x_k = \frac{A^k x_0}{\max(A^k x_0)}$$

Since A is diagonalizable, the eigenvectors v_1 through v_n associated with $\lambda_1, \ldots, \lambda_n$ can be chosen to be linearly independent. We can then write $x_0 = \alpha_1 v_1 + \alpha_2 v_2 + \cdots + \alpha_n v_n$, $\alpha_1 \neq 0$. So,

$$\begin{aligned} A^k x_0 &= A^k(\alpha_1 v_1 + \alpha_2 v_2 + \cdots + \alpha_n v_n) \\ &= \alpha_1 \lambda_1^k v_1 + \alpha_2 \lambda_2^k v_2 + \cdots + \alpha_n \lambda_n^k v_n \\ &= \lambda_1^k \left[\alpha_1 v_1 + \alpha_2 \left(\frac{\lambda_2}{\lambda_1}\right)^k v_2 + \cdots + \alpha_n \left(\frac{\lambda_n}{\lambda_1}\right)^k v_n \right] \end{aligned}$$

Because λ_1 is the dominant eigenvalue,

$$\left(\frac{\lambda_i}{\lambda_1}\right)^k \to 0 \quad \text{as} \quad k \to \infty, \qquad i = 2, 3, \ldots, n$$

Thus,

$$x_k = \frac{A^k x_0}{\max(A^k x_0)} \to c v_1, \qquad \text{where } c \text{ is a constant}$$

and

$$\{\max(\hat{x}_k)\} \to \lambda_1 \qquad \blacksquare$$

Remarks: We have derived the power method under two constraints:

1. $\alpha_1 \neq 0$.
2. λ_1 is the *only* dominant eigenvalue.

The first constraint ($\alpha_1 \neq 0$) is not really a serious practical constraint, because after a few iterations, roundoff errors will almost always make it happen.

As far as the second constraint is concerned, we note that if there are a number of independent eigenvectors corresponding to the dominant eigenvalue, the convergence is not affected. Thus if $\lambda_1 = \lambda_2 = \cdots = \lambda_r$ and $|\lambda_1| > |\lambda_{r+1}| \geq \cdots \geq |\lambda_n|$, and if

there are r independent eigenvectors associated with λ_1, then we have

$$\begin{aligned} A^k x_0 &= \lambda_1^k \left(\sum_{i=1}^{r} \alpha_i v_i + \sum_{i=r+1}^{n} \alpha_i \left(\frac{\lambda_i}{\lambda_1}\right)^k v_i \right) \\ &\cong \lambda_1^k \sum_{1}^{r} \alpha_i v_i \end{aligned}$$

(because $(\lambda_i/\lambda_1)^k$ is small for large values of k). This shows that in this case the iterates converge to some vector in the subspace spanned by $v_1, \ldots, v_r$.

Example 8.5.1

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \end{pmatrix}$$
$$x_0 = (1, 1, 1)^T$$

The eigenvalues of A are 0, -0.6235, and 9.6235. The normalized eigenvector corresponding to the largest eigenvalue 9.6235 is

$$(0.3851, 0.5595, 0.7339)^T$$

$k = 1$:

$$\hat{x}_1 = Ax_0 = \begin{pmatrix} 6 \\ 9 \\ 12 \end{pmatrix}$$
$$\max(\hat{x}_1) = 12$$
$$x_1 = \frac{\hat{x}_1}{\max(\hat{x}_1)} = \begin{pmatrix} \frac{1}{2} \\ \frac{3}{4} \\ 1 \end{pmatrix} = \begin{pmatrix} 0.50 \\ 0.75 \\ 1 \end{pmatrix}$$

$k = 2$:

$$\hat{x}_2 = Ax_1 = \begin{pmatrix} 5.00 \\ 7.25 \\ 9.50 \end{pmatrix}, \qquad x_2 = \begin{pmatrix} 0.5263 \\ 0.7632 \\ 1.0000 \end{pmatrix}$$
$$\max(\hat{x}_2) = 9.50$$

$k = 3$:

$$\hat{x}_3 = Ax_2 = \begin{pmatrix} 5.0526 \\ 7.3421 \\ 9.6316 \end{pmatrix}$$
$$x_3 = \frac{\hat{x}_3}{\max(\hat{x}_3)} = \begin{pmatrix} 0.5246 \\ 0.7623 \\ 1.000 \end{pmatrix}$$
$$\max(\hat{x}_3) = 9.6316$$

Thus $\{\max(\hat{x}_3)\}$ is converging toward the largest eigenvalue, 9.6235, and $\{x_k\}$ is converging toward the direction of the eigenvector associated with this eigenvalue. (Note that the normalized dominant eigenvector

$$\begin{pmatrix} 0.3851 \\ 0.5595 \\ 0.7339 \end{pmatrix}$$

is a scalar multiple of x_3.)

MATCOM Note: Algorithm 8.5.1 has been implemented in MATCOM program POWER.

Convergence of the Power Method

The rate of convergence of the power method is determined by the ratio $|\lambda_2/\lambda_1|$, as is easily seen from the following. From the proof of the power method (Algorithm 8.5.1), we have

$$\begin{aligned} \|x_k - \alpha_1 v_1\| &= \|\alpha_2 \left(\frac{\lambda_2}{\lambda_1}\right)^k v_2 + \cdots + \alpha_n \left(\frac{\lambda_n}{\lambda_1}\right)^k v_n\| \\ &\le |\alpha_2| \left|\frac{\lambda_2}{\lambda_1}\right|^k \|v_2\| + \cdots + |\alpha_n| \left|\frac{\lambda_n}{\lambda_1}\right|^k \|v_n\| \\ &\le \left|\frac{\lambda_2}{\lambda_1}\right|^k (|\alpha_2|\|v_2\| + \cdots + |\alpha_n|\|v_n\|) \end{aligned}$$

(because $|\lambda_i/\lambda_1| \le |\lambda_2/\lambda_1|$, $i = 3, 4, \ldots, n$).

Thus we have

$$\|x_k - \alpha_1 v_1\| \le \alpha \left|\frac{\lambda_2}{\lambda_1}\right|^k, \qquad k = 1, 2, 3, \ldots$$

where

$$\alpha = (|\alpha_2|\|v_2\| + \cdots + |\alpha_n|\|v_n\|)$$

This shows that the rate at which x_k approaches $\alpha_1 v_1$ depends upon how fast $|\lambda_2/\lambda_1|^k$ goes to zero. The absolute value of the error at each step decreases by the ratio $|\lambda_2/\lambda_1|$, that is, *if λ_2 is close to λ_1, then the convergence will be very slow. If this ratio is small, the convergence will be fast.*

The Power Method with Shift

In some cases, convergence can be significantly improved by using a suitable shift. Thus, if σ is a suitable shift so that $\lambda_1 - \sigma$ is the dominant eigenvalue of $A - \sigma I$ and if the power method is applied to the shifted matrix $A - \sigma I$, then the rate of convergence will be determined by the ratio

$$\left|\frac{\lambda_2 - \sigma}{\lambda_1 - \sigma}\right|, \quad \text{rather than} \quad \left|\frac{\lambda_2}{\lambda_1}\right|$$

(Note that by shifting the matrix A by σ, the eigenvalues are shifted by σ, but the eigenvectors remain unaltered.)

By choosing σ appropriately, in some cases, the ratio

$$\left|\frac{\lambda_2 - \sigma}{\lambda_1 - \sigma}\right|$$

can be made significantly smaller than $|\lambda_2/\lambda_1|$, thus yielding the faster convergence. An optimal choice (Wilkinson 1965, 572) of σ, assuming that λ_i are all real, is $\frac{1}{2}(\lambda_2 + \lambda_n)$. This simple choice of σ sometimes indeed yields very fast convergence, but there are many common examples where the convergence can still be slow with this choice of σ.

Consider a 20×20 matrix A with the eigenvalues $20, 19, \ldots, 2, 1$. The choice of $\sigma = (19 + 1)/2 = 10$ yields the ratio

$$\left|\frac{\lambda_2 - \sigma}{\lambda_1 - \sigma}\right| = \frac{9}{10}$$

still close to 1. Therefore, the rate of convergence will still be slow. Furthermore, the above choice of σ is not useful in practice, because the eigenvalues are not known a priori.

The Inverse Power Method/Inverse Iteration

The following iterative method, known as the **inverse iteration**, is an effective method for computing an eigenvector when a reasonably good approximation to an eigenvalue is known.

ALGORITHM 8.5.2 Inverse Iteration Let σ be an approximation to a real eigenvalue λ_1 such that $|\lambda_1 - \sigma| \ll |\lambda_i - \sigma|$ $(i \ne 1)$; that is, σ is much closer to λ_1 than to the other eigenvalues.

Step 1: Choose x_0

Step 2: For $k = 1, 2, 3, \ldots,$ do

Solve $(A - \sigma I)\hat{x}_k = x_{k-1}$ for $\hat{x}_k$, using Gaussian elimination with partial pivoting.

$x_k = \hat{x}_k / \max(\hat{x}_k)$

Stop if $\|(A - \sigma I)x_k\|_\infty < c\mu\|A\|_\infty$ where c is a constant of order unity.

Remark: Note that the use of the above stopping criterion will make the pair (σ, x_k) an exact eigenpair of a nearby matrix.

THEOREM 8.5.2 The sequence $\{x_k\}$ converges to the direction of the eigenvector corresponding to λ_1.

Remark: Note that inverse iteration is simply the power method applied to $(A - \sigma I)^{-1}$, which is why it is also known as the **inverse power method**.

Proof: The eigenvalues of $(A - \sigma I)^{-1}$ are $(\lambda_1 - \sigma)^{-1}, (\lambda_2 - \sigma)^{-1}, \ldots, (\lambda_n - \sigma)^{-1}$ and the eigenvectors $v_1, v_2, \ldots, v_n$ are the same as those of A. Thus, as in the case of

the power method, we can write

$$\hat{x}_k = \frac{c_1}{(\lambda_1 - \sigma)^k} v_1 + \frac{c_2 v_2}{(\lambda_2 - \sigma)^k} + \cdots + \frac{c_n}{(\lambda_n - \sigma)^k} v_n$$
$$= \frac{1}{(\lambda_1 - \sigma)^k}\left[c_1 v_1 + c_2\left(\frac{\lambda_1 - \sigma}{\lambda_2 - \sigma}\right)^k v_2 + \cdots + c_n\left(\frac{\lambda_1 - \sigma}{\lambda_n - \sigma}\right)^k v_n\right]$$

Because λ_1 is closer to σ than any other eigenvalue, the first term on the right-hand side is the dominating one and, therefore, $\{x_k\}$ converges to the direction of v_1. It is the direction of v_1 that we are trying to compute. ■

An Illustration

Let us illustrate these ideas with $k = 1$. Suppose that $x_0 = c_1 v_1 + c_2 v_2 + \cdots + c_n v_n$. Then

$$\hat{x}_1 = (A - \sigma I)^{-1} x_0 = (\lambda_1 - \sigma)^{-1} c_1 v_1 + (\lambda_2 - \sigma)^{-1} c_2 v_2 + \cdots + (\lambda_n - \sigma)^{-1} c_n v_n$$

Because λ_1 is closer to σ than any other eigenvalue, the coefficient of the first term in the expansion, $1/(\lambda_1 - \sigma)$, is the dominant one (it is the largest). Thus, $\hat{x}_1$ is roughly a multiple of v_1, which is what we desire.

Numerical Stability of the Inverse Iteration

At first sight the inverse iteration seems to be a dangerous procedure, because if σ is near λ_1, the matrix $(A - \sigma I)$ is obviously ill conditioned. Consequently, this ill-conditioning might affect the computed approximations of the eigenvector. Fortunately, in practice the ill-conditioning of the matrix $(A - \sigma I)$ is exactly what we want. The error at each iteration grows toward the direction of the eigenvector, and it is the direction of the eigenvector in which we are interested.

Wilkinson has remarked that in practice $\hat{x}_k$ is remarkably close to the solution of

$$(A - \sigma I + F)x_k = x_{k-1}$$

where F is small. For details see Wilkinson (1965, 620–621).*

Example 8.5.2

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \end{pmatrix}$$

$$x_0 = (1, 1, 1)^T, \qquad \sigma = 9$$

*The iterated vectors do indeed converge eventually to the eigenvectors of $A + F$.

$k = 1$:

$$\hat{x}_1 = (1, 1.5, 2)^T$$
$$x_1 = \frac{\hat{x}_1}{\|\hat{x}_1\|_2} = (0.3714, 0.5571, 0.7428)^T$$

$k = 2$:

$$\hat{x}_2 = (0.6190, 0.8975, 1.1761)^T$$
$$x_2 = \frac{\hat{x}_2}{\|\hat{x}_2\|_2} = (0.3860, 0.5597, 0.7334)^T$$

$k = 3$:

$$\hat{x}_3 = (0.6176, 0.8974, 1.1772)^T$$
$$x_3 = \frac{\hat{x}_3}{\|\hat{x}_3\|_2} = (0.3851, 0.5595, 0.7340)$$

$k = 4$:

$$\hat{x}_4 = (0.6176, 0.8974, 1.1772)^T$$
$$x_4 = \frac{\hat{x}_4}{\|\hat{x}_4\|_2} = (0.3851, 0.5595, 0.7340)^T$$

$k = 5$:

$$\hat{x}_5 = (0.6177, 0.8974, 1.1772)^T$$
$$x_5 = \frac{\hat{x}_5}{\|(\hat{x}_5)\|_2} = (0.3851, 0.5595, 0.7339)$$

Remark: In the preceding example we have used norm $\| \quad \|_2$ as a scaling for the vector x_i to emphasize that the scaling is immaterial, because we are working toward the direction of the eigenvector.

MATCOM Note: Algorithm 8.5.2 has been implemented in MATCOM program INVITR.

Choosing the Initial Vector x_0

To choose the initial vector x_0 we can run a few iterations of the power method and then switch to the inverse iteration, with the last vector generated by the power method as the initial vector x_0 in the inverse iteration. Wilkinson (1965, 627) has stated that if x_0 is assumed to be such that

$$Le = Px_0$$

where P is the permutation matrix satisfying

$$P(A - \sigma I) = LU$$

and e is the vector of unit elements, then two iterations are usually adequate, provided that σ is a good approximation to the eigenvalue.

Note: If x_0 is chosen as described, then the computation of $\hat{x}_1$ involves only solution of the triangular system:

$$U\hat{x}_1 = e$$

Example 8.5.3

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \end{pmatrix}$$

The eigenvalues of A are

$$0, \quad -0.6235, \quad \text{and} \quad 9.6235$$

Choose $\sigma = 9.1$:

$$A - \sigma I = \begin{pmatrix} -8.1 & 2.0 & 3.0 \\ 2.0 & -6.1 & 4.0 \\ 3.0 & 4.0 & -4.1 \end{pmatrix}$$

$$L = \begin{pmatrix} 1 & 0 & 0 \\ -0.2469 & 1 & 0 \\ -0.3704 & -0.8456 & 1 \end{pmatrix}, \quad U = \begin{pmatrix} -8.1 & 2 & 3 \\ 0 & -5.6062 & 4.7407 \\ 0 & 0 & 1.0200 \end{pmatrix}$$

$$P = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$x_0 = P^{-1}Le = \begin{pmatrix} 1.000 \\ 0.7531 \\ -0.2160 \end{pmatrix}$$

$k = 1$:

$$\hat{x}_1 = (A - \sigma I)^{-1}x_0 = \begin{pmatrix} 0.4003 \\ 0.6507 \\ 0.9804 \end{pmatrix}$$

$$x_1 = \frac{\hat{x}_1}{\max(\hat{x}_1)} = \begin{pmatrix} 0.4083 \\ 0.6637 \\ 1.000 \end{pmatrix}$$

$k = 2$:

$$\hat{x}_2 = (A - \sigma I)^{-1}x_1 = \begin{pmatrix} 0.9367 \\ 1.3540 \\ 1.7624 \end{pmatrix}$$

$$x_2 = \begin{pmatrix} 0.5315 \\ 0.7683 \\ 1.000 \end{pmatrix}$$

which is about 1.3 times the normalized eigenvector.

The normalized eigenvector correct to *four* digits is

$$\begin{pmatrix} 0.3851 \\ 0.5595 \\ 0.7339 \end{pmatrix}$$

The Rayleigh Quotient

THEOREM 8.5.3 Let A be a *symmetric matrix* and let $x \in \mathbb{R}^n$ be a reasonably good approximation to an eigenvector. Then the quotient

$$R_q = \sigma = \frac{x^T Ax}{x^T x}$$

is a good approximation to the eigenvalue λ for which x is the corresponding eigenvector.

Proof: Because A is symmetric, by Theorem 8.2.8, there exists a set of orthonormal eigenvectors $v_1, v_2, \ldots, v_n$. Therefore, we can write

$$x = c_1 v_1 + \cdots + c_n v_n$$

Again, since v_i, $i = 1, \ldots, n$ are normalized, $v_i^T v_i = 1$. Then, since $Av_i = \lambda_i v_i$, $i = 1, \ldots, n$, and noting that $v_i^T v_j = 0$, $i \neq j$, we have

$$\sigma = \frac{x^T Ax}{x^T x} = \frac{(c_1 v_1 + \cdots + c_n v_n)^T A(c_1 v_1 + \cdots + c_n v_n)}{(c_1 v_1 + \cdots + c_n v_n)^T (c_1 v_1 + \cdots + c_n v_n)}$$

$$= \frac{(c_1 v_1 + \cdots + c_n v_n)^T (c_1 \lambda_1 v_1 + \cdots + c_n \lambda_n v_n)}{c_1^2 + c_2^2 + \cdots + c_n^2}$$

$$= \frac{\lambda_1 c_1^2 + \lambda_2 c_2^2 + \cdots + \lambda_n c_n^2}{c_1^2 + c_2^2 + \cdots + c_n^2}$$

$$= \lambda_1 \left[\frac{1 + \left(\frac{\lambda_2}{\lambda_1}\right)\left(\frac{c_2}{c_1}\right)^2 + \cdots + \left(\frac{\lambda_n}{\lambda_1}\right)\left(\frac{c_n}{c_1}\right)^2}{1 + \left(\frac{c_2}{c_1}\right)^2 + \cdots + \left(\frac{c_n}{c_1}\right)^2} \right]$$

Because of our assumption that x is a good approximation to v_1, c_1 is larger than other c_i, $i = 2, \ldots, n$. Thus, the expression within [ ] is close to 1, which means that σ is close to λ_1. ■

In Theorem 8.5.3 we have seen that the quotient

$$R = \frac{x^T Ax}{x^T x}$$

where x is a real eigenvector of A, is a good approximation to an eigenvalue λ for which x is the corresponding real eigenvector. In general, if x is a nonzero vector,

then the quotient

$$\mu = R_q = \frac{x^T Ax}{x^T x}$$

minimizes

$$\|(A - \mu I)x\|_2$$

Note that the normal equations for the least squares problem of minimizing

$$\|Ax - \mu x\|_2$$

in the unknown μ are given by $x^T x\mu = x^T Ax$.

DEFINITION 8.5.1 The quotient $R_q = \dfrac{x^T Ax}{x^T x}$ is called the **Rayleigh quotient** of the vector x.

Example 8.5.4

$$A = \begin{pmatrix} 1 & 2 \\ 2 & 3 \end{pmatrix}$$

Let

$$x = \begin{pmatrix} 1 \\ -0.5 \end{pmatrix}$$

Then the Rayleigh quotient

$$\sigma = \frac{x^T Ax}{x^T x} = -0.2$$

is a good approximation to the eigenvalue -0.2361.

Note: It can be shown that for a symmetric matrix A, $\lambda_n \le R_q \le \lambda_1$, where λ_n and λ_1 are the smallest and the largest eigenvalue of A, respectively.

Rayleigh-Quotient Iteration

The idea of approximating an eigenvalue can be combined with inverse iteration to compute successive approximations of an eigenvalue and the corresponding eigenvector in an iterative fashion, known as **Rayleigh quotient iteration** and described as follows. Let N be the maximum number of iterations to be performed and x_0 is an initial approximation to an eigenvector.

ALGORITHM 8.5.3 Rayleigh Quotient Iteration

For $k = 0, 1, 2, \ldots,$ do

Step 1: Compute

$$\sigma_k = x_k^T Ax_k / x_k^T x_k$$

Step 2: Solve for $\hat{x}_{k+1}$:

$$(A - \sigma_k I)\hat{x}_{k+1} = x_k$$

Step 3: Compute

$$x_{k+1} = \hat{x}_{k+1} / \max(\hat{x}_{k+1})$$

Step 4: Stop if the pair (σ_k, x_k) is an acceptable eigenvalue–eigenvector pair or if $k > N$.

Flop count and convergence of the Rayleigh quotient iteration. In Step 2 of Algorithm 8.5.3, we need to solve a new system for each value of k. This makes the method quite expensive compared to the cost of inverse iteration—unless, of course, the symmetric matrix A is tridiagonal, in which case (as we have seen in Chapter 6) the cost is only $O(n)$ flops per iteration. But the rate of convergence of the Rayleigh quotient iteration is quite fast.

It is cubic; that is, the number of accurate figures triples per iteration. For details, see Parlett (1980b).

Choice of x_0. As for choosing an initial vector x_0, perhaps the best thing to do is to use the direct power method itself a few times and then use the last approximation as x_0.

Remark: Rayleigh quotient iteration can also be defined in the nonsymmetric case, where one finds both left and right eigenvectors at each step. We omit the discussion of the nonsymmetric case here and refer the reader to Wilkinson (1965, 636). See also Parlett (1974).

Example 8.5.5

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \end{pmatrix}$$

Let us take

$$x_0 = \begin{pmatrix} 0.5246 \\ 0.7622 \\ 1.000 \end{pmatrix}$$

which is obtained after 3 iterations of the power method. Then we have the following.

$k = 0$:

$$\sigma_0 = \frac{x_0^T Ax_0}{x_0^T x_0} = 9.6235$$

$$x_1 = \begin{pmatrix} 0.5247 \\ 0.7623 \\ 1.000 \end{pmatrix}$$

$k = 1$:

$$\sigma_1 = \frac{x_1^T Ax_1}{x_1^T x_1} = 9.6235$$

$$x_2 = \begin{pmatrix} 1.000 \\ 1.4529 \\ 1.9059 \end{pmatrix}$$

The normalized eigenvector associated with 9.6255 is

$$\begin{pmatrix} 0.3851 \\ 0.5595 \\ 0.7339 \end{pmatrix}$$

Note that 0.3851 times x_2 is this eigenvector to three digits. Thus, two iterations were sufficient.

MATCOM Note: Algorithm 8.5.3 has been implemented in MATCOM program RAYQOT.

8.5.4 Computing the Subdominant Eigenvalues and Eigenvectors: Deflation

Once the dominant eigenvalue λ_1 and the corresponding eigenvector v_1 have been computed, the next dominant eigenvalue λ_2 can be computed by using **deflation.** The basic idea behind deflation is to replace the original matrix by another matrix of the same or lesser dimension using a computed eigenpair such that the deflated matrix has the same eigenvalues as the original one except the one that is used to deflate.

Hotelling Deflation

The Hotelling deflation is a process that replaces the original matrix $A = A_1$ by a matrix A_2 of the same order such that all the eigenvalues of A_2 are the same as those of A_1, *except* the one used to construct A_2 from A_1.

Remarks: Although the Hotelling deflation is commonly recommended in the engineering literature as a practical process for computing a selected number of eigenvalues and eigenvectors and is well suited to large and sparse matrices, the method both in the symmetric and nonsymmetric cases is numerically unstable (Wilkinson 1965, 585).

Householder Deflation

We will now construct a deflation using a similarity transformation on A with Householder matrices. Of course, the technique will work with any similarity transformation; however, we will use Householder matrices for reasons of *numerical stability*.

The method is based upon the following result.

THEOREM 8.5.4 Let (λ_1, v_1) be an eigenpair of A and H be a Householder matrix such that Hv_1 is a multiple of e_1; then

$$A_1 = HAH = \begin{pmatrix} \lambda_1 & * & * & \cdots & * \\ 0 & & & & \\ \vdots & & A_2 & & \\ 0 & & & & \end{pmatrix} = \left(\begin{array}{c|c} \lambda_1 & b^T \\ \hline 0 & A_2 \end{array}\right)$$

where A_2 is $(n-1) \times (n-1)$ and the eigenvalues of A_2 are the same as those of A except for λ_1; in particular, if $|\lambda_1| > |\lambda_2| > |\lambda_3| \geq \cdots \geq |\lambda_n|$, then dominant eigenvalue of A_2 is λ_2, which is the second dominant (subdominant) eigenvalue of A.

Proof: From $Av_1 = \lambda_1 v_1$ we have

$$HAHHv_1 = \lambda_1 Hv_1 \qquad (\text{since } H^2 = I)$$

That is, $HAH(ke_1) = \lambda_1 ke_1$ (since $Hv_1 = ke_1$) or $HAHe_1 = \lambda_1 e_1$. (This means that the first column of HAH is λ_1 times the first column of the identity matrix.) Thus HAH must have the form

$$HAH = \begin{pmatrix} \lambda_1 & * & * & * & \cdots & * \\ 0 & & & & & \\ \vdots & & & A_2 & & \\ 0 & & & & & \end{pmatrix}$$

Because $\det(HAH - \lambda I) = \det(\lambda_1 - \lambda)\det(A_2 - \lambda I)$, it follows that the eigenvalues of A_2 are the same as those of A_1 minus λ_1. Moreover, if

$$|\lambda_1| > |\lambda_2| > |\lambda_3| \geq |\lambda_4| \geq \cdots \geq |\lambda_n|$$

the dominant eigenvalue of A_2 is λ_2, which is the second dominant eigenvalue of A. ■

ALGORITHM 8.5.4 Householder Deflation to Compute the Subdominant Eigenvalue

Step 1: Compute the dominant eigenvalue λ_1 and the corresponding eigenvector v_1 using the power method and the inverse power method.

Step 2: Find a Householder matrix H such that

$$Hv_1 = \begin{pmatrix} * \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

Step 3: Compute HAH

Step 4: Discard the first row and the first column of HAH and find the dominant eigenvalue of the $(n-1) \times (n-1)$ matrix thus obtained.

Example 8.5.6

$$A = \begin{pmatrix} 0.2190 & 0.6793 & 0.5194 \\ 0.0470 & 0.9347 & 0.8310 \\ 0.6789 & 0.3835 & 0.0346 \end{pmatrix}$$

The eigenvalues of A are 0.0018, −0.3083, 1.4947.

1. $\lambda_1 = 1.4947$, $v_1 = \begin{pmatrix} -0.5552 \\ -0.7039 \\ -0.4430 \end{pmatrix}$

2. $u = v_1 - \|v_1\|_2 e_1 = \begin{pmatrix} -1.5552 \\ -0.7039 \\ -0.4430 \end{pmatrix}$

$$H = I - \frac{2uu^T}{u^T u} = \begin{pmatrix} -0.5552 & -0.7039 & -0.4430 \\ -0.7039 & 0.6814 & -0.2005 \\ -0.4430 & -0.2005 & -0.8738 \end{pmatrix}$$

$$Hv_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

3. $$HAH = \begin{pmatrix} 1.4977 & -0.3223 & -0.3331 \\ 0 & 0.1987 & 0.2672 \\ 0 & 0.3736 & -0.5052 \end{pmatrix}$$

$$= \left(\begin{array}{c|cc} 1.4977 & -0.3223 & -0.3331 \\ \hline 0 & \multicolumn{2}{c}{A_2} \\ 0 & & \end{array}\right)$$

4. The dominant eigenvalue of A_2 is -0.3083, which is the subdominant eigenvalue of A.

Computing the Subdominant Eigenvector

Once the subdominant eigenvalue λ_2 has been computed using the above procedure, the corresponding eigenvector v_2 can be found from the inverse iteration.

Computing the Other Largest Eigenvalues and Eigenvectors

Once the pair (λ_2, v_2) is computed, the matrix A_2 can be deflated using this pair to compute the pair (λ_3, v_3). From the pair (λ_3, v_3), we then compute the pair (λ_4, v_4) and so on. Thus, by repeated applications of the process, we can compute successively all the n eigenvalues and eigenvectors of A.

Remark: If more than a few eigenvalues and eigenvectors are needed, the QR iteration method to be described a little later should be used, because in that case, the QR iteration will be more cost-effective, and it is a stable method.

Computing the Smallest Eigenvalues

It is easy to see that the power method applied to A^{-1} gives us the smallest eigenvalue in magnitude (the least dominant one) of A.

Let A be nonsingular and let the eigenvalues of A be ordered such that

$$|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \cdots \geq |\lambda_{n-1}| > |\lambda_n| > 0$$

Then the eigenvalues of A^{-1} (which are the reciprocals of the eigenvalues of A) are arranged as

$$\left|\frac{1}{\lambda_n}\right| > \left|\frac{1}{\lambda_{n-1}}\right| \geq \left|\frac{1}{\lambda_{n-2}}\right| \geq \cdots \geq \frac{1}{|\lambda_1|} > 0$$

That is, $1/\lambda_n$ is the dominant eigenvalue of A^{-1}. This fact suggests that the reciprocal of the smallest eigenvalue can be computed by applying the power method to A^{-1}.

ALGORITHM 8.5.5 Computing the Smallest Eigenvalue in Magnitude

Step 1: Apply the power method to A^{-1}.

Step 2: Take the reciprocal of the eigenvalue obtained in Step 1.

Note: Because the power method is implemented by matrix–vector multiplication only, the inverse of A does not have to be computed explicitly. This statement is true because computing $A^{-1}x$, where x is a vector, is equivalent to solving the linear system: $Ay = x$.

To compute the next least dominant eigenvalue (the *second smallest eigenvalue* in magnitude), we compute the smallest eigenvalue and the corresponding eigenvector and then apply *deflation*. The inverse power method applied to the $(n-1) \times (n-1)$ matrix at the bottom right-hand corner will yield the reciprocal of the second smallest eigenvalue of A. Once the required eigenvalues are computed, we can always use the inverse power method to compute the corresponding eigenvectors or use deflation as shown earlier.

Remark: To accelerate the convergence, a suitable shift should be made.

Example 8.5.7

$$A = \begin{pmatrix} 1 & 4 & 5 \\ 2 & 3 & 3 \\ 1 & 1 & 1 \end{pmatrix}$$

The power method (without shift) applied to A^{-1} with the starting vector $x_0 = (1, -1, 1)^T$ gave $\lambda = 9.5145$ after 5 iterations. Thus, the smallest eigenvalue of A is

$$\frac{1}{\lambda} = 0.1051$$

(Note the eigenvalues of A are 6.3850, -1.4901, and 0.1051.)

Summary of the Process for Finding the Selected Eigenvalues and Eigenvectors

To compute a selected number of eigenvalues (mainly the first few largest or smallest eigenvalues and the corresponding eigenvectors), the following combination of the power method, inverse iteration, and deflation can be used in the following sequence:

1. Use the power method (the power method applied to A^{-1}) to compute a reasonably good approximation of the largest (smallest) eigenvalue in magnitude and of the corresponding eigenvectors.
2. Use the inverse iteration with the approximate eigenvalue (keeping it fixed) and the approximate eigenvector obtained in step 1 as the starting vector.
3. Apply deflation to compute the next set of eigenpairs.

8.6 Similarity Transformations and Eigenvalue Computations

Recall (Theorem 8.2.3) that two similar matrices A and B have the same eigenvalues; that is, if X is a nonsingular matrix such that

$$X^{-1}AX = B$$

then the eigenvalues of A are the same as those of B.

One obvious approach to compute the eigenvalues of A is, therefore, to reduce A to a suitable "simpler" form B by similarity so that the eigenvalues of B can be more easily computed. However, *extreme caution* must be taken here. It can be shown (Golub and Van Loan 1989, 340) that

THEOREM 8.6.1 $\mathrm{fl}(X^{-1}AX) = X^{-1}AX + E$, where

$$\|E\|_2 \approx \mu\|X\|_2\|X^{-1}\|_2\|A\|_2$$

Remark: Because the error matrix E clearly depends upon $\mathrm{Cond}_2(X)$, Theorem (8.6.1) tells us that in computing the eigenvalues and eigenvectors, we should avoid ill-conditioned transforming matrices to transform A by similarity to a "simpler" form. Primarily because of this, the eigenvalues of a matrix A are not computed using the characteristic polynomial of A or by transforming A to the Jordan canonical form. We discuss them in some detail in the following paragraphs.

8.6.1 Difficulties with Using the Characteristic Polynomial

Problems in Using the Characteristic Polynomial

Because the eigenvalues of a matrix are the zeros of the characteristic polynomial, it is natural to think of computing the eigenvalues of A by finding the zeros of its characteristic polynomial. However, this approach is *not* numerically effective.

- First, the process of explicitly computing the coefficients of the characteristic polynomial may be numerically unstable.
- Second, the zeros of the characteristic polynomial may be very sensitive to perturbations on the coefficients of the characteristic polynomial. Thus, if the coefficients of the characteristic polynomial are not computed accurately, there will be errors in the computed eigenvalues.

In Chapter 3 we illustrated the sensitivity of the root-finding problem by means of the Wilkinson polynomial and other examples. We now discuss the difficulty of computing the characteristic polynomial in some detail.

Computing the characteristic polynomial explicitly amounts to transforming the matrix to a direct sum of companion matrices. Every matrix A can be reduced by similarity to

$$C = \begin{pmatrix} C_1 & & 0 \\ & \ddots & \\ 0 & & C_k \end{pmatrix}$$

Frobenius canonical form

where each C_i is a **companion matrix.** The matrix C is said to be in Frobenius form. If $k = 1$, the matrix A is **nonderogatory**.

Assume that A is nonderogatory; let's see how A can be reduced to a companion matrix by similarity. This can be achieved in two stages.

Reduction of a Matrix to a Companion Matrix

Stage 1: The matrix A is transformed to an unreduced Hessenberg matrix H by orthogonal similarity using the Householder or Givens method.

Stage 2: The transformed unreduced Hessenberg matrix H is further reduced to a companion matrix by similarity.

We have already seen in Chapter 5 how a matrix A can be transformed to an upper Hessenberg matrix by orthogonal similarity in a *numerically stable way* using the Householder or Givens method.

Consider now stage 2—that is, the transformation of the unreduced Hessenberg matrix H to a companion matrix C.

Let X be the nonsingular transforming matrix, that is $HX = XC$, where

$$C = \begin{pmatrix} 0 & 0 & 0 & \cdots & 0 & c_1 \\ 1 & 0 & 0 & \cdots & 0 & c_2 \\ 0 & 1 & 0 & \cdots & 0 & c_3 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & c_n \end{pmatrix}.$$

If $x_1, x_2, \ldots, x_n$ are the n successive columns of X, then from

$$HX = XC$$

it follows that if x_1 is chosen, x_2 through x_n can be determined by the recursion

$$x_{i+1} = Hx_i, \qquad i = 1, \ldots, n-1$$

Thus, it follows that if x_1 is chosen so that

$$X = (x_1, Hx_1, \ldots, Hx_{n-1})$$

is nonsingular, then X will transform H to C by similarity.

Choose $x_1 = (1, 0, \ldots, 0)^T$; then the upper triangular matrix

$$X = (x_1, Hx_1, \ldots, Hx_{n-1}) = \begin{pmatrix} 1 & * & * & \cdots & * \\ 0 & h_{21} & * & \cdots & * \\ 0 & 0 & h_{21}h_{32} & \cdots & * \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots\ 0 & h_{21}h_{32}\ldots h_{n,n-1} \end{pmatrix}$$

is nonsingular because $h_{i+1,i} \neq 0$, $i = 1, \ldots, n-1$.

So, *if one or more subdiagonal entries $h_{i+1,i}$ of H are significantly small, then the inverse of X will have large entries; consequently X will be ill conditioned.*

Thus, in such a case the transformation of an unreduced Hessenberg matrix H to a companion matrix will be unstable. We now summarize:

> The first stage, in which A is transformed to H using the Householder or the Givens method, is numerically stable, whereas the second stage, in which H is further reduced to C, can be highly unstable.

Example 8.6.1

$$H = \begin{pmatrix} 1 & 2 & 3 \\ 0.0001 & 1 & 1 \\ 0 & 2 & 3 \end{pmatrix}$$

$$x_1 = (1, 0, 0)^T, \qquad x_2 = Hx_1 = (1, 0.0001, 0)^T$$

$$x_3 = Hx_2 = (1.0002, 0.0002, 0.0002)^T$$

$$X = \begin{pmatrix} 1 & 1 & 1.0002 \\ 0 & 0.0001 & 0.0002 \\ 0 & 0 & 0.0002 \end{pmatrix}$$

$$X^{-1}HX = C = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & -4.9998 \\ 0 & 1 & 5 \end{pmatrix}$$

$$\text{Cond}_2(X) = 3.1326 \times 10^4$$

(Note that the existence of a small subdiagonal entry of H—namely, h_{21}—made the transforming matrix X ill conditioned.)

There are also other equivalent methods for finding the characteristic polynomial, such as LeVerrier's method, which are also unstable. For details, see Wilkinson (1965, 434).

Having emphasized the danger of using the Frobenius form in the eigenvalue-computations of a matrix, let's point out some remarks of Wilkinson about Frobenius forms of matrices arising in certain applications such as mechanical and electrical systems.

> Although we have made it clear that we regard the use of the Frobenius form as dangerous, in that it may well be catastrophically worse-conditioned than the original matrix, we have found the program based on its use surprisingly satisfactory in general for matrices arising from damped mechanical or electrical systems. It is common for the corresponding characteristic polynomial to be well-conditioned. When this is true methods based on the use of the explicit characteristic polynomial are both fast and accurate. (Wilkinson 1965, 482)

Remark: The preceding remarks of Wilkinson clearly support a long tradition by engineers of computing the eigenvalues via the characteristic polynomial.

8.6.2 Eigenvalue Computations via the Jordan Canonical Form

Let us now discuss the use of some other obvious canonical forms in eigenvalue computations. In this connection, of course, the Jordan canonical form is the one that comes to mind first. Recall (Theorem 8.2.10) that given an $n \times n$ matrix A, there exists a nonsingular matrix X such that

$$X^{-1}AX = \text{diag}(J_1, \ldots, J_k)$$

where

$$J_i = \begin{pmatrix} \lambda_i & 1 & & 0 \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ 0 & & & \lambda_i \end{pmatrix}$$

If J_i is of order ν_i, then

$$\nu_1 + \nu_2 + \cdots + \nu_k = n$$

The matrix on the right-hand side is the Jordan canonical form (JCF) of A, λ_i's are the eigenvalues. Thus, the eigenvalues of A are displayed as soon as the JCF is computed.

Unfortunately, this computation can also be highly unstable.

> If A is close to a nondiagonalizable matrix, the transforming matrix X can be very ill conditioned (Golub and Van Loan 1989, 339–40). It then follows from Theorem 8.6.1 that the computed JCF in that case will be inaccurate.

8.7 Eigenvalue Sensitivity

In the previous two sections we cautioned the readers about the danger of computing the eigenvalues via the Jordan canonical form or the Frobenius form of a matrix. The danger was mainly the possibility of the transforming matrix X being ill conditioned.

In this section we see what specific role the condition number of the transforming matrix, $\text{Cond}(X) = \|X\| \cdot \|X^{-1}\|$, plays in the eigenvalue sensitivity. We have illustrated the eigenvalue sensitivity phenomenon by means of some examples in Chapter 3. Here is a more specific result, known as the *Bauer–Fike theorem*, on the eigenvalue sensitivity of a diagonalizable matrix.

8.7.1 The Bauer–Fike Theorem

THEOREM 8.7.1 Let A be diagonalizable; that is, there exists a nonsingular matrix X such that $X^{-1}AX$ is a diagonal matrix D. Then for an eigenvalue λ of $A + E$, we have

$$\min |\lambda_i - \lambda| \le \|X\| \, \|X^{-1}\| \, \|E\|$$

where $\| \, \|$ is a subordinate matrix norm and $\lambda_1, \lambda_2, \ldots, \lambda_n$ are the eigenvalues of A.

Proof: Consider two cases.

Case 1: $\lambda = \lambda_i$ for some i. The theorem is trivially true.

Case 2: $\lambda \neq \lambda_i$ for any i. Then the diagonal entries of the diagonal matrix $\lambda I - D$ are different from zero. Because the determinant of a matrix is equal to the product of its eigenvalues, the matrix $(\lambda I - D)$ is nonsingular. From

$$(A + E)x = \lambda x$$

we have

$$\begin{aligned} Ex &= (\lambda I - A)x \\ &= (\lambda I - XDX^{-1})x \qquad \text{(note that } A = XDX^{-1}) \\ &= X(\lambda I - D)X^{-1}x \end{aligned}$$

Set $X^{-1}x = y$. Then from the last equation, we have, by multiplying the equation by X^{-1} to the left,

$$(\lambda I - D)y = X^{-1}Ex$$

or

$$y = (\lambda I - D)^{-1}X^{-1}EXy \qquad \text{(note that } x = Xy)$$

Taking a subordinate norm on both sides, we have

$$\begin{aligned} \|y\| &= \|(\lambda I - D)^{-1}X^{-1}EXy\| \\ &\leq \|(\lambda I - D)^{-1}\|\|X^{-1}EX\|\|y\| \end{aligned}$$

Dividing both sides by $\|y\|$, we get

$$1 \leq \|(\lambda I - D)^{-1}\| \, \|X^{-1}\|\|E\| \, \|X\|$$

For a subordinate norm,

$$\begin{aligned} \|(\lambda I - D)^{-1}\| &= \max_i \left|\frac{1}{\lambda - \lambda_i}\right| \\ &= \frac{1}{\min_i |\lambda - \lambda_i|} \end{aligned}$$

So,

$$1 \leq \frac{1}{\min_i |\lambda - \lambda_i|}\|X^{-1}\|\|X\|\|E\|$$

or

$$\min |\lambda - \lambda_i| \leq \|X^{-1}\| \, \|X\| \, \|E\|$$

■

Implications of the Theorem

Theorem 8.7.1 tells us that if

$$\text{Cond}(X) = \|X\|^{-1}\|X^{-1}\|$$

is large, then an eigenvalue λ of the perturbed matrix $A + E$ can be significantly different from an eigenvalue λ_i of A. In general, the more ill-conditioned the eigenvector matrix X is, the more ill-conditioned will be the eigenproblem for A.

Remark: If case A is not diagonalizable, a similar result also holds. (For details, see Golub and Van Loan (1989, 342–343.))

Example 8.7.1

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 0 & 0.9990 & 1 \\ 0 & 0 & 2 \end{pmatrix}$$

The eigenvalues of A are 1, 0.9990, and 2.

$$X = \begin{pmatrix} 1 & -1 & 0.9623 \\ 0 & 0.0005 & 0.1923 \\ 0 & 0 & 0.1925 \end{pmatrix}$$

$$X^{-1}AX = \text{diag}(1, 0.9990, 2)$$

Let

$$E = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 10^{-5} & 0 & 0 \end{pmatrix}$$

The eigenvalues of $A + E$ are $0.9995 + 0.0044i$, $0.9995 - 0.0044i$, and 2.

A change of 10^{-5} *in the* $(3, 1)$*th entry of A completely changed the first two eigenvalues.* This is because, $\text{Cond}(X) = 6.8708 \times 10^3$ (large!). Note that $\text{Cond}(X) \cdot \|E\| = 0.0687$. Thus, by the Bauer–Fike theorem, such a change was expected. The question that arises, however, is *why did the first two eigenvalues of A change and not the third one?* We will explain this in Example 8.7.2.

8.7.2 Sensitivity of the Individual Eigenvalues

The condition number $\|X\| \, \|X^{-1}\|$ gives an overall assessment of the changes in eigenvalues with respect to changes in the coefficient of the matrix. However, as we have seen from the examples in Chapter 3, some eigenvalues of A may be more sensitive than the others. In fact, some may be very well conditioned, whereas others are ill conditioned. Similarly, some eigenvectors may be well conditioned, whereas others are not.

It is, therefore, more appropriate to talk about conditioning of the individual eigenvalues than about conditioning of the eigenvalue problem. Recall that in Chapter 3 an analysis of the ill-conditioning of the individual eigenvalues of the slightly perturbed Wilkinson matrix was given in terms of the numbers s_i. In general, this can be done for any diagonalizable matrix.

Let $X^{-1}AX = \text{diag}(\lambda_1, \ldots, \lambda_n)$. Then the normalized right-hand and left-hand eigenvectors corresponding to an eigenvalue λ_i are given by

$$x_i = \frac{Xe_i}{\|Xe_i\|_2}, \qquad y_i = \frac{(X^{-1})^T e_i}{\|(X^{-1})^T e_i\|_2}$$

DEFINITION 8.7.1 The number $1/s_i$, where s_i is defined by

$$s_i = |y_i^T x_i|$$

is called the **condition number** of the eigenvalue λ_i.

If this number is large, then λ_i is an ill-conditioned eigenvalue. *This typically happens when A is close to a matrix having some nonlinear elementary divisors.*

Notes:

1. There are n condition numbers associated with the n eigenvalues of A.
2. If x_i and y_i are real, then s_i is the cosine of the angle between x_i and y_i.

Example 8.7.2

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 0 & 0.999 & 1 \\ 0 & 0 & 2 \end{pmatrix}$$

$$\begin{aligned}
x_1 &= (1, 0, 0)^T \\
x_2 &= (1, -0.0005, 0)^T \\
x_3 &= (0.9623, 0.1923, 0.1925)^T \\
y_1 &= (0.0004, 0.7066, -0.7076)^T \\
y_2 &= (0, -0.7075, 0.7068)^T \\
y_3 &= (0, 0, 1)^T \\
s_1 &= |y_1^T x_1| = 3.5329 \times 10^{-4} \\
s_2 &= |y_2^T x_2| = 3.5373 \times 10^{-4} \\
s_3 &= |y_3^T x_3| = 0.1925
\end{aligned}$$

Thus, these computations clearly show that $\lambda_1 = 1$, $\lambda_2 = 0.999$ are ill conditioned, whereas $\lambda_3 = 2$ is well conditioned.

Indeed, when $a(3, 1)$ is perturbed to 0.000001 and the eigenvalues of the perturbed matrix are computed, the first two eigenvalues of the perturbed matrix (those corresponding to 1 and 0.999 of the original matrix) become complex. The computed

eigenvalues of the perturbed matrix are (to three digits):

$$0.999 + 0.001i, \qquad 0.999 - 0.001i, \qquad \text{and} \quad 2$$

For yet another nontrivial example, see Exercise 23.

A Relationship Between s_i and $\text{Cond}_2(X)$

It is easy to see that the numbers s_i and $\text{Cond}_2(X)$ with respect to the 2-norm are related.

$$\begin{aligned}
s_i = |y_i^T x_i| &= \frac{|e_i^T X^{-1} X e_i|}{\|Xe_i\|_2 \|(X^{-1})^T e_i\|_2} \\
&= \frac{1}{\|Xe_i\|_2 \|(X^{-1})^T e_i\|_2}
\end{aligned}$$

Now

$$\|Xe_i\|_2 \le \|X\|_2 \|e_i\|_2 = \|X\|_2$$

and

$$\begin{aligned}
\|(X^{-1})^T e_i\|_2 &\le \|(X^{-1})^T\|_2 \|e_i\|_2 \\
&= \|(X^{-1})^T\|_2 = \|X^{-1}\|_2
\end{aligned}$$

So,

$$\frac{1}{s_i} \le \|X\|_2 \|X^{-1}\|_2 = \text{Cond}_2(X)$$

Thus, for each i,

$$\frac{1}{s_i} \le \text{Cond}_2(X)$$

Similarly, it is possible to prove that there exists a matrix X of eigenvectors such that

$$\text{Cond}_2(X) \le \sum_{i=1}^{n} |s_i^{-1}|$$

See Wilkinson (1965, 1989).

Example 8.7.3

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 0 & 0.999 & 1 \\ 0 & 0 & 2 \end{pmatrix}$$

$$\frac{1}{s_1} = 2.8305 \times 10^3$$

$$\frac{1}{s_2} = 2.8270 \times 10^3$$

$$\frac{1}{s_3} = 5.1940, \quad \text{let } X = \begin{pmatrix} 53.20 & -53.17 & 2.193 \\ 0 & 0.0266 & 0.4383 \\ 0 & 0 & 0.4388 \end{pmatrix}$$

$$\text{Cond}_2(X) = 5.6577 \times 10^3$$

Thus,

$$\frac{1}{s_i} < \text{Cond}_2(X), \qquad i = 1, 2, 3$$

and $\text{Cond}_2(x) < s_1^{-1} + s_2^{-1} + s_3^{-1}$.

The Condition Numbers and Linear Dependence of Eigenvectors

For a diagonalizable matrix, the columns of the matrix X are the eigenvectors of A, so $\text{Cond}_2(X)$ gives us an indication of how linearly independent the eigenvectors are. *If* $\text{Cond}_2(X)$ *is large, it means that there are nearly dependent eigenvectors.*

For the preceding matrix A, a matrix X of eigenvectors is:

$$X = \begin{pmatrix} 1 & 1 & 0.9623 \\ 0 & -0.0005 & 0.1923 \\ 0 & 0 & 0.1925 \end{pmatrix}$$

Note the linear dependence of the first two eigenvectors.

$$\text{Cond}_2(X) = 6.8708 \times 10^3$$

The Eigenvalue Sensitivity of a Normal Matrix

A matrix A is called **normal** if $AA^* = A^*A$ where $A^* = (\bar{A})^T$. Using the Schur triangularization theorem (Theorem 8.2.6) it can easily be shown (Exercise 19) that if A is a normal matrix then there exists a unitary matrix U such that U^*AU is diagonal. Thus, an immediate consequence of the Bauer-Fike theorem is the following.

COROLLARY 8.7.1 Corollary to the Bauer–Fike Theorem Let A be a normal matrix and $\lambda_1, \ldots, \lambda_n$ be the eigenvalues of A. Then for an eigenvalue λ of $A + E$, we have

$$\min |\lambda_i - \lambda| \leq \|E\|_2$$

> **Eigenvalue Sensitivity of a Normal Matrix**
> In other words, the eigenvalues of a normal matrix are perfectly well-conditioned.

Remark: The normal matrices most commonly found in practical applications are symmetric (or Hermitian, if complex) matrices. Thus, by the preceding corollary, *the eigenvalues of a symmetric (or Hermitian) matrix are well conditioned.* We discuss the symmetric eigenvalue problem in more detail in Section 8.11.

MATCOM Note: MATCOM program SENSEIG computes a vector containing the reciprocals of the condition numbers of the eigenvalues of a matrix.

8.8 Eigenvector Sensitivity

We shall not go into details in this discussion on the sensitivity of eigenvectors. We just state a theorem (in a somewhat crude form) that highlights the main differences between eigenvalue and eigenvector sensitivities. For an exact statement and proof, see Watkins (1991, 332–333).

THEOREM 8.8.1 Let ΔA be a very small perturbation of A and let the eigenvalue λ_k of A be perturbed by $\delta\lambda_k$; that is, $\lambda_k + \delta\lambda_k$ is an eigenvalue of $A + \Delta A$. Let $x_k + \delta x_k$ be the eigenvector corresponding to $\lambda_k + \delta\lambda_k$. Then, assuming that the eigenvalues of A are all distinct, we have

$$x_k + \delta x_k = x_k + \sum_{j \neq k} \frac{\alpha_{jk}}{(\lambda_k - \lambda_j)s_j} x_j + O(\|\Delta A\|^2)$$

where

$$\alpha_{jk} = y_j^T(\Delta A)x_k,$$

and x_k and y_k are normalized eigenvectors of A, asociated with λ_k.

Implications of the Theorem

Theorem 8.8.1 tells us that *if A is perturbed by a small amount, then the amount of perturbation an eigenvector x_k experiences is determined by*

1. Condition numbers of all the eigenvalues other than λ_k, and
2. The distance of λ_k from the other eigenvalues.

An immediate consequence of this theorem is that if there is a multiple eigenvalue or an eigenvalue near another eigenvalue, then there are some ill-conditioned eigenvectors. This is significant especially for a Hermitian or a symmetric matrix, because we know that *the eigenvalues of a Hermitian matrix are all well conditioned, but the eigenvectors could be ill conditioned.* If the eigenvalues are well separated and well conditioned, then the eigenvectors are well conditioned.

Example 8.8.1

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0.99 & 0 \\ 0 & 0 & 2 \end{pmatrix}$$

$$A' = A + \Delta A = \begin{pmatrix} 1 & 0.0001 & 0 \\ 0.0001 & 0.99 & 0 \\ 0 & 0 & 2 \end{pmatrix}$$

The eigenvalues of $A + \Delta A$ are 1, 0.99, and 2. (There is little change; because A is symmetric, the eigenvalues are well conditioned.) However, the eigenvectors of A' are

$$\begin{pmatrix} -1 \\ -0.01 \\ 0 \end{pmatrix}, \quad \begin{pmatrix} 0.01 \\ -1 \\ 0 \end{pmatrix}, \quad \text{and} \quad \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

whereas those of A are

$$\begin{pmatrix}1\\0\\0\end{pmatrix}, \quad \begin{pmatrix}0\\1\\0\end{pmatrix}, \quad \text{and} \quad \begin{pmatrix}0\\0\\1\end{pmatrix}$$

Note that the eigenvector corresponding to $\lambda_3 = 2$ has not changed, whereas the other two eigenvectors have changed because of the proximity of the eigenvalues 1 and 0.99 and the structure of the perturbation.

8.9 The Real Schur Form and QR Iterations

In the preceding discussions we have seen that computing eigenvalues of A via reduction of A to the Frobenius or to the Jordan canonical form is not numerically effective. If the transforming matrix is ill conditioned, then there may be large errors in the computed canonical form; this in turn will introduce large errors in the eigenvalues.

A question therefore arises as to whether we can obtain a similarity reduction of A to a suitable canonical form using a well-conditioned transforming matrix.

A perfectly well-conditioned matrix, for example, is an orthogonal matrix (or unitary, if it is complex), the condition number (with respect to 2-norm and F-norm) of such a matrix being 1.

Indeed, if a matrix A is transformed to a matrix B using a unitary similarity transformation, then a perturbation in A will result in a perturbation in B of the same magnitude; that is, if

$$B = U^*AU$$

and

$$U^*(A + \Delta A)U = B + \Delta B$$

then

$$\|\Delta B\|_2 = \|\Delta A\|_2$$

Example 8.9.1

$$A = \begin{pmatrix}1 & 2 & 3\\3 & 4 & 5\\6 & 7 & 8\end{pmatrix}, \qquad U = \begin{pmatrix}-0.5774 & -0.5774 & -0.5774\\-0.5774 & 0.7887 & -0.2113\\-0.5774 & -0.2113 & 0.7887\end{pmatrix}$$

$$B = \begin{pmatrix}13 & -0.6340 & -2.3660\\-0.9019 & 0 & 0\\-6.0981 & 0 & 0\end{pmatrix}$$

$$\Delta A = 10^{-5} \times I_{3\times 3}$$

$$A_1 = A + \Delta A = \begin{pmatrix}1.00001 & 2 & 3\\3 & 4.00001 & 5\\6 & 7 & 8.00001\end{pmatrix}$$

$$B_1 = U^*(A + \Delta A)U = \begin{pmatrix}13.00001 & -0.633974 & -2.3660\\-0.9019 & 0.00001 & 0\\-6.0981 & 0 & 0.00001\end{pmatrix}$$

$$\Delta B = B_1 - B = 10^{-5} \times I_{3\times 3}$$

$$\|\Delta A\| = \|\Delta B\| = 10^{-5}$$

A perfect canonical form displaying the eigenvalues is a triangular form (the diagonal entries are the eigenvalues). In this context we now recall a classical result due to Schur (Theorem 8.2.6). We restate this important result next and give a proof.

THEOREM 8.9.1 The Schur Triangularization Theorem (restatement of Theorem 8.2.6) If A is an $n \times n$ matrix, then there exists a unitary matrix U such that

$$U^*AU = T$$

where T is a triangular matrix with the eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$ as the diagonal entries.

Proof: We prove the theorem using induction on n.

If $n = 1$, the theorem is trivially true. Next assume the theorem is true for $n - 1$; then we will show that it is also true for n.

Let u be an eigenvector of A with $\|u\|_2 = 1$ associated with an eigenvalue λ_1. Then we can always choose an $n \times (n-1)$ matrix V such that

$$U_1 = (u, V)$$

is unitary. Because $AU_1 = A(u, V) = (\lambda_1 u, AV)$, we have

$$A_1 = U_1^*AU_1 = \begin{pmatrix}u^*\\V^*\end{pmatrix}(\lambda_1 u, AV) = \left(\begin{array}{c|ccc}\lambda_1 & * & * & *\\\hline 0 & & & \\ \vdots & & \hat{A} & \\ 0 & & & \end{array}\right)$$

where $\hat{A} = V^*AV$ is $(n-1) \times (n-1)$. Except λ_1, the eigenvalues of A and $\hat{A}$ are the same. By our hypothesis then there exists unitary matrix V_1 of order $(n-1)$ such that

$$\hat{T} = V_1^*(\hat{A})V_1$$

is triangular. Then, defining

$$U_2 = \left(\begin{array}{c|ccc}1 & 0 & \cdots & 0\\\hline 0 & & & \\ \vdots & & V_1 & \\ 0 & & & \end{array}\right)$$

we see that U_2 is unitary (because V_1 is so), and

$$\begin{aligned}U_2^*A_1U_2 &= U_2^*U_1^*AU_1U_2\\ &= U^*AU, \qquad \text{where } U = U_1U_2.\end{aligned}$$

So,

$$U^*AU = \left(\begin{array}{c|ccc}\lambda_1 & * & \cdots & *\\\hline 0 & & & \\ \vdots & & V_1^*\hat{A}V_1 & = \hat{T} \\ 0 & & & \end{array}\right)$$

Because $\hat{T}$ is triangular, so is U^*AU. Because the eigenvalues of a triangular matrix appear on the diagonal, we are done. ■

Because a real matrix can have complex eigenvalues (occurring in complex conjugate pairs), even for a real matrix A, U and T in the Schur theorem can be complex. However, we can choose U to be real orthogonal if T is replaced by a quasi-triangular matrix R, known as the *real Schur form (RSF)* of A.

THEOREM 8.9.2 The Real Schur Triangularization Theorem Let A be an $n \times n$ real matrix. Then there exists an $n \times n$ orthogonal matrix Q such that

$$Q^T AQ = R = \begin{pmatrix} R_{11} & R_{12} & \cdots & R_{1k} \\ 0 & R_{22} & \cdots & R_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & R_{kk} \end{pmatrix}$$

where each R_{ii} is either a scalar or a 2×2 matrix. The scalar diagonal entries correspond to real eigenvalues, and 2×2 matrices on the diagonal correspond to complex conjugate eigenvalues.

Proof: We give here a brief sketch of the proof. Details can be found in Golub and Van Loan (1989, 362).

The proof is by induction. Suppose that there are s number of complex eigenvalues of A. Then for $s = 0$, the theorem follows from Theorem 8.9.1. (Note that since all the eigenvalues are real, U in Theorem 8.9.1 is real orthogonal).

Consider now $s \geq 1$. Since the complex eigenvalues occur in conjugate pairs, it is easy to see that there exists an $n \times n$ real orthogonal matrix U_1, such that

$$U_1^T AU_1 = \begin{pmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{pmatrix}$$

where A_{11} is 2×2 and has a pair of complex conjugate eigenvalues. Analogously, by induction, there exists an orthogonal matrix U_2 of order $(n-2)$, such that $U_2^T A_{22} U_2$ has the above structure.

Set now

$$Q = U_1 \begin{pmatrix} I_2 & 0 \\ 0 & U_2 \end{pmatrix}$$ ■

DEFINITION 8.9.1 The matrix R in Theorem 8.9.2 is known as the **real Schur form** (RSF) of A.

Notes:

1. The 2×2 matrices on the diagonal are usually referred to as "bumps."
2. The RSF and the Schur form (T of Theorem 8.9.1) are not unique.

Remark: Because the proofs of both the theorems are based on the knowledge of eigenvalues and eigenvectors of the matrix A, they can not be considered to be constructive. They do not help us in computing the eigenvalues and eigenvectors.

We present a method known as the QR iteration method for computing the real Schur form of A. A properly implemented QR method is widely used today for computing the eigenvalues of an arbitrary matrix. As the name suggests, the method is based on the QR factorization and is iterative in nature. The QR iteration method was proposed in algorithmic form by J. G. Francis (1961), though its roots can be traced to a work of H. Rutishauser (1958). The method was also independently discovered by the Russian mathematician V. N. Kublanovskaya (1961).

Note: Because the eigenvalues of a matrix A are the n zeros of the characteristic polynomial and it is well known (proved by Abel and Galois more than a century ago) that the roots of an arbitrary polynomial equation of degree higher than four cannot be found in a finite number of steps, any numerical eigenvalue method for an arbitrary matrix has to be iterative in nature.

8.9.1 The Basic QR Iteration

We first present the basic QR iteration method.

Set $A_0 = A$

Now compute a sequence of matrices (A_k) defined by

$$A_0 = Q_0 R_0$$
$$A_1 = R_0 Q_0 = Q_1 R_1$$
$$A_2 = R_1 Q_1 = Q_2 R_2$$

and so on. In general,

$$A_k = Q_k R_k = R_{k-1} Q_{k-1} \qquad k = 1, 2, \ldots$$

The matrices in the sequence $\{A_k\}$ have a very interesting property. *Each matrix in the sequence is orthogonally similar to the previous one and is, therefore, orthogonally similar to the original matrix.* It is easy to see this. For example,

$$A_1 = R_0 Q_0 = Q_0^T A_0 Q_0 \qquad \text{(because } Q_0^T A_0 = R_0\text{)}$$
$$A_2 = R_1 Q_1 = Q_1^T A_1 Q_1$$

Thus, A_1 is orthogonally similar to A, and A_2 is orthogonally similar to A_1. Therefore, A_2 is orthogonally similar to A, as the following computation shows:

$$A_2 = Q_1^T A_1 Q_1 = Q_1^T (Q_0^T A_0 Q_0) Q_1 = (Q_0 Q_1)^T A_0 (Q_0 Q_1)$$

Because each matrix is orthogonally similar to the original matrix A and, therefore, has the same eigenvalues as A, then if the sequence $\{A_k\}$ converges to a triangular or quasi-triangular matrix, we will be done. The following result shows that, under certain conditions, this happens (Wilkinson 1965, 518–519).

A Condition for Convergence

THEOREM 8.9.3 A Convergence Theorem for Basic QR Iteration Let the eigenvalues $\lambda_1, \ldots, \lambda_n$ be such that $|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n|$, and let the eigenvector matrix Y of the left eigenvectors (that is, $Y = (X^T)^{-1}$) be such that its leading principal minors are nonzero. Then $\{A_k\}$ converges to an upper triangular matrix or to the real Schur form.

In fact, it can be shown that under the preceding conditions, the first column of A_k approaches a multiple of e_1. Thus, for sufficiently large k we get

$$A_k = \begin{pmatrix} \lambda_1 & u \\ 0 & \hat{A}_k \end{pmatrix}$$

We can apply the QR iteration again to $\hat{A}_k$ and the process can be continued to see that the sequence converges to an upper triangular matrix.

Example 8.9.2

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

Eigenvalues of A are 5.3723, and -0.3723; $|\lambda_1| > |\lambda_2|$.

$k = 0$:

$$A_0 = A = Q_0 R_0$$

$$Q_0 = \begin{pmatrix} -0.3162 & -0.9487 \\ -0.9487 & 0.3162 \end{pmatrix}$$

$$R_0 = \begin{pmatrix} -3.1623 & -4.4272 \\ 0 & -0.6325 \end{pmatrix}$$

$k = 1$:

$$A_1 = R_0 Q_0 = \begin{pmatrix} 5.2 & 1.6 \\ 0.6 & -0.2 \end{pmatrix} = Q_1 R_1$$

$$Q_1 = \begin{pmatrix} -0.9934 & -0.1146 \\ -0.1146 & -0.9934 \end{pmatrix}$$

$$R_1 = \begin{pmatrix} -5.2345 & -1.5665 \\ 0 & -0.3821 \end{pmatrix}$$

$k = 2$:

$$A_2 = R_1 Q_1 = \begin{pmatrix} 5.3796 & -0.9562 \\ 0.0438 & -0.3796 \end{pmatrix} = Q_2 R_2$$

(Note that we have already made some progress toward obtaining the eigenvalues.)

$$Q_2 = \begin{pmatrix} -1 & -0.0082 \\ -0.0081 & 1 \end{pmatrix}$$

$$R_2 = \begin{pmatrix} -5.3797 & 0.9593 \\ 0 & -0.3718 \end{pmatrix}$$

$k = 3$:

$$A_3 = R_2 Q_2 = \begin{pmatrix} 5.3718 & 1.0030 \\ 0.0030 & -0.3718 \end{pmatrix} = Q_3 R_3$$

$$Q_3 = \begin{pmatrix} -1 & -0.0006 \\ -0.0006 & 1 \end{pmatrix}$$

$$R_3 = \begin{pmatrix} -5.3718 & -1.0028 \\ 0 & -0.3723 \end{pmatrix}$$

$k = 4$:

$$A_4 = R_3 Q_3 = \begin{pmatrix} 5.3723 & -0.9998 \\ 0.0002 & -0.3723 \end{pmatrix}$$

MATCOM Note: Basic QR iteration as described above has been implemented in MATCOM program QRITRB.

8.9.2 The Hessenberg QR Iteration

The QR iteration method as just presented is not efficient if the matrix A is full and dense. We have seen before that the QR factorization of such a matrix A requires $O(n^3)$ flops; thus, n iterations of QR method will require $O(n^4)$ flops, making the method impractical.

Fortunately, something simple can be done.

> Reduce the matrix A to a Hessenberg matrix by orthogonal similarity before starting the QR iterations. An interesting practical consequence of this is that if $A = A_0$ is initially reduced to an upper Hessenberg matrix and is assumed to be unreduced, then each member of the sequence $\{A_k\}$ is also upper Hessenberg.

This result can be seen as follows. Suppose A_k is an unreduced upper Hessenberg matrix and Givens rotations are used to factor A_k as $Q_k R_k$. Then

$$Q_k = (J(n-1, n, \theta) \cdots J(2, 3, \theta) J(1, 2, \theta))^T$$

is also upper Hessenberg, because $J(i, i+1, \theta)$ has off-diagonal entries only in the positions $(i+1, i)$ and $(i, i+1)$.

Thus, because R_k is upper triangular, $A_{k+1} = R_k Q_k$ is also upper Hessenberg.

Because the QR factorization of a Hessenberg matrix requires only $O(n^2)$ flops, the QR iteration method with the initial reduction of A to Hessenberg will be bounded by $O(n^3)$, making the method efficient.

Example 8.9.3

$$A = A_0 = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 5 \\ 0 & 1 & 1 \end{pmatrix}$$

$$Q_0 = \begin{pmatrix} -0.4472 & 0.3651 & -0.8165 \\ -0.8944 & -0.1826 & 0.4082 \\ 0 & 0.9129 & 0.4082 \end{pmatrix}$$

$$R_0 = \begin{pmatrix} -2.2361 & -3.5777 & -5.8138 \\ 0 & 1.0954 & 1.0954 \\ 0 & 0 & -0.0000 \end{pmatrix}$$

$k = 1$:

$$A_1 = R_0 Q_0 = \begin{pmatrix} 4.2000 & -5.4705 & -2.0083 \\ -0.9798 & 0.8000 & 0.8944 \\ 0 & -0.0000 & -0.0000 \end{pmatrix}$$

$$Q_1 = \begin{pmatrix} -0.9739 & -0.2272 & -0.0000 \\ 0.2272 & -0.9739 & -0.0000 \\ 0 & -0.0000 & 1.0000 \end{pmatrix}$$

$$R_1 = \begin{pmatrix} -4.3128 & 5.5092 & 2.1590 \\ 0 & 0.4637 & -0.4148 \\ 0 & -0.0000 & -0.0000 \end{pmatrix}$$

$k = 2$:

$$A_2 = R_1 Q_1 = \begin{pmatrix} 5.4516 & -4.3854 & 2.1590 \\ 0.1054 & -0.4516 & -0.4148 \\ 0 & 0 & 0 \end{pmatrix}$$

$$Q_2 = \begin{pmatrix} -0.9998 & 0.0193 & -0.0000 \\ -0.0193 & -0.9998 & 0.0000 \\ 0 & 0 & 1.0000 \end{pmatrix}$$

$$R_2 = \begin{pmatrix} -5.4526 & 4.3933 & -2.1506 \\ 0 & 0.3668 & 0.4564 \\ 0 & 0 & -0.0000 \end{pmatrix}$$

Note that each Q_i and A_i are upper Hessenberg. Also, the eigenvalues of A_2 are 5.3723, 0, −0.3722, and those of A are 5.3723, 0, and −0.3723.

MATCOM Note: Hessenberg-QR iteration has been implemented in MATCOM program QRITRH.

8.9.3 Convergence of the QR Iterations and the Shift of Origin

Once it is known that the QR iteration method can be made efficient by initial reduction of A to a Hessenberg matrix, it is natural to wonder about the rate of convergence of the iterations. By rate of convergence we mean how fast the subdiagonal entries of the transformed Hessenberg matrix converge to zero.

Let A be initially reduced to an upper Hessenberg matrix H; the QR iteration is performed on H to obtain the sequence H_k. Then it can be shown (Golub and Van Loan 1989, 374) that the subdiagonal entry $h_{i,i-1}^{(k)}$ of H_k converges to zero at a rate determined by the ratio

$$\left|\frac{\lambda_i}{\lambda_{i-1}}\right|^k$$

Thus, *the rate of convergence will be very slow if the moduli of two eigenvalues λ_i and λ_{i-1} are very close to each other.* Fortunately, the rate of convergence can significantly be improved by shifting the origin.

Let $\hat{\lambda}_i$ be an approximation of an eigenvalue λ_i of H. Let the QR iteration be applied to the matrix

$$\hat{H} = H - \hat{\lambda}_i I$$

The eigenvalues of $\hat{H}$ are $\lambda_1 - \hat{\lambda}_i, \lambda_2 - \hat{\lambda}_i, \ldots, \lambda_n - \hat{\lambda}_i$. Let these eigenvalues be ordered so that

$$|\lambda_1 - \hat{\lambda}_i| > |\lambda_2 - \hat{\lambda}_i| > \cdots > |\lambda_n - \hat{\lambda}_i|$$

Then, in this case, the $(i-1)$th subdiagonal entry of $\hat{H}_k$ will converge to zero at a rate determined by the ratio

$$\left|\frac{\lambda_i - \hat{\lambda}_i}{\lambda_{i-1} - \hat{\lambda}_i}\right|^k$$

rather than by the ratio $|\lambda_i/\lambda_{i-1}|^k$. The former is usually smaller than the latter.

Consider the convincing example from Ortega and Poole (1981, 227).

Let $\lambda_i = 0.99$, $\lambda_{i-1} = 1.1$, and $\hat{\lambda}_i = 1$. Then

$$\left|\frac{\lambda_i - \hat{\lambda}_i}{\lambda_{i-1} - \hat{\lambda}_i}\right| = 0.1$$

and

$$\left|\frac{\lambda_i}{\lambda_{i-1}}\right| = 0.9$$

This observation tells us that if we apply the QR iteration to the shifted matrix $\hat{H}$ rather than to the original matrix H, then the rate of convergence will be faster. Of course, once an eigenvalue of $\hat{H}$ is found, the corresponding eigenvalue of H can be computed just by adding in the shift.

8.9.4 The Single-Shift QR Iteration

To implement the single-shift QR iteration, we need to have an approximate $\hat{\lambda}_i$ of the eigenvlaue λ_i. Since $\hat{\lambda}_i$ is not available in practice, $h_{nn}^{(0)}$, the (n, n)th entry of $H = H_0$ is taken as an initial approximation, and then the iteration is continued using the (n, n)th entry of the current matrix as the new shift. The preceding argument suggests the following modified QR iteration method, known as the **single-shift QR iteration method**.

Step 1: Transform A to an upper Hessenberg matrix H.

Step 2: Set $H_0 = H$.

For $k = 0, 1, 2, \ldots$ do until $h_{n,n-1}^{(k)}$ becomes computationally zero.

$$H_k - h_{nn}^{(k)} I = Q_k R_k$$
$$H_{k+1} = R_k Q_k + h_{nn}^{(k)} I$$

In the preceding, $h_{ij}^{(k)}$ denotes the (i, j)th entry of H_k. Of course, each of the matrices $\{H_k\}$ can overwrite H.

Convergence. The rate of convergence of the single-shift QR iteration method is ultimately quadratic (see Stewart, 1973, 366–7).

Remark: An *implicit* version of the single shift iteration known as the **implicit QR iteration** can be worked out, in which we do not subtract the shifts but implicitly construct the matrix H_{k+1}. This is explained in Exercise 29.

Example 8.9.4 Single-shift QR Iteration

$$H = H_0 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \\ 0 & 1 & 1 \end{pmatrix}$$

$k = 0$:

$$H_0 - h_{33}^{(0)} I = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 3 \\ 0 & 1 & 0 \end{pmatrix} = Q_0 R_0$$

$$Q_0 = \begin{pmatrix} 0 & 0.7071 & -0.7071 \\ -1 & 0 & 0 \\ 0 & 0.7071 & 0.7071 \end{pmatrix}$$

$$R_0 = \begin{pmatrix} -1.0000 & -1.0000 & -3.000 \\ 0 & 1.4142 & 0.7071 \\ 0 & 0 & -0.7071 \end{pmatrix}$$

$$H_1 = R_0 Q_0 + h_{33}^{(0)} I = \begin{pmatrix} 2.000 & -2.8284 & -1.4142 \\ -1.4142 & 1.5000 & 0.5000 \\ 0 & -0.5000 & 0.5000 \end{pmatrix}$$

$k = 1$:

$$H_1 - h_{33}^{(1)} I = Q_1 R_1$$

$$Q_1 = \begin{pmatrix} -0.7276 & -0.6342 & -0.2615 \\ 0.6860 & -0.6727 & -0.2774 \\ 0 & -0.3812 & 0.9245 \end{pmatrix}$$

$$R_1 = \begin{pmatrix} -2.0616 & 2.7440 & 1.3720 \\ 0 & 1.3117 & 0.5606 \\ 0 & 0 & 0.2311 \end{pmatrix}$$

$$H_2 = R_1 Q_1 + h_{33}^{(1)} I = \begin{pmatrix} 3.8824 & -1.0613 & 1.0464 \\ 0.8998 & -0.5960 & 0.1544 \\ 0 & -0.0881 & 0.7137 \end{pmatrix}$$

$k = 2$:

$$H_2 - h_{33}^{(2)} I = Q_2 R_2$$

$$Q_2 = \begin{pmatrix} -0.9620 & 0.2721 & 0.0247 \\ -0.2732 & -0.9580 & -0.0870 \\ 0 & -0.0905 & 0.9959 \end{pmatrix}$$

$$R_2 = \begin{pmatrix} -3.2940 & 1.3787 & -1.0488 \\ 0 & 0.9740 & 0.1367 \\ 0 & 0 & 0.0124 \end{pmatrix}$$

$$H_3 = R_2 Q_2 + h_{33}^{(2)} I = \begin{pmatrix} 3.5057 & -2.1221 & -1.2459 \\ -0.2661 & -0.2318 & 0.0514 \\ 0 & -0.0011 & 0.7260 \end{pmatrix}$$

$k = 3$:

$$H_3 - h_{33}^{(3)} I = Q_3 R_3$$

$$Q_3 = \begin{pmatrix} -0.9955 & -0.0953 & -0.0001 \\ 0.0953 & -0.9955 & -0.0010 \\ 0 & -0.0010 & 1.000 \end{pmatrix}$$

$$R_3 = \begin{pmatrix} -2.7924 & 2.0212 & 1.2451 \\ 0 & 1.1556 & 0.0675 \\ 0 & 0 & 0.0001 \end{pmatrix}$$

$$H_4 = R_3 Q_3 + h_{33}^{(3)} I = \begin{pmatrix} 3.6983 & -1.7472 & 1.2434 \\ 0.1101 & -0.4244 & 0.0664 \\ 0 & 0.0000 & 0.7261 \end{pmatrix}$$

The iteration is clearly converging toward the eigenvalue 0.7261. (The eigenvalues of H, in four-digit arithmetic, are 0.7261, 3.6511, and −0.3772.)

MATCOM Note: Explicit single-shift QR iteration has been implemented in MATCOM program QRITRSSE.

8.9.5 The Double-Shift QR Iteration

If the eigenvalues are real, the preceding single-shift QR iteration works well. However, a real matrix can have complex eigenvalues. In such a case,

1. At some stage of iteration, we could encounter a trailing 2×2 submatrix on the bottom right-hand corner corresponding to a complex conjugate pair of eigenvalues, if there are any.
2. The (n, n)th entry of the trailing 2×2 matrix, which is real, will not be a good approximation.
3. It is natural to use the eigenvalues of that 2×2 matrix as shift parameters, yielding the **double-shift QR iteration**.

One Iteration Step of the Double-Shift QR (Complex)

Let the eigenvalues of the 2×2 bottom right-hand corner of the Hessenberg matrix H_s be k_1 and $k_2 = \overline{k}_1$. Then one iteration step of the double-shift QR iteration is

$$\begin{aligned} H_s - k_1 I &= Q_s R_s, & H_{s+1} &= R_s Q_s + k_1 I \\ H_{s+1} - k_2 I &= Q_{s+1} R_{s+1}, & H_{s+2} &= R_{s+1} Q_{s+1} + k_2 I \end{aligned}$$

Example 8.9.5

$$H_0 = H = \begin{pmatrix} 1 & 2 & 2 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{pmatrix}$$

$$k_1 = i, \qquad k_2 = -i$$

$H_0 - k_1 I = Q_0 R_0$:

$$Q_0 = \begin{pmatrix} -1 & 0 & 0 \\ 0 & -0.7071 & -0.7071i \\ 0 & 0.7071i & 0.7071 \end{pmatrix}$$

$$R_0 = \begin{pmatrix} -1+i & -2 & -2 \\ 0 & 1.4142i & -1.4142 \\ 0 & 0 & 0 \end{pmatrix}$$

$$H_1 = R_0 Q_0 + k_1 I = \begin{pmatrix} 1 & 1.4142 - 1.4142i & -1.4142 + 1.4142i \\ 0 & -i & 0 \\ 0 & 0 & i \end{pmatrix}$$

$H_1 - k_2 I = Q_1 R_1$:

$$Q_1 = \begin{pmatrix} -1 & 0 & 0 \\ 0 & -0.8165 & -0.5744i \\ 0 & 0.5744i & 0.8165 \end{pmatrix}$$

$$R_1 = \begin{pmatrix} -1-i & -1.4142 + 1.4142i & 1.4142 - 1.4142i \\ 0 & 0 & 1.1547 \\ 0 & 0 & 1.6330i \end{pmatrix}$$

$$H_2 = R_1 Q_1 + k_2 I = \begin{pmatrix} 1 & 1.9712 - 0.3382i & 1.9712 - 0.3382i \\ 0 & -0.3333i & 0.9428 \\ 0 & -0.9428 & 0.3333i \end{pmatrix}$$

Note that the eigenvalues of the 2×2 bottom right-hand corner matrix

$$\begin{pmatrix} -0.3333i & 0.9428 \\ -0.9428 & 0.3333i \end{pmatrix}$$

are $-i$ and i.

Thus, the eigenvalues of H_2 are 1, i, and $-i$.

Avoiding Complex Arithmetic in Double-Shift QR Iteration

Because k_1 and k_2 can be complex, the preceding double-shift QR iteration step may require complex arithmetic for implementation, even though the starting matrix H_s is real. However, with a little manipulation complex arithmetic can be avoided. We discuss this aspect now.

We show that the matrix H_{s+2} is orthogonally similar to H_s through a real transforming matrix and can be formed directly from H_s without computing H_{s+1}.

Consider the matrix

$$N = (H_s - k_2 I)(H_s - k_1 I) = H_s^2 - (k_1 + k_2)H_s + k_1 k_2 I$$

Because $k_2 = \bar{k}_1$, the matrix N is real. Next, we show that $(Q_s Q_{s+1})(R_{s+1} R_s)$ is the QR factorization of N.

$$\begin{aligned} N &= (H_s - k_2 I)(H_s - k_1 I) \\ &= (H_s - k_2 I) Q_s R_s \\ &= Q_s Q_s^* (H_s - k_2 I) Q_s R_s \\ &= Q_s (H_{s+1} - k_2 I) R_s = Q_s Q_{s+1} R_{s+1} R_s \end{aligned}$$

Because N is real and $(Q_s Q_{s+1})(R_{s+1} R_s)$ is the QR factorization of N, the matrix $Q_s Q_{s+1}$ can be chosen to be real and orthogonal.

Finally, we show that H_{s+2} is orthogonally similar to H_s through this real transforming matrix $Q_s Q_{s+1}$.

$$\begin{aligned} H_{s+2} &= R_{s+1} Q_{s+1} + k_2 I \\ &= Q_{s+1}^* (H_{s+1} - k_2 I) Q_{s+1} + k_2 I \\ &= Q_{s+1}^* (R_s Q_s + (k_1 - k_2) I) Q_{s+1} + k_2 I \\ &= Q_{s+1}^* [Q_s^* (H_s - k_1 I) Q_s + (k_1 - k_2) I] Q_{s+1} + k_2 I \\ &= Q_{s+1}^* Q_s^* H_s Q_s Q_{s+1} \\ &= (Q_s Q_{s+1})^T H_s (Q_s Q_{s+1}) \end{aligned}$$

Because $Q_s Q_{s+1}$ and H_s are real, H_{s+2} is also real.

From the preceding discussions we conclude that it is possible to obtain H_{s+2} directly from H_s through real orthogonal transformations.

> Eigenvalues k_1 and k_2 need not be computed explicitly.

Although computing the eigenvalues of a 2×2 matrix is almost a trivial job, we note that k_1 and k_2 need not be computed explicitly. To form the matrix

$$N = (H_s - k_1 I)(H_s - k_2 I) = H_s^2 - (k_1 + k_2)H_s + k_1 k_2 I$$

all we need to compute is the trace and the determinant of the 2×2 matrix.

Let

$$\begin{pmatrix} h_{n-1,n-1} & h_{n-1,n} \\ h_{n,n-1} & h_{nn} \end{pmatrix}$$

be the 2×2 right-hand corner matrix of the current matrix H_s. Then

$$\begin{aligned} t &= k_1 + k_2 = \text{sum of the eigenvalues} = \text{trace} \\ &= h_{n-1,n-1} + h_{nn}, \text{ is real} \\ d &= k_1 k_2 = \text{product of the eigenvalues} = \text{determinant} \\ &= h_{n-1,n-1} h_{nn} - h_{n,n-1} h_{n-1,n}, \text{ is real} \end{aligned}$$

This allows us to write one step of double-shift QR iteration in *real arithmetic* as follows.

> **One Step of Double-Shift QR Iteration (Real Arithmetic)**
>
> 1. Form the matrix $N = H_s^2 - tH_s + dI$.
> 2. Find the QR factorization of N: $N = QR$.
> 3. Form $H_{s+2} = Q^T H_s Q$.

We call this computation **explicit double-shift** QR iteration, for reasons to be stated in the next section.

Example 8.9.6

$$H = H_0 = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 0 & 1 \\ 0 & -2 & 2 \end{pmatrix}$$

$$t = 2, \qquad d = 2$$

$$N = H^2 - tH + dI = \begin{pmatrix} 3 & -8 & 5 \\ -1 & 2 & 3 \\ -2 & 0 & 0 \end{pmatrix}$$

Find the QR factorization of N:

$$Q = \begin{pmatrix} -0.8018 & -0.5470 & -0.2408 \\ 0.2673 & 0.0322 & -0.9631 \\ 0.5345 & -0.8365 & 0.1204 \end{pmatrix}$$

$$H_2 = Q^T H_0 Q = \begin{pmatrix} -0.8571 & 1.1007 & 2.5740 \\ -1.1867 & 3.0455 & -0.8289 \\ 0.0000 & 1.8437 & 0.8116 \end{pmatrix}$$

MATCOM Note: Explicit double-shift QR iteration has been implemented in MATCOM program QRITRDSE.

8.9.6 Implicit QR Iteration

After all this, we note, with utter disappointment, that the preceding double-shift (explicit) QR iteration is not *practical*. It is not practical because forming the matrix N itself in step 1 requires $O(n^3)$ flops. Fortunately, a little trick again allows us to implement the step in $O(n^2)$ flops.

One Iteration of the Double-Shift Implicit QR

1. Compute the first column n_1 of the matrix N.
2. Find a Householder matrix P_0 such that $P_0 n_1$ is a multiple of e_1.
3. Form $H_s' = P_0^T H_s P_0$.
4. Transform H_s' to an upper Hessenberg matrix by orthogonal similarity using Householder matrices:

$$(P_{n-2}^T \cdots P_2^T P_1^T) H_s' (P_1 P_2 \cdots P_{n-2}) = H_{s+2}'$$

Using the implicit Q theorem (Theorem 5.5.3) we can show that the matrix H_{s+2} of the explicit QR and H_{s+2}' of the implicit QR are both unreduced upper Hessenberg and are essentially the same matrix.

The preceding four steps constitute one iteration of the double-shift implicit QR. To understand how these four steps can be implemented in $O(n^2)$ flops, we must look into the structures of the computations.

First, because H_s is Hessenberg, computing the first column of $N = (H_s - k_1 I)(H_s - k_2 I)$ is almost trivial. It contains only three nonzero elements. They can be written down explicitly.

Let

$$Ne_1 = n_1 = \begin{pmatrix} n_{11} \\ n_{21} \\ n_{31} \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

then

$$\begin{aligned} n_{11} &= h_{11}^2 - th_{11} + d + h_{12}h_{21} \\ n_{21} &= h_{21}(h_{11} + h_{22} - t) \\ n_{31} &= h_{21}h_{32} \end{aligned}$$

Here h_{ij} refers to the (i, j)th entry of H_s. Second, because only three elements of n_1 are nonzero, the Householder matrix P_0 has the form

$$P_0 = \begin{pmatrix} \hat{P}_0 & 0 \\ 0 & I_{n-3} \end{pmatrix}$$

where $\hat{P}_0$ is a 3×3 Householder matrix. Because of this form of P_0 and H_s being Hessenberg, the matrix $H_s' = P_0 H_s P_0$ is not a full matrix. It is a Hessenberg matrix with a bulge. For example, when $n = 6$, we have

$$H_s' = P_0 H_s P_0 = \begin{pmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ 0 & 0 & 0 & * & * & * \\ 0 & 0 & 0 & 0 & * & * \end{pmatrix}$$

A bulge will be created at each step of the reduction of H_s' to Hessenberg form, and the constructions of Householder matrices P_1 through P_{n-2} amount to chasing these bulges systematically.

Each P_k, $k = 1, 2, \ldots, n-3$ has the form

$$P_k = \begin{pmatrix} I_k & & 0 \\ & \hat{P}_k & \\ 0 & & I_{n-k-3} \end{pmatrix}$$

where $\hat{P}_k$ is a 3×3 Householder matrix. The last Householder matrix, P_{n-2}, has the form

$$P_{n-2} = \begin{pmatrix} I_{n-2} & 0 \\ 0 & \hat{P}_{n-2} \end{pmatrix}$$

Taking into consideration the preceding structures of computations, it can be shown that one step of the implicit QR iteration requires only $O(n^2)$ flops.

For details of this $O(n^2)$ computations of one iteration of the double-shifted implicit QR, see the book by Stewart (1973, 375–8) and the recent book by Watkins (1991, 277–8).

ALGORITHM 8.9.1 One Iteration Step of the Double-Shift Implicit QR Let H be an $n \times n$ unreduced upper Hessenberg matrix. Then the following algorithm, which constitutes one step of the double-shift implicit QR iteration, produces orthogonal matrices $P_0, P_1, \ldots, P_{n-2}$ such that

$$Q^T HQ, \quad \text{where} \quad Q = P_0 P_1 \ldots P_{n-2}$$

is an upper Hessenberg matrix. The algorithm overwrites H with $Q^T HQ$.

1: Compute the shifts:

$$t = h_{n-1,n-1} + h_{nn}$$
$$d = h_{n-1,n-1}h_{nn} - h_{n,n-1}h_{n-1,n}$$

2: Compute the first three nonzero entries of the first column of $N = H^2 - tH + dI$:

$$x = n_{11} = h_{11}^2 - th_{11} + d + h_{12}h_{21}$$
$$y = n_{21} = h_{21}(h_{11} + h_{12} - t)$$
$$z = h_{31} = h_{21}h_{32}$$

3: Compute the Householder matrices $P_0 P_1 \ldots P_{n-2}$ such that the final matrix is upper Hessenberg.

3(a) For $k = 0, 1, 2, \ldots, n-3$ do
Find a 3×3 Householder matrix $\hat{P}_k$ such that

$$\hat{P}_k \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} * \\ 0 \\ 0 \end{pmatrix}$$

Overwrite H with $P_k H P_k^T$, where

$$P_k = \begin{pmatrix} I_k & & 0 \\ & \hat{P}_k & \\ 0 & & I_{n-k-3} \end{pmatrix}$$

Update x, y, and z:

$$x \equiv h_{k+2,k+1}$$
$$y \equiv h_{k+3,k+1}$$
$$z \equiv h_{k+4,k+1} \quad (\text{if } k < n-3)$$

3(b) Find a Householder matrix $\hat{P}_{n-2}$ of order 2 such that

$$\hat{P}_{n-2} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} * \\ 0 \end{pmatrix}$$

Form

$$P_{n-2} = \begin{pmatrix} I_{n-2} & 0 \\ 0 & \hat{P}_{n-2} \end{pmatrix}$$

Flop count. One step of the double-shift implicit QR iteration takes about $6n^2$ flops. If the transforming matrix Q is needed and accumulated, then another $6n^2$ flops will be needed.

Example 8.9.7

1. $t = 2, d = 2, H = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 0 & 1 \\ 0 & -2 & 2 \end{pmatrix}$
2. $x = n_{11} = 3, y = n_{21} = -1, z = n_{31} = -2$
3. $k = 0$:

$$P_0 = I - \frac{2uu^T}{u^T u}, \quad \text{where} \quad u = \begin{pmatrix} 6.7417 \\ -1.0000 \\ -2.0000 \end{pmatrix}$$

$$P_0 = \begin{pmatrix} -0.8018 & 0.2673 & 0.5345 \\ 0.2673 & 0.9604 & -0.0793 \\ 0.5345 & -0.0793 & 0.8414 \end{pmatrix}$$

$$H \equiv P_0^T H P_0 = \begin{pmatrix} -0.8571 & -2.6248 & -0.9733 \\ 0.0581 & 0.8666 & 1.9505 \\ 1.1852 & -0.7221 & 2.9906 \end{pmatrix}$$

Update x and y: $x = h_{21} = 0.0581$; $y = h_{31} = 1.1852$.
Find P_1:

$$\hat{P}_1 = \begin{pmatrix} 0.0490 & -0.9988 \\ -0.9988 & 0.0490 \end{pmatrix}, \quad \hat{P}_1 \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} -1.1867 \\ 0 \end{pmatrix}$$

$$P_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0.0490 & -0.9988 \\ 0 & -0.9988 & 0.0490 \end{pmatrix}$$

$$H \equiv P_1^T H P_1 = \begin{pmatrix} -0.8571 & 1.1007 & 2.5740 \\ -1.1867 & 3.0455 & -0.8289 \\ 0 & 1.8437 & 0.8116 \end{pmatrix}$$

Note that the matrix H obtained by the implicit QR is the same as H_2 obtained earlier in Section 8.9.5 by the explicit QR (Example 8.9.6).

MATCOM Note: Algorithm 8.9.1 has been implemented in MATCOM program QRITRDSI.

8.9.7 Obtaining the Real Schur Form

1. Transform the matrix A to Hessenberg form (Algorithm 5.4.4 or Algorithm 5.5.5).
2. Iterate with the implicit double-step QR method (Algorithm 8.9.1).

Typically, after two to three steps of the double-shift implicit QR iteration, one or two (and sometimes more) subdiagonal entries of the Hessenberg matrix converge

to zero. This result will give us a real eigenvalue or a pair of complex conjugate eigenvalues. Once a real or a pair of complex conjugate eigenvalues is computed, the last row and the last column in the first case or the last two rows and the last two columns in the second case can be deleted, and computation of the other eigenvalues can be continued with the submatrix. This process is also known as *deflation*.

Note that the eigenvalues of the deflated submatrix are also the eigenvalues of the original matrix. Suppose that, immediately before deflation, the matrix has the form

$$H_k = \begin{pmatrix} A' & C' \\ 0 & B' \end{pmatrix}$$

where B' is the 2×2 trailing submatrix or is a 1×1 matrix. Then the characteristic equation of H_k is

$$\det(\lambda I - H_k) = \det(\lambda I - A')\det(\lambda I - B')$$

Thus, the eigenvalues of H_k are the eigenvalues of A' together with those of B'. But H_k is orthogonally similar to the original matrix A and therefore has the same eigenvalues as A. For a precise algorithmic statement of finding the RSF, see Golub and Van Loan (1989, 380).

When to Accept a Subdiagonal Entry as Zero

A major decision that we have to make during the iteration procedure is when to accept a subdiagonal entry as zero so that the matrix can be deflated. It seems that there are no clear-cut conventions here; however, we just give a commonly used criterion below. For a good discussion on this matter, see the book *The Symmetric Eigenvalue Problem* (Parlett 1980b).

> Accept a subdiagonal entry $h_{i,i-1}$ to be zero if
>
> $$|h_{i,i-1}| \le \epsilon(|h_{ii}| + |h_{i-1,i-1}|),$$
>
> where ϵ is the tolerance.

Example 8.9.8 Find the real Schur form (RSF) of

$$H = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 0 & 1 \\ 0 & -2 & 2 \end{pmatrix}$$

Iteration	h_{21}
1	−1.1867
2	0.3543
3	0.0129
4	0.0000

The computed RSF is

$$\begin{pmatrix} -1.1663 & -1.3326 & -2.0531 \\ 0 & 1.2384 & 1.6659 \\ 0 & -1.9409 & 2.9279 \end{pmatrix}$$

The eigenvalues of the 2×2 right-hand lower corner submatrix are $2.0832 \pm 1.5874i$.

Example 8.9.9 Find the real Schur form of

$$H = \begin{pmatrix} 0.2190 & -0.0756 & 0.6787 & -0.6391 \\ -0.9615 & 0.9032 & -0.4571 & 0.8804 \\ 0 & -0.3822 & 0.4526 & -0.0641 \\ 0 & 0 & -0.1069 & -0.0252 \end{pmatrix}$$

Iteration	h_{21}	h_{32}	h_{43}
1	0.3860	−0.5084	−0.0064
2	−0.0672	−0.3773	0.0001
3	0.0089	−0.3673	0
4	−0.0011	−0.3590	0
5	0.0001	−0.3905	0

The computed RSF is

$$H = \begin{pmatrix} 1.4095 & 0.7632 & -0.1996 & 0.8394 \\ 0.0001 & 0.1922 & 0.5792 & 0.0494 \\ 0 & -0.3905 & 0.0243 & -0.4089 \\ 0 & 0 & 0 & -0.0763 \end{pmatrix}$$

The eigenvalues of $\begin{pmatrix} 0.1922 & 0.5792 \\ -0.3905 & 0.0243 \end{pmatrix}$ are $0.1082 \pm 0.4681i$.

Balancing

As in the process of solving linear system problems, it is advisable to balance the entries of the original matrix A, if they vary widely, before starting the QR iterations.

Balancing amounts to preprocessing the matrix A so that its eigenproblem becomes easier. There are two types: the first is **permutation**. It amounts to finding a permutation matrix P so that the matrix $A' = PAP^T$, where P is a permutation matrix, is nearly upper triangular or closer to real Schur form. The second is **scaling**. It is equivalent to finding a diagonal matrix D so that the matrix $DA'D^{-1}$ has rows and columns of more or less equal norms. These two operations are referred to as **balancing**. LAPACK routine SGEBAL performs balancing before eigenvalues are computed.

In general, preprocessing the matrix by balancing improves the accuracy of the QR iteration method. Note that no roundoff error is involved in this computation, and it takes only $O(n^2)$ flops.

Flop count. Because the QR iteration method is an iterative method, it is hard to give an exact flop count. However, empirical observations have established that it takes about two QR iterations per eigenvalue. Thus, it will require about $6n^3$ flops to compute all the eigenvalues, taking deflation into account. If the transforming matrix Q and the final quasi-triangular matrix T are also needed, then the cost will be about $15n^3$ flops. These counts are very approximate.

Roundoff property. The QR iteration method is quite stable. An analysis of the roundoff property of the algorithm shows that the computed RSF is orthogonally similar to a nearby matrix $A + E$, where

$$\|E\|_F \leq \phi(n)\mu\|A\|_F$$

$\phi(n)$ is a slowly growing function of n. The computed matrix Q is also nearly orthogonal.

8.9.8 The Real Schur Form and Invariant Subspaces

DEFINITION 8.9.2 Let $\mathbb{S}$ be a subspace of the complex plane $\mathbb{C}^n$. Then $\mathbb{S}$ is called an **invariant subspace** (with respect to premultiplication by A) if $x \in \mathbb{S}$ implies that $Ax \in \mathbb{S}$.

Thus, since

$$Ax = \lambda x$$

for each eigenvalue λ, each eigenvector is an invariant subspace of dimension 1 associated with the corresponding eigenvalue.

The real Schur form of A displays information on the invariant subspaces.

Basis of an Invariant Subspace from RSF

Let

$$Q^T AQ = R = \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix}$$

and let us assume that R_{11} and R_{22} do not have eigenvalues in common. Then the first p columns of Q, where p is the order of R_{11}, forms a basis for the invariant subspace associated with the eigenvalues of R_{11}.

In many applications, such as in the solution of algebraic Riccati equations (Laub 1979), we need to compute the orthonormal basis of an invariant subspace associated with a selected number of eigenvalues. Unfortunately, the transformed real Schur form obtained by QR iteration may not give the eigenvalues in some desired order. Thus, if the eigenvalues are not in a desired order, we might wonder if some extra work can be done to bring them into that order. That this can indeed be done is seen from the following simple discussion. Let A be 2×2.

Let

$$Q_1^T AQ_1 = \begin{pmatrix} \lambda_1 & r_{12} \\ 0 & \lambda_2 \end{pmatrix}, \qquad \lambda_1 \neq \lambda_2$$

If λ_1 and λ_2 are not in right order, all we need to do to reverse the order is to form a Givens rotation $J(1, 2, \theta)$ such that

$$J(1, 2, \theta)\begin{pmatrix} r_{12} \\ \lambda_2 - \lambda_1 \end{pmatrix} = \begin{pmatrix} * \\ 0 \end{pmatrix}$$

Then $Q = Q_1 J(1, 2, \theta)^T$ is such that

$$Q^T AQ = \begin{pmatrix} \lambda_2 & \pm r_{12} \\ 0 & \lambda_1 \end{pmatrix}$$

Example 8.9.10

$$A = \begin{pmatrix} 1 & 2 \\ 2 & 3 \end{pmatrix}$$

$$Q_1 = \begin{pmatrix} 0.8507 & 0.5257 \\ -0.5257 & 0.8507 \end{pmatrix}$$

$$Q_1^T AQ_1 = \begin{pmatrix} -0.2361 & 0.0000 \\ 0.0000 & 4.2361 \end{pmatrix}$$

$$J(1, 2, \theta) = \begin{pmatrix} 0 & -1 \\ -1 & 0 \end{pmatrix}$$

$$J(1, 2, \theta)\begin{pmatrix} 0 \\ -4.4722 \end{pmatrix} = \begin{pmatrix} 4.4722 \\ 0 \end{pmatrix}$$

$$Q = Q_1 J(1, 2, \theta)^T = \begin{pmatrix} -0.5257 & -0.8507 \\ -0.8507 & 0.5257 \end{pmatrix}$$

$$Q^T AQ = \begin{pmatrix} 4.2361 & 0.00 \\ 0.00 & -0.2361 \end{pmatrix}$$

This simple process can easily be extended to achieve any desired ordering of the eigenvalues in the real Schur form. For details see Golub and Van Loan (1989, 386).

The process is quite inexpensive. It requires only $k(8n)$ flops, where k is the number of interchanges required to achieve the desired order. Stewart (1976b) has provided useful Fortran routines for such an ordering of the eigenvalues.

8.10 Computing the Eigenvectors

8.10.1 The Hessenberg Inverse Iteration

As soon as an eigenvalue λ is computed by QR iteration, we can invoke inverse iteration (Algorithm 8.5.2) to compute the corresponding eigenvector. However, because A is initially reduced to a Hessenberg matrix H for the QR iteration, it is natural

to take advantage of the structure of the Hessenberg matrix H in the solutions of the linear system that need to be solved in the process of inverse iteration. Thus the Hessenberg inverse iteration can be stated as follows:

ALGORITHM 8.10.1 The Hessenberg Inverse Iteration

Step 1: Reduce the matrix A to an upper Hessenberg matrix H:

$$P^T AP = H$$

Step 2: Compute an eigenvalue λ, whose eigenvector x is sought, using the implicit QR algorithm.

Step 3: Apply the inverse iteration

For $k = 1, 2, \ldots$, do until convergence
Solve $(H - \lambda I)z^{(k)} = y^{(k-1)}$
$y^{(k)} = z^{(k)} / \max(z^{(k)})$
Stop if $\|(H - \lambda I)y^{(k)}\| \le c\mu\|A\|_\infty$, where c is a constant of order unity.

Step 4: Recover the eigenvector x:

$$x = Py^{(k)}$$

where $y^{(k)}$ is the approximation of the eigenvector y obtained from step 3.

8.10.2 Calculating the Eigenvectors from the Real Schur Form

The eigenvectors can also be calculated directly from the real Schur form without invoking the inverse iteration. The process is described as follows.

Let A be transformed to the RSF T by the implicit QR iteration:

$$Q^T AQ = T$$

Then $Ax = \lambda x$ can be written as

$$Q^T AQQ^T x = \lambda Q^T x$$

That is, writing $Q^T x = y$, we have

$$Ty = \lambda y$$

Thus, after A has been transformed to the RSF T, an eigenvector x corresponding to an eigenvalue λ can be computed as follows:

1. Solve the homogeneous triangular or the quasi-triangular system $Ty = \lambda y$.
2. Compute $x = Qy$.

We now show how the solution of $Ty = \lambda y$ can be simplified assuming that T is triangular and that all the eigenvalues of A (that is, the diagonal entries of T) are distinct.

Let $\lambda = t_{kk}$; that is, we are trying to find a y such that

$$(T - t_{kk}I)y = 0$$

Write

$$T - t_{kk}I = \begin{pmatrix} T_{11} & T_{12} \\ 0 & T_{22} \end{pmatrix}$$

where T_{11} is $k \times k$. Partition y accordingly:

$$y = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$$

where y_1 has k entries. Then

$$(T - t_{kk}I)y = 0$$

gives

$$\begin{pmatrix} T_{11} & T_{12} \\ 0 & T_{22} \end{pmatrix}\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = 0$$

that is,

$$T_{11}y_1 + T_{12}y_2 = 0, \qquad T_{22}y_2 = 0$$

Now, T_{22} is nonsingular, because its diagonal entries are $t_{jj} - t_{kk}$, $j = k + 1, \ldots, n$, which are different from zero. So, the homogeneous system

$$T_{22}y_2 = 0$$

has only the trivial solution $y_2 = 0$. From before, we have

$$T_{11}y_1 = 0$$

Because the kth diagonal entry of T_{11} is zero, T_{11} is singular; therefore, $y_1 \neq 0$. Again, note that T_{11} has the form

$$T_{11} = \left(\begin{array}{c|c} \hat{T}_{11} & s \\ \hline 0 & 0 \end{array}\right)$$

where $\hat{T}_{11}$ is $(k - 1) \times (k - 1)$ and nonsingular. Thus, $T_{11}y_1 = 0$ reduces to

$$\hat{T}_{11}\hat{y}_1 + sz = 0$$

where

$$y_1 = \begin{pmatrix} \hat{y}_1 \\ z \end{pmatrix}$$

z can be chosen to be any nonzero number. Because $\hat{T}_{11}$ is upper triangular, $\hat{y}_1$ can be computed by back substitution.

ALGORITHM 8.10.2 Computing an Eigenvector Directly from the RSF

Step 1: Transform A to RSF by the implicit QR iteration

$$Q^T AQ = T$$

(Assume that T is triangular and that the diagonal entries of T are different.)

Step 2: Select the eigenvalue $\lambda = t_{kk}$ whose eigenvector x is to be computed.

Step 3: Partition

$$(T - t_{kk}I) = \begin{pmatrix} T_{11} & T_{12} \\ 0 & T_{22} \end{pmatrix}$$

Step 4: Partition

$$T_{11} = \begin{pmatrix} \hat{T}_{11} & s \\ 0 & 0 \end{pmatrix}$$

Step 5: Solve by back substitution:

$$\hat{T}_{11}\hat{y}_1 = -sz$$

choosing z as a nonzero number.

Step 6: Form $y_1 = \begin{pmatrix} \hat{y}_1 \\ z \end{pmatrix}$

Step 7: Form $y = \begin{pmatrix} y_1 \\ 0 \end{pmatrix}$

Step 8: Compute $x = Qy$

Example 8.10.1

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 2 & 3 & 4 \\ 4 & 6 & 7 \end{pmatrix}$$

$$T = \begin{pmatrix} 10.8998 & 2.5145 & 2.7563 \\ 0 & 0.3571 & 0.2829 \\ 0 & 0 & -0.2569 \end{pmatrix}$$

$$Q = \begin{pmatrix} -0.1358 & -0.9606 & -0.2423 \\ -0.4747 & 0.2778 & -0.8352 \\ -0.8696 & -0.0016 & 0.4937 \end{pmatrix}$$

Suppose we want to compute the eigenvector corresponding to $\lambda = t_{22} = .3571$. Then

$$T_{11} = \begin{pmatrix} 10.5428 & 2.5145 \\ 0 & 0 \end{pmatrix}$$

$$\hat{T}_{11} = 10.5428$$

$$s = 2.5145$$

Choose $z = 1$.

$$\hat{y}_1 = -(\hat{T}_{11})^{-1}sz = -0.2385$$

$$y_1 = \begin{pmatrix} \hat{y}_1 \\ z \end{pmatrix} = \begin{pmatrix} -0.2385 \\ 1 \end{pmatrix}$$

Choose $y_2 = 0$. Then

$$y = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} -0.2385 \\ 1 \\ 0 \end{pmatrix}$$

$$x = Qy = \begin{pmatrix} -0.9283 \\ 0.3910 \\ 0.2058 \end{pmatrix}$$

It is easily verified that

$$Ax = \begin{pmatrix} -0.3315 \\ 0.1396 \\ 0.0735 \end{pmatrix}$$

and

$$\lambda_2 x = \begin{pmatrix} -0.3315 \\ 0.1396 \\ 0.0735 \end{pmatrix}$$

8.11 The Symmetric Eigenvalue Problem

The QR iteration can, of course, be applied to find the eigenvalues of a symmetric matrix. Indeed, in the symmetric case the method simplifies to a large extent. We discuss the symmetric QR iteration briefly. However, because the eigenvalues and eigenvectors of a symmetric matrix enjoy certain special remarkable properties over those of the nonsymmetric case, some special methods exploiting these properties can be developed for the symmetric problem. One such method is based on the Sturm sequence property of the characteristic polynomial of the matrix.

Some Special Properties

Eigenvalues of a real symmetric matrix. The eigenvalues of a real symmetric matrix are real. The eigenvectors can be chosen to be orthogonal (Theorem 8.2.8).

THEOREM 8.11.1 Real Schur Form of a Real Symmetric Matrix The real Schur form of a real symmetric matrix is a diagonal matrix.

Proof: From the real Schur triangularization theorem (Theorem 8.9.2) we have

$$Q^T AQ = R$$

where R is in real Schur form; that is, R is a triangular matrix with each diagonal entry as either a scalar or a 2×2 matrix. Each 2×2 matrix on the diagonal corresponds to a pair of complex conjugate eigenvalues. Because a real symmetric matrix cannot have a complex eigenvalue, it follows that R cannot have a 2×2 matrix on the diagonal; therefore, R is a diagonal matrix. ■

THEOREM 8.11.2 General Perturbation Property of the Symmetric Eigenvalue Problem Let A be an $n \times n$ real symmetric matrix. Let $A' = A + E$, where E is a real symmetric

perturbation of the matrix A, and let $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$ and $\lambda_1' \geq \lambda_2' \geq \cdots \geq \lambda_n'$ be the eigenvalues of A and A', respectively. Then

$$\lambda_i - \|E\|_2 \leq \lambda_i' \leq \lambda_i + \|E\|_2, \quad i = 1, 2, \ldots, n$$

We will give a proof of Theorem 8.11.2 using the following well-known theorem.

THEOREM 8.11.3 Courant–Fischer Minimax Theorem Let $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$ be the eigenvalues of a symmetric matrix A. Then

$$\lambda_i = \min_S \max_{0 \neq x \in S} \left(\frac{x^T A x}{x^T x} \right)$$

where the minimum is taken over all subspaces S of dimensions $(n - i + 1)$ and the maximum is taken over all nonzero vectors x in the subspace S.

Proof: See Saad (1992, 30–31). ■

Characterizations of the Extreme Eigenvalues

It is interesting to note that two particular cases of Theorem 8.11.3 yield the characterizations of the extreme eigenvalues λ_1 and λ_n of A.

$i = 1$. In this case minimum is to be taken over the only subspace S of dimension n, which is $\mathbb{R}^n$ itself. Thus,

$$\lambda_1 = \lambda_{\max} = \max_{x \neq 0} \frac{x^T A x}{x^T x}$$

$i = n$. In this case minimum is to be taken over the subspace S of dimension 1 consisting only of the vectors x and its multiples αx. Thus,

$$\lambda_n = \lambda_{\min} = \min_{x \neq 0} \frac{x^T A x}{x^T x}$$

Proof of Theorem 8.11.2: Applying Theorem 8.11.3 to A', we have

$$\begin{aligned}
\lambda_i' = \min \max \frac{x^T A' x}{x^T x} &= \min \left(\max \left(\frac{x^T A x}{x^T x} + \frac{x^T E x}{x^T x} \right) \right) \\
&\leq \min \left(\max \frac{x^T A x}{x^T x} + \max \frac{x^T E x}{x^T x} \right) \\
&= \min \max \left(\frac{x^T A x}{x^T x} \right) + \min \max \left(\frac{x^T E x}{x^T x} \right) \\
&\leq \min \max \frac{x^T A x}{x^T x} + \|E\|_2 \\
&= \lambda_i + \|E\|_2
\end{aligned}$$

(Note that in the preceding proof, we have used the result that

$$\frac{x^T E x}{x^T x} \leq \text{the largest eigenvalue of } E$$

and the absolute value of the largest eigenvalue of $E \leq \|E\|_2$ (Theorem 8.4.3).

Analogously, we can prove that

$$\lambda_i - \|E\|_2 \leq \lambda_i'$$ ■

Perturbation Result on the Eigenvalues of a Symmetric Matrix

The eigenvalues of a real symmetric matrix are well conditioned; that is, small changes in the elements of A cause only small changes in the eigenvalues of A.

Example 8.11.1

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 6 \end{pmatrix}, \qquad E = 10^{-4} \times I_{3\times 3}$$

The eigenvalues of A are -0.4203, 0.2336, and 10.1867. The eigenvalues of $A + E$ are -0.4202, 0.2337, and 10.1868. Note that $\|E\|_2 = 10^{-4}$.

Rank-one perturbation property. We state a theorem that shows how the eigenvalues are shifted if E is a rank-one perturbation. The result plays an important role in the divide-and-conquer algorithm (Dongarra and Sorensen 1987, and Cuppen 1981) for the symmetric eigenvalue problem.

THEOREM 8.11.4 Eigenvalues of a Rank-One Perturbed Matrix Suppose $B = A + \alpha b b^T$, where A is an $n \times n$ symmetric matrix, α is a scalar, and b is an n-vector. Let $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$ be the eigenvalues of A and $\lambda_1' \geq \cdots \geq \lambda_n'$ be the eigenvalues of B. Then

$$\lambda_i' \in [\lambda_i, \lambda_{i-1}], \qquad i = 2, \ldots, n, \quad \text{if} \quad \alpha \geq 0$$
$$\lambda_i' \in [\lambda_{i+1}, \lambda_i], \qquad i = 1, \ldots, n-1, \quad \text{if} \quad \alpha < 0$$

Proof: See Wilkinson (1965, 97–98). ■

Example 8.11.2

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 3 & 5 & 6 \end{pmatrix}, \qquad \alpha = -1, \ b = (1, 2, 3)^T$$

The eigenvalues of B are $\lambda_3' = -3.3028$, $\lambda_2' = 0$, and $\lambda_1' = 0.3028$. The eigenvalues of A are $\lambda_3 = -0.5157$, $\lambda_2 = 0.1709$, and $\lambda_1 = 11.3448$. It is easily verified that $\lambda_2 < \lambda_1' < \lambda_1$, and $\lambda_3 < \lambda_2' < \lambda_2$.

Hoffman–Wielandt theorem. We have so far assumed that both A and $A + E$ are symmetric. The next theorem considers a more general case, where A and $A + E$ are both normal.

THEOREM 8.11.5 Hoffman–Wielandt Theorem Let A and $A + E$ be both normal. Let the eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$ of A be given in some given order, and also let the eigenvalues $\lambda_1', \lambda_2', \ldots, \lambda_n'$ of $A = A + E$ be in some order. Then there is a permutation $\sigma(i)$ of the first n natural numbers $1, 2, \ldots, n$, such that

$$\left[\sum_{i=1}^{n} |\lambda_{\sigma(i)}' - \lambda_i|^2\right]^{1/2} \leq \|E\|_F$$

For proof and some interesting corollaries of this theorem, see Horn and Johnson (1985, 368–371).

8.11.1 The Sturm Sequence and the Bisection Method

In this section we describe a method for finding the eigenvalues of a symmetric matrix. The method is particularly useful if eigenvalues are required in an interval. In principle, it can be used to find all eigenvalues. *However, in practice, if all the eigenvalues are required explicitly, the symmetric QR iteration method is preferred over the method of this section for matrices of moderate sizes.*

First, the symmetric matrix A is transformed to a symmetric tridiagonal matrix T using Householder's method, described earlier:

$$PAP^T = T = \begin{pmatrix} \alpha_1 & \beta_1 & & & & \\ \beta_1 & \alpha_2 & \beta_2 & & 0 & \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & 0 & & \beta_{n-2} & \alpha_{n-1} & \beta_{n-1} \\ & & & & \beta_{n-1} & \alpha_n \end{pmatrix}$$

Let $p_i(\lambda)$ denote the characteristic polynomial of the $i \times i$ principal submatrix of T. Then these polynomials satisfy a three-term recursion:

$$p_i(\lambda) = (\alpha_i - \lambda)p_{i-1}(\lambda) - \beta_{i-1}^2 p_{i-2}(\lambda), \qquad i = 2, 3, \ldots, n$$

with

$$p_0(\lambda) = 1 \quad \text{and} \quad p_1(\lambda) = \alpha_1 - \lambda$$

Without loss of generality, we may assume that $\beta_i \neq 0$, $i = 1, 2, \ldots, n-1$. Recall that the matrix T with this property is called *unreduced*. If a subdiagonal entry of T is zero, then T is a block diagonal and its eigenproblem can thus be reduced to that of its submatrices. For unreduced symmetric tridiagonal matrix T, the following *interlacing property* is very important.

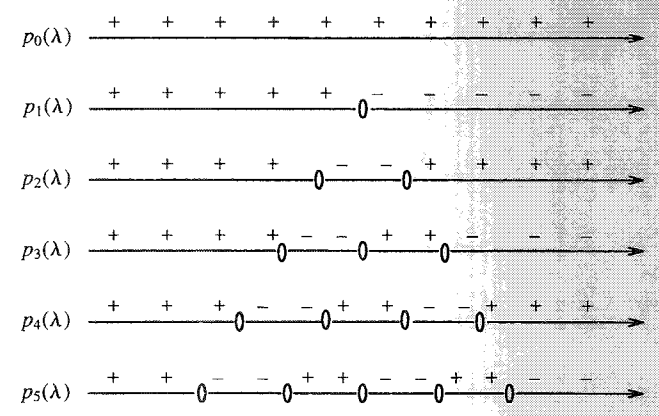


FIGURE 8.9

THEOREM 8.11.6 Interlacing Property Let T be an unreduced symmetric tridiagonal matrix and $\lambda_i^{(k)}$ be the ith smallest eigenvalue of its $k \times k$ principal submatrix. Then

$$\lambda_1^{(k+1)} < \lambda_1^{(k)} < \lambda_2^{(k+1)} < \cdots < \lambda_i^{(k+1)} < \lambda_i^{(k)} < \lambda_{i+1}^{(k+1)} < \cdots$$
$$< \lambda_k^{(k+1)} < \lambda_k^{(k)} < \lambda_{k+1}^{(k+1)}$$

(Golub and Van Loan 1989, 438).

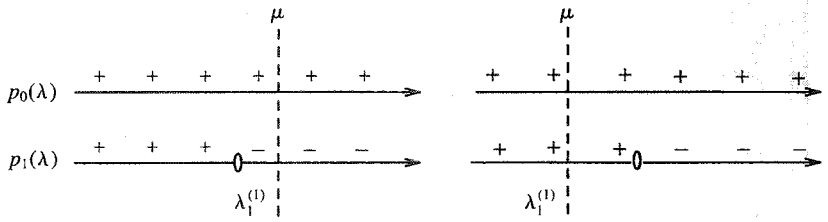
We shall show that this interlacing property leads to an interesting result on the number of eigenvalues of T in an interval.

It is clear that $p_k(\lambda) = (-1)^k \lambda^k + \cdots$. Thus, $p_k(\lambda)$ is positive if λ is negative with large magnitude. The zeros $\lambda_1^{(k)} < \lambda_2^{(k)} < \cdots < \lambda_k^{(k)}$ of $p_k(\lambda)$ separate the real line into $k + 1$ intervals:

$$(-\infty, \lambda_1^{(k)}), (\lambda_1^{(k)}, \lambda_2^{(k)}), \ldots, (\lambda_{k-1}^{(k)}, \lambda_k^{(k)}), (\lambda_k^{(k)}, \infty)$$

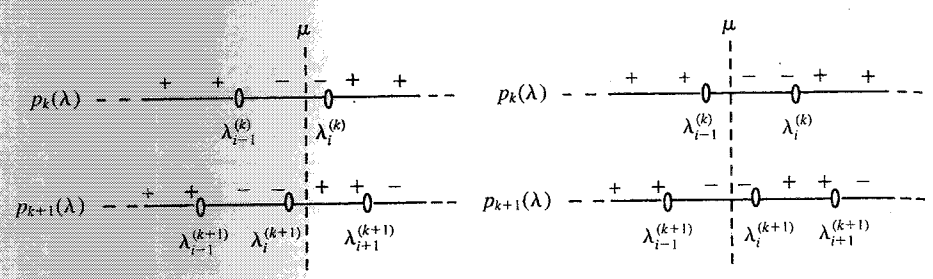
$p_k(\lambda)$ is positive in the first interval and takes alternate signs in consecutive intervals. The interlacing property and the sign changes of $p_k(\lambda)$ are illustrated in Figure 8.9.

Let μ be a real number. There are two cases for the signs of $p_0(\mu)$ and $p_1(\mu)$ (Figure 8.10).



Case 1: No sign agreement Case 2: Signs agree

FIGURE 8.10



Case 1: No sign agreement

Case 2: Signs agree

FIGURE 8.11

Clearly, a sign agreement between $p_0(\mu)$ and $p_1(\mu)$ "pushes" $\lambda_1^{(1)}$ to the right of μ. By the interlacing property, this implies

$$\mu < \lambda_1^{(1)} < \lambda_2^{(2)} < \cdots < \lambda_n^{(n)}$$

Thus, it pushes one eigenvalue $\lambda_n^{(n)}$ of A (or T) as well as one eigenvalue of each $k \times k$ submatrix to the right of μ.

Suppose we have counted the sign agreements between consecutive two terms of $p_0(\mu), p_1(\mu), \ldots, p_k(\mu)$. Let's consider the signs of $p_k(\mu)$ and $p_{k+1}(\mu)$ and the interlacing property (Figure 8.11).

Let μ be between $\lambda_{i-1}^{(k)}$ and $\lambda_i^{(k)}$. It is clear from Figure 8.11 that a sign agreement between $p_k(\mu)$ and $p_{k+1}(\mu)$ pushes λ_i^{k+1} to the right of μ, which means that p_{k+1} has one more zero than p_k in $[\mu, \infty)$.

There is no zero of p_0 in $[\mu, \infty)$. We know the number of zeros of p_1 in $[\mu, \infty)$ by the sign agreement or disagreement of $p_0(\mu)$ and $p_1(\mu)$. Generally, when we know the number of zeros of p_k in $[\mu, \infty)$, we know the number of zeros of p_{k+1} in $[\mu, \infty)$ by checking the sign agreement between $p_k(\mu)$ and $p_{k+1}(\mu)$. This recursion can be continued until we know the number of zeros of p_n—that is, the number of eigenvalues of T in $[\mu, \infty]$. The rule is clear:

THEOREM 8.11.7 The number of sign agreements between consecutive terms of $p_0(\mu)$, $p_1(\mu), \ldots, p_n(\mu)$ equals the number of eigenvalues of T larger than μ.

Note: It is also clear that $p_{k+1}(\mu) = 0$ should be considered a sign agreement with $p_k(\mu)$, because one more zero is pushed into $[\mu, \infty)$.

Example 8.11.3 Let

$$T = \begin{pmatrix} 2 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 2 \end{pmatrix}$$

$$p_0(\lambda) = 1, \; p_1(\lambda) = 2 - \lambda$$
$$p_2(\lambda) = (2 - \lambda)^2 - 1$$

$$\begin{aligned} p_3(\lambda) &= (2 - \lambda)p_2(\lambda) - p_1(\lambda) \\ &= (2 - \lambda)[(2 - \lambda)^2 - 1] - (2 - \lambda) \\ &= (2 - \lambda)^3 - 2(2 - \lambda) \end{aligned}$$

Let $\mu = 0$.

Then the sequence $p_0, p_1(\mu), p_2(\mu)$, and $p_3(\mu)$ is 1, 2, 3, 4. There are three agreements in sign. Thus, all the eigenvalues of T are greater than or equal to zero. In fact, because $p_3(0) = 4 \neq 0$, it follows that all the eigenvalues of T are positive. The eigenvalues of T are easily seen to be $2, 2 + \sqrt{2}$, and $2 - \sqrt{2}$.

Let $\mu = 2$ then, the sequence $p_0(\mu), p_1(\mu), p_2(\mu)$, and $p_3(\mu)$ is

$$1, 0, -1, 0$$

We count the agreements in signs:

$$+ \quad + \quad - \quad -$$

There are two agreements; confirming that T has two eigenvalues greater than or equal to 2.

The Sturm sequence-bisection method for computing a specified zero of $p_n(\lambda)$ now can be stated as follows:

ALGORITHM 8.11.1 The Sturm Sequence-Bisection Algorithm Let $\lambda_1 < \lambda_2 < \cdots < \lambda_n$ be the eigenvalues of T—that is, the zeros of $p_n(\lambda)$. Suppose the desired eigenvalue is λ_{n-m+1} for a given $m \leq n$. Then

Step 1: Find an interval $[s_1, s_2]$ containing λ_{n-m+1}. Because $\lambda_i \leq \|T\|$ initially, we can take

$$s_1 = -\|T\|_\infty, \; s_2 = \|T\|_\infty$$

Step 2: Compute $s_3 = \dfrac{s_1 + s_2}{2}$

Step 3: Compute $N(s_3)$ = the number of agreements in sign in the sequence

$$1, \; p_1(s_3), p_2(s_3), \ldots, p_n(s_3)$$

If $N(s_3) < m$, set $s_2 = s_3$; otherwise, set $s_1 = s_3$.

Let $\epsilon > 0$ be a preassigned small positive number. Test if $|s_2 - s_1| < \epsilon$. If so, accept $s_3 = (s_1 + s_2)/2$ as an approximate value of λ_{n-m+1}. Otherwise go to Step 2.

Note: After k steps, the desired zero is located in an interval of width $\dfrac{(s_2 - s_1)}{2^k}$.

Example 8.11.4 Let

$$A = \begin{pmatrix} 2 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 2 \end{pmatrix}$$

Suppose we want to approximate $\lambda_1 = 2 - \sqrt{2}$:

$$m = 3$$

Iteration 0: Initially,

$$s_1 = -4, \qquad s_2 = 4$$
$$s_3 = 0$$
$$N(s_3) = N(0) = 3$$
$$\text{Set } s_1 = s_3$$

Iteration 1:

$$s_1 = 0, \qquad s_2 = 4$$
$$s_3 = \frac{0+4}{2} = 2$$
$$N(s_3) = N(2) = 2 < 3$$
$$\text{Set } s_2 = s_3$$

Iteration 2:

$$s_1 = 0, \qquad s_2 = 2, \qquad s_3 = \frac{0+2}{2} = 1$$

$N(s_3)$ = number of agreements of sign in the sequence:

$$1,\ 1,\ 0,\ -1$$
$$(+\quad +\quad +\quad -)$$
$$N(s_3) = 2 < 3$$
$$\text{Set } s_2 = s_3$$

Iteration 3:

$$s_1 = 0, \qquad s_2 = 1, \qquad s_3 = 0.5$$
$$N(s_3) = 3$$
$$\text{Set } s_1 = s_3$$

The eigenvalue λ_1 is clearly in the interval [0.5, 1], which is, in fact, the case. We can continue our iterations until the length of the interval $|s_2 - s_1| < \epsilon$.

8.11.2 The Symmetric QR Iteration Method

When A is symmetric, the transformed Hessenberg matrix $PAP^T = H$ is also symmetric; therefore, it is symmetric tridiagonal. So, in this case we have to deal with the eigenvalue problem of a symmetric tridiagonal matrix rather than of a Hessenberg matrix.

To apply the QR iteration method to a symmetric tridiagonal matrix, we note that, if the starting matrix is a symmetric tridiagonal matrix T, then so is each matrix T_k in the sequence

$$T_k - \mu_k I = Q_k R_k$$

and, furthermore, we need only $0(n)$ flops to generate each T_k (note that the QR factorization of a symmetric tridiagonal matrix requires only $0(n)$ flops).

Also, because the eigenvalues of a symmetric matrix are all real, the double-shift strategy discussed for the general eigenvalue problem is not needed in this case. However, in this case a popular shift, known as the *Wilkinson shift*, is normally used.

Instead of taking the (n, n)th entry at every iteration as the shift, the eigenvalue of the trailing 2×2 matrix that is closer to the (n, n)th entry, is usually chosen as the shift. This is known as the **Wilkinson shift**. Thus, if a trailing 2×2 submatrix of T_k is given by

$$\begin{pmatrix} t_{n-1,n-1}^{(k)} & t_{n,n-1}^{(k)} \\ t_{n,n-1}^{(k)} & t_{nn}^{(k)} \end{pmatrix}$$

then the Wilkinson shift is

$$w = t_{nn}^{(k)} + r - \text{sign}(r)\sqrt{r^2 + \left(t_{n,n-1}^{(k)}\right)^2}$$

where

$$r = \frac{(t_{n-1,n-1}^{(k)} - t_{nn}^{(k)})}{2}$$

A better way to compute w is given in Exercise 38.

ALGORITHM 8.11.2 Symmetric QR Iteration

Step 1: Transform A to a symmetric tridiagonal matrix T using orthogonal similarity transformations:

$$PAP^T = T$$

Step 2: Apply single-shift QR iteration to T with the Wilkinson shift w:

Set $T = T_1$. Compute w as shown above.
For $k = 1, 2, \ldots$ do until convergence occurs
  $T_k - wI = Q_k R_k$
  $T_{k+1} = R_k Q_k + wI$
  Choose a new shift w.

Remark: It is possible to compute T_{k+1} from T_k without explicitly forming the matrix $T_k - w_k I$. This is known as *implicit symmetric QR*. See Exercise 29.

Convergence of the Symmetric QR Iteration

It can be shown (see Lawson and Hanson 1974, 109, and Parlett 1980, chapter 8) that the convergence of $t_{n,n-1}^{(k)}$ to zero is quadratic; that is, there exists $e > 0$ depending upon A such that for all k

$$|t_{n,n-1}^{(k+1)}| \le e|t_{n,n-1}^{(k)}|^2$$

Remark: In practice, it has been seen that the convergence is almost always cubic; but the quadratic convergence is all that has been proved.

Flop count. Since the QR factorization of a tridiagonal matrix requires $O(n)$ flops, it will require $O(n^2)$ flops to compute all the eigenvalues of a symmetric tridiagonal matrix using step 2 of Algorithm 8.11.2. The reduction of the symmetric matrix A to the symmetric tridiagonal matrix T will, of course, require about $\frac{2}{3}n^3$ flops.

Roundoff error property. As in the general nonsymmetric case, the **symmetric** QR with implicit shift is stable. It can be shown that, given a symmetric matrix A, the symmetric QR algorithm with implicit shift generates an orthogonal matrix Q and a diagonal matrix D such that

$$Q^T A Q = D + E$$

where

$$\|E\|_F \approx \mu\phi(n)\|A\|_F$$

and $\phi(n)$ is a slowly growing function of n.

> Furthermore, each computed eigenvalue $\hat{\lambda}_i$ satisfies the inequality
>
> $$|\lambda_i - \hat{\lambda}_i| \le \phi(n)\mu\|A\|_2$$
>
> Thus the absolute error in each computed eigenvalue is small, but the *relative error* in the tiny ones may be large.

8.11.3 The Divide-and-Conquer Method

As the title suggests, this method is based on "the divide and conquer" principle. The algorithm first *divides* a given symmetric tridiagonal eigenvalue problem into two smaller subproblems, and then glues the solutions of the subproblems to recover (*conquer*) the solution of the original problem. The method was originally suggested by Cuppen (1981).

The method can be used to compute all the eigenvalues and the corresponding eigenvectors of a symmetric matrix, and it is faster than the symmetric QR iteration method just described. We present here a very brief sketch of the method.

Suppose that the symmetric matrix A has been transformed to a symmetric tridiagonal matrix T by orthogonal similarity. Let

$$T = \begin{pmatrix} a_1 & b_1 & & 0 \\ b_1 & \ddots & \ddots & \\ & \ddots & \ddots & b_{n-1} \\ 0 & & b_{n-1} & a_n \end{pmatrix}$$

Define

$$T_1 = \begin{pmatrix} a_1 & b_1 & & 0 \\ b_1 & \ddots & \ddots & \\ & \ddots & a_{k-1} & b_{k-1} \\ 0 & & b_{k-1} & a_k - b_k \end{pmatrix}$$

$$T_2 = \begin{pmatrix} a_{k+1} - b_k & b_{k+1} & & 0 \\ b_{k+1} & \ddots & \ddots & \\ & \ddots & \ddots & b_{n-1} \\ 0 & & b_{n-1} & a_n \end{pmatrix}$$

Then

$$T = \begin{pmatrix} T_1 & 0 \\ 0 & T_2 \end{pmatrix} + b_k v v^T$$

where

$$v = (0,\ 0, \ldots,\ 1,\ 1,\ 0,\ \ldots,\ 0)^T$$

Since T_1 and T_2 are symmetric tridiagonal, we can find orthogonal matrices Q_1 and Q_2 such that

$$T_1 = Q_1 D_1 Q_1^T \qquad T_2 = Q_2 D_2 Q_2^T$$

Then

$$T = \begin{pmatrix} Q_1 & 0 \\ 0 & Q_2 \end{pmatrix}\left[\begin{pmatrix} D_1 & 0 \\ 0 & D_2 \end{pmatrix} + b_k u u^T\right]\begin{pmatrix} Q_1^T & 0 \\ 0 & Q_2^T \end{pmatrix}$$

where

$$u = \begin{pmatrix} Q_1^T & 0 \\ 0 & Q_2^T \end{pmatrix} v$$

Therefore, the eigenvalues of T are the same as those

$$\hat{D} = D + b_k u u^T = D + \rho u u^T$$

where D is a real $n \times n$ diagonal matrix given by

$$D = \begin{pmatrix} D_1 & 0 \\ 0 & D_2 \end{pmatrix}$$

and $\rho = b_k$. We therefore concentrate now on how to obtain the eigenvalues and eigenvectors of the rank-one perturbed diagonal matrix $\hat{D} = D + \rho u u^T$.

Assume without any loss of generality that $\|u\|_2 = 1$, and $\rho = b_k \neq 0$. Let $D = \mathrm{diag}(d_1, d_2, \ldots, d_n)$. Assume that $d_1 < d_2 < d_3 < \cdots < d_n$ and none of the components of the vector u is zero.

In fact, a zero component of u is a blessing in disguise. We can show (Exercise 41) that, in this case, we get an eigenvalue and eigenvector pair free. Furthermore,

this presents an opportunity for deflation. Also, if k eigenvalues of D are equal, then the problem can be deflated by deleting $(k-1)$ rows and columns (Exercise 42).

Let (λ, q) be an eigenpair of $\hat{D}$. Then we show that

1. λ is a root of the equation

$$1 + \rho u^T (D - \lambda I)^{-1} u = 0$$

and

2. $$q = (D - \lambda I)^{-1} u$$

is an eigenvector of $(D + \rho u u^T)$ corresponding to λ.

To show (1), we note that since (λ, q) is an eigenpair of $\hat{D}$ we must have

$$(D + \rho u u^T) q = \lambda q$$

that is,

$$(D - \lambda I) q = -\rho (u^T q) u$$

Now our assumptions that $\rho \neq 0$, $d_1 < d_2 < d_3 < \cdots < d_n$ and, that none of the components of u is zero, imply that (1) $(D - \lambda I)$ is nonsingular, and (2) $u^T q$ is nonzero (Exercise 44).

Multiplying the last equation by

$$(D - \lambda I)^{-1}$$

we have

$$q = -\rho (u^T q)(D - \lambda I)^{-1} u$$

Multiplying both sides by u^T and dividing by the nonzero scalars $u^T q$, we have

$$1 + \rho u^T (D - \lambda I)^{-1} u = 0$$

To show (2), we note that

$$\begin{aligned}
(D + \rho u u^T)(D - \lambda I)^{-1} u &= (D - \lambda I + \lambda I + \rho u u^T)(D - \lambda I)^{-1} u \\
&= u + \lambda (D - \lambda I)^{-1} u + u \rho u^T (D - \lambda I)^{-1} u \\
&= u + \lambda (D - \lambda I)^{-1} u + u(-1) \\
&= u + \lambda (D - \lambda I)^{-1} u - u \\
&= \lambda (D - \lambda I)^{-1} u
\end{aligned}$$

The equation

$$1 + \rho u^T (D - \lambda I)^{-1} u = 0$$

can be written in terms of the components u_i of u:

$$f(\lambda) = 1 + \rho u^T (D - \lambda I)^{-1} u = 1 + \rho \sum_{j=1}^{n} \frac{u_j^2}{d_j - \lambda} = 0$$

This equation is usually known as the **secular equation**.

Again, because d_i's are all distinct and none of the components of u is zero, we can show (Exercises 43 and 45) that $f(\lambda) = 0$ has precisely n roots, one in each of the intervals (d_j, d_{j+1}), $j = 1, 2, \ldots, n-1$ and one to the right of d_n if $\rho > 0$ or one to the left of d_1 if $\rho < 0$.

We can then apply the bisection-method (Algorithm 8.11.1) or Newton's method to find these roots in each of these intervals. Once a root is found, the corresponding eigenvector can be obtained from (2).

Note that if $q = (D - \lambda I)^{-1} u$, then the components of q are given by

$$q_i = \frac{u_i}{d_i - \lambda}, \quad i = 1, \ldots, n$$

The above discussion leads to the following algorithm.

ALGORITHM 8.11.3 Divide-and-conquer method Given a symmetric tridiagonal matrix T, the algorithm finds the eigenvalues and eigenvectors of T.

Step 1: Form

$$T = \begin{pmatrix} T_1 & 0 \\ 0 & T_2 \end{pmatrix} + b_k v v^T$$

Step 2: Find orthogonal matrices Q_1 and Q_2 such that $Q_1^T T_1 Q_1 = D_1$ and $Q_2^T T_2 Q_2 = D_2$.

Step 3: Form $D = \text{diag}(D_1, D_2) = \text{diag}(d_1 \cdots d_n)$ and $u = \text{diag}(Q_1^T, Q_2^T) v$.

Step 4: Find the eigenvalues of T by solving the secular equation

$$f(\lambda) = 1 + \rho \sum_{j=1}^{n} \frac{u_j^2}{d_j - \lambda} = 0$$

and obtain the eigenvectors of $D + \rho u u^T$ from Step 2.

Step 5: Recover the eigenvectors of T: If Q' is the eigenvector matrix of $D + \rho u u^T$, then the eigenvector matrix of T is given by

$$\begin{pmatrix} Q_1 & 0 \\ 0 & Q_2 \end{pmatrix} Q'$$

Flop count. Assuming that one Newton iteration step costs about $O(n)$ flops, the algorithm will require only $O(n^2)$ flops for all the n eigenvalues. The cost of computing each eigenvector from a computed eigenvalue is also $O(n)$ flops.

Remarks: Bunch, Nielsen, and Sorensen (1978) proposed a method for solving $f(\lambda) = 0$ using rational function approximations. Their method converges quadratically. For more on this method and implementational details, see Bunch, Nielsen, and Sorensen (1978) and the recent paper of Dongarra and Sorensen (1987). The divide-and-conquer method is naturally parallel.

Note that, because T_1 and T_2 are both symmetric tridiagonal, the eigenvalue problem for each of these matrices can further be decomposed into two subproblems,

resulting in four subproblems. These four subproblems again can be decomposed into eight smaller problems and the process can be continued until desired (possibly until the problem sizes become 1 by 1 or 2 by 2). Since each of these subproblems is independent, the original problem can be divided into many independent subproblems. See Dongarra and Sorensen (1987) for parallel implementational aspects of this method. In fact, the divide-and-conquer algorithm was originally targeted as a parallel scheme, but it turned out to be faster than the symmetric QR algorithm, if properly implemented. See the recent papers of Gu and Eisenstat (1994a, 1994b).

8.12 The Lanczos Algorithm for Symmetric Matrices

There are areas of applications, such as power systems, space science, quantum physics and chemistry, and nuclear studies, where the eigenvalue problems for matrices of very large order are commonly found.

Most large problems arising in applications are sparse. Research in this area is very active. The symmetric problem is better understood than the nonsymmetric problem. There are now well-established techniques to compute the spectrum or at least a part of the spectrum of very large and sparse symmetric matrices. A method originally devised by Lanczos (1950) has received considerable attention in this context. We discuss this method and its applications to eigenvalue computations in this section. The QR iteration method described in the last section is not practical for large and sparse eigenvalue problems. The sparsity is destroyed and storage becomes an issue.

The Symmetric Lanczos Algorithm

Given an $n \times n$ symmetric matrix A and a unit vector v_1, the Lanczos algorithm constructs simultaneously a symmetric tridiagonal matrix T and an orthonormal matrix V such that

$$T = V^T A V \tag{8.12.1}$$

The algorithm can be deduced easily as follows:

Let

$$T = \begin{pmatrix} \alpha_1 & \beta_1 & \cdots & 0 \\ \beta_1 & \alpha_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \beta_{n-1} \\ 0 & \cdots & \beta_{n-1} & \alpha_n \end{pmatrix}$$

and $V = (v_1, v_2, \ldots, v_n)$. Then the equation

$$V^T A V = T$$

or

$$AV = VT$$

or

$$A(v_1, v_2, \ldots, v_n) = (v_1, v_2, \ldots, v_n) \begin{bmatrix} \alpha_1 & \beta_1 & & & 0 \\ \beta_1 & \alpha_2 & \beta_2 & & \\ & \beta_2 & \ddots & \ddots & \\ 0 & & \ddots & \ddots & \beta_{n-1} \\ & & & \beta_{n-1} & \alpha_n \end{bmatrix}$$

gives

$$Av_j = \alpha_j v_j + \beta_{j-1} v_{j-1} + \beta_j v_{j+1}, \qquad j = 1, 2, \ldots, n-1 \tag{8.12.2}$$

(where we assume that $\beta_0 v_0 = 0$). Multiplying both sides of this equation by v_j^T to the left and observing that the orthonormality condition gives

$$v_j^T v_j = 1$$
$$v_j^T v_k = 0, \qquad j \neq k$$

we obtain

$$\alpha_j = v_j^T A v_j, \qquad j = 1, 2, \ldots, n \tag{8.12.3}$$

Also, if we write

$$r_j = Av_j - \alpha_j v_j - \beta_{j-1} v_{j-1} \tag{8.12.4}$$

then from (8.12.2) and (8.12.4), we get

$$v_{j+1} = \frac{r_j}{\beta_j}$$

provided that

$$\beta_j \neq 0$$

The nonvanishing of β_j can be assured if we take $\beta_j = \|r_j\|_2$.

This discussion gives us the following basic Lanczos algorithm, named after the Hungarian mathematician and physicist Cornelius Lanczos (1893–1974).

ALGORITHM 8.12.1 The Basic Symmetric Lanczos Given an $n \times n$ symmetric matrix A and a unit vector v_1, the following algorithm constructs simultaneously the entries of a symmetric tridiagonal matrix T and an orthonormal matrix $V = (v_1, \ldots, v_n)$ such that

$$V^T A V = T$$

Set $v_0 = 0$, $\beta_0 = 1$, $r_0 = v_1$.

For $j = 1, 2, \ldots, n$ do

$v_j = r_{j-1}/\beta_{j-1}$

$\alpha_j = v_j^T A v_j$

$r_j = (A - \alpha_j I) v_j - \beta_{j-1} v_{j-1}$

$\beta_j = \|r_j\|_2$

Notes:

1. The vectors $v_1, v_2, \ldots, v_n$ are called Lanczos vectors.
2. Each Lanczos vector v_{j+1} is orthogonal to all the previous ones, provided $\beta_j \neq 0$ (in exact arithmetic).
3. The whole algorithm can be implemented just by using a subroutine that can perform matrix–vector multiplication, and thus the sparsity of the original matrix A can be maintained. In fact, in contrast with Householder's method or Givens' method, the matrix A is never altered during the whole procedure. Such a feature makes the Lanczos algorithm attractive for sparse computations.
4. The vectors $\{v_1, \ldots, v_j\}$ form an orthonormal basis of the Krylov subspace $\kappa_j(A, v_1)$.
5. The Arnoldi method in the symmetric case described in Chapter 6 is the same as the symmetric Lanczos method described here.

We now present a modified Gram–Schmidt version of Algorithm 8.12.1, which is computationally more effective. Let $m \leq n$.

ALGORITHM 8.12.2 The Reformulated Lanczos

Set $v_0 = 0, \beta_0 = 1, r_0 = v_1$
For $j = 1, 2, \ldots, m$ do

Step 1: $v_j = r_{j-1}/\beta_{j-1}$

Step 2: $u_j \equiv Av_j$

Step 3: $r_j \equiv u_j - \beta_{j-1}v_{j-1}$

Step 4: $\alpha_j = v_j^T u_j$

Step 5: $r_j = r_j - \alpha_j v_j$

Step 6: $\beta_j = \|r_j\|$

Example 8.12.1

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \end{pmatrix}, \qquad r_0 = v_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \qquad \beta_0 = 1$$

$j = 1$:

$$\alpha_1 = 1, \qquad \beta_1 = 3.6056, \; v_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

$j = 2$:

$$\alpha_2 = 8.0769, \qquad \beta_2 = 0.6154, \; v_2 = \begin{pmatrix} 0 \\ 0.5547 \\ 0.8321 \end{pmatrix}$$

$$T_2 = \begin{pmatrix} \alpha_1 & \beta_1 \\ \beta_1 & \end{pmatrix} = \begin{pmatrix} 1 & 3.6056 \\ 3.6056 & 8.0769 \end{pmatrix}$$

$j = 3$:

$$\alpha_3 = -0.0769, \qquad \beta_3 = 1.5466 \times 10^{-14}, \qquad v_3 = \begin{pmatrix} 0 \\ 0.8321 \\ -0.5547 \end{pmatrix}$$

$$T = T_3 = \begin{pmatrix} 1 & 3.6056 & 0 \\ 3.6056 & 8.0769 & 0.6154 \\ 0 & 0.6154 & -0.0769 \end{pmatrix}$$

$$V = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0.5547 & 0.8321 \\ 0 & 0.8321 & -0.5547 \end{pmatrix}$$

Note that $V^T AV = T$.

Roundoff Properties of the Symmetric Lanczos Algorithm

It is clear from (8.12.2) that if the symmetric Lanczos algorithm is run from $j = 1$ to k ($k < n$), then we have

$$AV_j = V_j T_j + r_j e_j^T$$

where $V_j = (v_1, \ldots, v_j)$ and T_j is the $j \times j$ principal submatrix of T. If $\tilde{V}_j$, $\tilde{T}_j$, and $\tilde{r}_j$ denote the respective computed quantities, then it can be shown (Paige 1980) that

$$A\tilde{V}_j = \tilde{V}_j \tilde{T}_j + \tilde{r}_j e_j^T + E_j$$

where $\|E_j\|_2 \approx \mu\|A\|_2$, which shows that the Lanczos algorithm has very favorable numerical properties as far as the equation $AV_j = V_j T_j + r_j e_j^T$ is concerned. However, the loss of orthogonality among the Lanczos vectors is the real concern, as explained in the following.

Loss of Orthogonality

The Lanczos algorithm clearly breaks down when any of the $\beta_j = 0$. However, as we will see later, computationally this is a blessing in disguise. We immediately obtain an invariant subspace. This, however, seldom happens in practice. The real difficulty is that the computed $\tilde{\beta}_j$ can be very small, due to the cancellation that takes place during the computations of r_j, and a small $\tilde{\beta}_j$ can cause a severe loss of orthogonality among the computed vectors $\tilde{v}_j$, as can be seen from the following result that can easily be established using the basic laws of floating-point arithmetic:

$$|\tilde{v}_{j+1}^T \tilde{v}_j| \approx \frac{|\tilde{r}_j^T \tilde{v}_j| + \mu\|A\|_2}{|\tilde{\beta}_j|}$$

Thus, if $\tilde{\beta}_j$ is small, then $\tilde{v}_j$ and $\tilde{v}_{j+1}$ will be far from being orthogonal.

Because the Lanczos algorithm plays an important role in sparse matrix computations, a significant amount of research has been devoted to making the algorithm a practical one. There now exist procedures such as *Lanczos with complete orthogonalization*, which produces the Lanczos vectors that are orthogonal to working precision, or *Lanczos with selective orthogonalization*, which is used to enforce orthogonality only in selected vectors, whenever needed. For details, we refer the readers to well-known books in the area by Parlett (1980b) and by Cullum and Willoughby (1985). Several papers of Paige (1970, 1976, 1980), whose pioneering work in the early

1970s rejuvenated the interests of the researchers in this area, are also very useful reading. Procedures for complete and selective orthogonalization have been described in some detail in Golub and Van Loan (1989, 487–9). For selective orthogonalization, see Parlett and Scott (1979). See also the recent book by Saad (1992).

Computing the Eigenvalues of A

The sole purpose of presenting the Lanczos algorithm here is to show that the Lanczos matrices T_j can be used to compute certain eigenvalues of the matrix A.

We remarked earlier that when a β_j is exactly equal to zero, we have an invariant subspace. This is, indeed, good news. Unfortunately, this happens very rarely in practice. In practice, for large enough values of j, the eigenvalues of T_j provide very good approximations to the extremal eigenvalues of A. The question arises: Can we give posteriori bounds? To this end, we introduce the following definitions.

DEFINITION 8.12.1 Let (θ_j, s_j) be an eigenpair of T_j. Then the pair (θ_j, y_j), where y_j is defined by $y_j = V_j s_j$, is called the **Ritz pair**; the θ_j's are known as the **Ritz values** and the y_j's are called the **Ritz vectors**.

Now, returning to the question of how well a Ritz pair approximates an eigenpair of A, we state the following result which can be easily proved (Exercise 46).

THEOREM 8.12.1 Let $r_i = Ay_i - y_i\theta_i$, $i = 1, \ldots, j$. Then in each interval $[\theta_i - \|r_i\|_2, \theta_i + \|r_i\|_2]$, there is an eigenvalue of A.

Thus, it follows from Theorem 8.12.1 that $\|r_i\|_2$ is a good measure of how accurate the Ritz pair (θ_i, y_i) is.

Computing $\|r_i\|_2$

Fortunately, we can compute $\|r_i\|$ from T_j without computing $y_i = V_j s_i$ at every step. Consider the Schur decomposition of T_j:

$$S_j^T T_j S_j = \operatorname{diag}(\theta_1, \theta_2, \ldots, \theta_j)$$

and let

$$Y_j = (y_1, y_2, \ldots, y_j) = V_j S_j = V_j(s_1, s_2, \ldots, s_j)$$

Then using 2-norm, we have

$$\begin{aligned}
\|r_i\| &= \|Ay_i - y_i\theta_i\| = \|AV_j s_i - V_j s_i \theta_i\| \\
&= \|(AV_j - V_j T_j)s_i\| \qquad (\text{because } T_j s_i = s_i\theta_i) \\
&= \|(\beta_j v_{j+1} e_j^T)s_i\| \qquad (\text{note that } AV_j - V_j T_j = \beta_j v_{j+1} e_j^T) \\
&= |\beta_j|\,\|e_j^T s_i\| \qquad (\text{because } \|v_{j+1}\|_2 = 1) \\
&= |\beta_j|\,|s_{ji}|
\end{aligned}$$

where s_{ij} is the (j, i)th entry of S_j.

This discussion can be summarized in the following theorem:

THEOREM 8.12.2 Residual Theorem for a Ritz Pair Let T_j denote the $j \times j$ symmetric tridiagonal matrix obtained after j steps of the Lanczos algorithm. Let $S_j^T T_j S_j$ denote

the real Schur form of T_j,

$$S_j^T T_j S_j = \operatorname{diag}(\theta_1, \ldots, \theta_j)$$

that is, $\theta_1, \ldots, \theta_j$ are the eigenvalues of T_j. Let

$$Y_j = (y_1, y_2, \ldots, y_j) = V_j S_j$$

Then for each i from 1 to j, we have

$$\|r_i\|_2 = \|Ay_i - \theta_i y_i\| = |\beta_j|\,|s_{ji}|$$

where

$$s_{ji} = e_j^T s_i; \qquad s_i \text{ is the } i\text{th column of } S_j$$

Example 8.12.2

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \end{pmatrix}$$

The eigenvalues of A are: 0, -0.6235, and 9.6235.

$j = 2$:

$$T_2 = \begin{pmatrix} 1 & 3.6056 \\ 3.6056 & 8.0769 \end{pmatrix}, \qquad \beta_2' = 0.6154$$

$$S_2 = \begin{pmatrix} 0.9221 & 0.3870 \\ -0.3870 & 0.9221 \end{pmatrix}$$

$$\theta_1 = -0.5133, \qquad \theta_2 = 9.5903$$

(Note that $\theta_2 = 9.5903$ is a reasonably good approximation of the largest eigenvalue 9.6235 of A.)

$$\|r_1\|_2 = |\beta_2|\,|s_{21}| = 0.6154 \times 0.3970 = 0.2382$$
$$\|r_2\|_2 = |\beta_2|\,|s_{22}| = 0.5674$$

$j = 3$:

$$T_3 = \begin{pmatrix} 1 & 3.6056 & 0 \\ 3.6056 & 8.0769 & 0.6154 \\ 0 & 0.6154 & -0.0769 \end{pmatrix}$$

$$S_3 = \begin{pmatrix} 0.8277 & -0.4082 & 0.3851 \\ -0.3727 & 0.1132 & 0.9210 \\ 0.4196 & 0.9058 & 0.0584 \end{pmatrix}$$

$$\|r_1\| = |\beta_3|\,|s_{31}| = 6.487 \times 10^{-15}$$
$$\|r_2\| = |\beta_3|\,|s_{32}| = 1.4009 \times 10^{-14}$$
$$\|r_3\| = |\beta_3|\,|s_{33}| = 9.0367 \times 10^{-16}$$

(Note the smallness of r_1, r_2, and r_3 in this case.)

8.13 Review and Summary

This chapter has been devoted to the study of the eigenvalue problem, the problem of computing the eigenvalues and eigenvectors of a matrix A.

1. *Applications of the eigenvalues and eigenvectors:* The eigenvalue problem arises in a wide variety of practical applications. Mathematical models of many of the problems arising in engineering applications are systems of differential and difference equations, and the eigenvalue problem arises principally in the analysis of stability of these equations. Maintaining the stability of a system is a real concern for engineers. For example, in the study of vibrations of structures, the eigenvalues and eigenvectors are related to the natural frequencies and amplitude of the masses, and if any of the natural frequencies becomes equal or close to a frequency of the imposed periodic force on the structure, resonance occurs. This is a situation engineers would like to avoid. In this chapter we included examples on the European arms race, buckling of a beam, simulating transient current of an electric circuit, vibration of a building, and principal components analysis in statistics with a reference to a stock-market analysis. We have attempted to show just how important the eigenvalue problem is in practical applications.
2. *Localization of eigenvalues:* In several applications explicit knowledge of the eigenvalues is not required; all that is needed is a knowledge of the distribution of the eigenvalues in a region of the complex plane or estimates of some specific eigenvalues.
   (a) The Geršgorin disk theorems (Theorems 8.4.1 and 8.4.2) can be used to obtain a region of the complex plane containing all the eigenvalues or, in some cases, a number of the eigenvalues in a region. The estimates, however, can be very crude.
   (b) Also, $|\lambda| \leq \|A\|$ (Theorem 8.4.3). This result says that the upper bound of any eigenvalue of A can be found by computing its norm. Recall that this result is important in convergence analysis of iterative methods for linear systems.
3. *The power method and the inverse iteration:* There are applications such as analysis of dynamic systems, vibration analysis of structures, buckling of a beam, and principal component analysis in statistics where only the largest and the smallest (in magnitudes) eigenvalues or only the first or last few eigenvalues and their corresponding eigenvectors are needed.

   The power method and the inverse power method based on implicit construction of powers of A can be used to compute these eigenvalues and the eigenvectors. The power method is extremely simple to implement and is suitable for large and sparse matrices, but there are certain numerical limitations.

   In practice, the power method should be used with a suitable shift. The inverse power method is simply the power method applied to $(A - \sigma I)^{-1}$, where σ is a suitable shift. It is used to compute an eigenvector when a reasonably good approximation to an eigenvalue is known.

4. *The Rayleigh quotient iteration:* The quotient

$$R_q = \frac{x^T A x}{x^T x}$$

   known as the Rayleigh quotient, gives an estimate of an eigenvalue λ when an estimate of the eigenvector x is known. This idea, when combined with the inverse iteration method, can be used to compute an approximation to an eigenvalue and the corresponding eigenvector. The process is known as the Rayleigh quotient iteration (Algorithm 8.5.3).
5. *Sensitivity of eigenvalues and eigenvectors:*
   (a) The Bauer-Fike theorem (Theorem 8.7.1) tells us that if A is a diagonalizable matrix, then the condition number of the transforming matrix X, $\text{Cond}(X) = \|X\| \, \|X^{-1}\|$, plays the role of the condition number of the eigenvalue problem. If this number is large, then a small change in A can cause significant changes in the eigenvalues. Because a symmetric matrix A can be transformed to a diagonal matrix by orthogonal similarity and the condition number of an orthogonal matrix (with respect to 2-norm) is 1, it immediately follows from the Bauer–Fike theorem that the eigenvalues of a symmetric matrix are insensitive to small perturbations. See also Theorem 8.11.2.
   (b) If an eigenvalue problem is ill conditioned, then it might happen that some eigenvalues are more sensitive than the others. It is thus important to know the sensitivity of the individual eigenvalues. Unfortunately, to measure the sensitivity of an individual eigenvalue, one needs the knowledge of both left and right eigenvectors corresponding to that eigenvalue. The condition number of the eigenvalue λ_i is the reciprocal of the number $|y_i^T x_i|$, where x_i, and y_i are, respectively, the normalized right and left eigenvectors corresponding to λ_i.
   (c) The sensitivity of an eigenvector x_k corresponding to an eigenvalue λ_k depends upon (i) the condition number of all the eigenvalues other than λ_k and (ii) the distance of λ_k from the other eigenvalues (Theorem 8.8.1). Thus, if the eigenvalues are well separated and well conditioned, then the eigenvectors are well conditioned. On the other hand, if there is a multiple eigenvalue or there is an eigenvalue close to another eigenvalue, then there are some ill-conditioned eigenvectors. This is especially significant for a symmetric matrix. The eigenvalues of a symmetric matrix are well conditioned, but the eigenvectors can be quite ill conditioned.
6. *Eigenvalue computation via the characteristic polynomial and the Jordan canonical form:* A similarity transformation preserves the eigenvalues, and it is well known that a matrix A can be transformed by similarity to the Jordan canonical form (JCF) and to the Frobenius form (or a companion form if A is nonderogatory). The eigenvalues of these condensed forms are rather easily computed. The JCF displays the eigenvalues explicitly, and with the companion or Frobenius form, the characteristic polynomial of A is trivially computed; then a root-finding method can be applied to the

characteristic polynomial to obtain the eigenvalues, which are the zeros of the characteristic polynomial.

However, computation of eigenvalues via the characteristic polynomial or the JCF is not recommended in practice. Obtaining these forms may require a very ill-conditioned transforming matrix, and the sensitivity of the eigenvalue problem depends upon the condition number of this transforming matrix. In general, ill-conditioned similarity transformation should be avoided in eigenvalue computation.

The use of well-conditioned transforming matrices, such as orthogonal matrices, is desirable.

7. *The QR iteration algorithm:* The most widely used algorithm for finding the eigenvalues of a matrix is the QR iteration algorithm. For a real matrix A, the algorithm basically constructs iteratively the real Schur form (RSF) of A by orthogonal similarity. Because the algorithm is based on repeated QR factorizations and each QR factorization of an $n \times n$ matrix requires $O(n^3)$ flops, the n steps of QR iteration algorithm, if implemented naively (which we call the basic QR iteration), will require $O(n^4)$ flops, making the algorithm impractical.

   (a) The matrix A is, therefore, initially reduced to a Hessenberg matrix H by orthogonal similarity before the start of the QR iteration. The key observations here are that the reduction of A to H has to be made once for all, and the Hessenberg form is preserved at each iteration.

   (b) The convergence of the Hessenberg QR iteration algorithm, however, can be quite slow in the presence of a near-multiple or a multiple eigenvalue. The convergence can be accelerated by using suitable shifts. In practice, double shifts are used. At each iteration, the shifts are the eigenvalues of the 2×2 submatrix at the bottom right-hand corner. Because the eigenvalues of a real matrix can be complex, complex arithmetic is usually required. However, computations can be arranged so that complex arithmetic can be avoided. Also, the eigenvalues of the 2×2 bottom right-hand corner matrix at each iteration do not need to be computed explicitly. The process is known as the double-shift implicit QR iteration (Section 8.9.6).

   With double shifts, the eigenvalues are computed two at a time. Once two eigenvalues are computed, the matrix is deflated, and the process is applied to the deflated matrix. *The double-shift implicit QR iteration seems to be the most practical algorithm for computing the eigenvalues of a dense matrix of modest size.*

8. *Ordering the eigenvalues:* The eigenvalues appearing in RSF obtained by the QR iteration algorithm may not appear in the desired order, although there are some applications that need this. However, with a little extra work, the eigenvalues can be put in any desired order. There is an excellent Fortran routine, designed by Stewart, that will accomplish this (Section 8.9.8).

9. *Computing the eigenvectors:* Once an approximation to an eigenvalue is obtained by the QR iteration, inverse iteration can be invoked to compute the corresponding eigenvector. Because the matrix A is initially reduced to a Hessenberg matrix for practical implementation of the QR iteration algorithm, advantage can be taken of the structure of a Hessenberg matrix in computing an eigenvector using inverse iteration. Alternatively, we can compute the eigenvectors directly from the RSF of A (Section 8.10.2).

10. *The symmetric eigenvalue problem:*

    (a) The QR iteration algorithm (Algorithm 8.11.2), of course, can be used to compute the eigenvalues of a symmetric matrix. A shift called the Wilkinson shift is normally used here. The convergence of the symmetric QR iteration algorithm with the Wilkinson shift has been proven to be quadratic; however, in practice it is often cubic.

    (b) The eigenvalues and eigenvectors of a symmetric matrix A enjoy some remarkable special properties, and there are methods for the symmetric problem that can exploit these properties. One such method is the **bisection method**. The given symmetric matrix A is first transformed to a symmetric tridiagonal matrix T using orthogonal similarity, and then the well-known bisection algorithm for the root-finding problem is applied to the characteristic polynomial of T, obtained by a simple recursion (Section 8.11.1). This recursion not only gives the characteristic polynomial of T, but also gives the characteristic polynomials of all the leading principal submatrices. A remarkable fact is that these polynomials have the **Sturm sequence property**, which is used in the implementation of the bisection method. The bisection method is especially useful for finding eigenvalues of a symmetric matrix in a given interval of the real line.

    (c) There are other methods, such as the **divide-and-conquer method** and the **Jacobi method**, for the symmetric eigenvalue problem. We have only briefly introduced the former (Algorithm 8.11.3) in this chapter. These methods are important, especially for parallel computations.

11. *Large and sparse eigenvalue problem:* The eigenvalue problem for large and sparse matrices is an active area of research. The state-of-the-art techniques using the Lanczos or Arnoldi type of methods with some sort of reorthogonalization and proper preconditioning can compute only a few extremal eigenvalues. The techniques for large and sparse symmetric eigenvalue problems are more well established and better understood than those for the nonsymmetric problem.

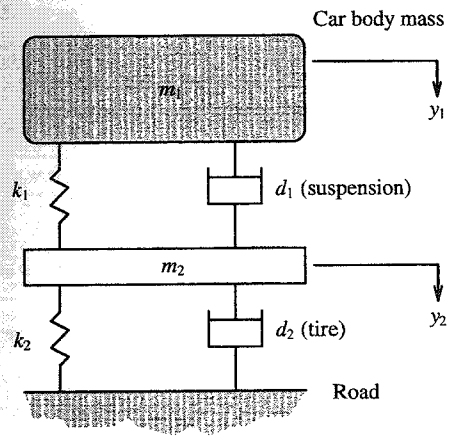
    For the sake of completeness and to give the readers an idea of how the Lanczos-type methods are used in eigenvalue computations, we included a brief description of the symmetric Lanczos method (Section 8.12).

EXERCISES

Note: Use MATLAB whenever needed and appropriate.

Section 8.3

1. Consider the following model for the vertical vibration of a motor car (Muller and Schienlen, 1985):



(a) Formulate the following equations of motion of the car, neglecting the damping constants d_1 of the shock absorber and d_2 of the tire:

$$M\ddot{y} + Ky = 0$$

where $M = \text{diag}(m_1, m_2)$,

$$K = \begin{pmatrix} k_1 & -k_1 \\ -k_1 & k_1 + k_2 \end{pmatrix}$$

and

$$y = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$$

Determine the stability of motion when

$$m_1 = m_2 = 1200 \text{ kg}$$

$$k_1 = k_2 = 300 \frac{\text{N}}{\text{cm}}$$

(b) Show that the equation of motion when just the damping d_2 of the tire is neglected is given by:

$$M\ddot{y} + D\dot{y} + Ky = 0$$

M, and K are the same as in part (a), and

$$D = \begin{pmatrix} d_1 & -d_1 \\ -d_1 & d_1 \end{pmatrix}$$

Investigate the stability of motion in this case when

$$d_1 = 4500 \frac{Ns}{m}$$

Hint: Show that the system

$$M\ddot{y} + D\dot{y} + Ky = 0$$

is equivalent to the first-order system

$$\dot{x}(t) = Ax(t)$$

where

$$A = \begin{pmatrix} 0 & I \\ -M^{-1}K & -M^{-1}D \end{pmatrix}$$

and

$$x(t) = \begin{pmatrix} y(t) \\ \dot{y}(t) \end{pmatrix}$$

2. Write the solution of the equation $M\ddot{y} + Ky = 0$ with M and K as given in Problem 1(a), using initial conditions $y(0) = 0$ and $\dot{y}(0) = (1, 1, \ldots, 1)^T$.
3. Develop an eigenvalue problem for an LC network similar to the case study given in Section 8.2.2, but with only three loops.

   Show that in this case the natural frequencies are given by

$$w = \begin{cases} \dfrac{0.4451}{\sqrt{LC}} \\ \dfrac{1.2470}{\sqrt{LC}} \\ \dfrac{1.8019}{\sqrt{LC}} \end{cases}$$

   Find the modes, and illustrate how the currents oscillate in these modes.

Section 8.4

4. Apply the Geršgorin disk theorems to obtain bounds of the eigenvalues for the following matrices:

(a) $A = \begin{pmatrix} 10 & 1 & 1 \\ 2 & 10 & 1 \\ 2 & 2 & 10 \end{pmatrix}$

(b) $\begin{pmatrix} 1 & 0 & 0 \\ 2 & 5 & 0 \\ 1 & 1 & 6 \end{pmatrix}$

(c) $\begin{pmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{pmatrix}$

(d) $\begin{pmatrix} 1 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{pmatrix}$

(e) $\begin{pmatrix} 1.000 & 0.577 & 0.509 & 0.387 & 0.462 \\ 0.577 & 1.000 & 0.599 & 0.389 & 0.322 \\ 0.509 & 0.599 & 1.000 & 0.436 & 0.426 \\ 0.387 & 0.389 & 0.436 & 1.000 & 0.523 \\ 0.452 & 0.322 & 0.426 & 0.523 & 1.000 \end{pmatrix}$

(f) $\begin{pmatrix} 1-i & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1-i & 1+i \end{pmatrix}$

5. Using a Geršgorin disk theorem, prove that a diagonally dominant matrix is nonsingular.
6. Let x be an eigenvector corresponding to an isolated eigenvalue λ in the Geršgorin disk R_k. Prove that $|x_k| > |x_i|$ for $i \neq k$.
7. Let $A = (a_{ij})$ be an $n \times n$ symmetric matrix. Then using a Geršgorin disk theorem, prove that each eigenvalue of A will lie in one of the intervals: $[a_{ij} - r_i, a_{ij} + r_i]$. Find an interval in which all the eigenvalues of A must lie.

Section 8.5

8. Applying the power method and inverse power method, find the dominant eigenvalue and the corresponding eigenvector for each of the matrices in the Exercise 4.
9. Prove that if $\lambda_1, \ldots, \lambda_n$ are the eigenvalues of A and $v_1, \ldots, v_n$ are the corresponding eigenvectors, then $\lambda_1 - \sigma, \ldots, \lambda_n - \sigma$ are the eigenvalues of $A - \sigma I$, and the corresponding eigenvectors are $v_1, \ldots, v_n$.
10. Explain the slow rate of convergence of the power method with the following matrices:

(a) $A = \begin{pmatrix} 3 & 3 & 2 \\ 0 & 1 & 0 \\ 0 & 1 & 2.9 \end{pmatrix}$

(b) $A = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 10 & 0 \\ 1 & 1 & 9.8 \end{pmatrix}$

Choose a suitable shift σ and then apply the shifted power method to each of the matrices and observe the improvement of the rates of convergence.

11. *Orthogonal Iteration:* The following iterative procedure generalizes the power method and is known as the **orthogonal iteration process**. The process can be used to compute the p ($p > 1$) largest eigenvalues and the corresponding eigenvectors. Let Q_1 be an $n \times p$ orthonormal matrix. Then

For $k = 2, 3, \ldots$ do

Step 1: Compute $B_k = AQ_{k-1}$

Step 2: Factorize into QR: $B_k = Q_k R_k$

Write a complete algorithm based on this method to compute the first p eigenvalues and the corresponding eigenvectors and then test your algorithm with each of the matrices in Exercise 10, taking $p = 2$.

12. *Inverse Orthogonal Iteration:* The following iteration, called the **inverse orthogonal iteration**, generalizes the inverse power method and can be used to compute the p smallest eigenvalues of a nonsingular matrix A. Let Q_1 be an $n \times p$ orthonormal matrix.

For $k = 2, 3, \ldots$

Step 1: Solve for B_k: $AB_k = Q_{k-1}$

Step 2: Factorize into QR: $B_k = Q_k R_k$

Apply the inverse orthogonal iteration to compute the two smallest (least dominant) eigenvalues of each of the matrices in Exercise 4.

13. Let T be a symmetric tridiagonal matrix. Let the Rayleigh quotient iteration algorithm (Algorithm 8.5.3) using 2-norm in Step 3 be applied to T with $x_0 = e_n$; then prove that $x_1 = q_n$, where q_n is the last column of Q in $(T - \sigma_0 I) = QR$.
14. Compute the subdominant eigenvalue of each of the matrices in Exercise 4 using Householder deflation. Then compute the corresponding eigenvector without invoking the inverse iteration.
15. Compute the smallest eigenvalue of each of the matrices A in Exercise 4 by applying the power method to A^{-1}, without explicitly computing A^{-1}.
16. *Deflation Using an Invariant Subspace:* Suppose that we have an $n \times m$ matrix X with independent columns and an $m \times m$ matrix M such that

$$AX = XM$$

Consider the QR factorization of X:

$$Q^T X = \begin{pmatrix} R \\ 0 \end{pmatrix}$$

Then show the following.

(a) $Q^T A Q = \begin{pmatrix} A_1 & A_2 \\ 0 & A_3 \end{pmatrix}$

(b) The eigenvalues of A are those of A_1 and A_3.

(c) The eigenvalues of A_1 are those of M.

Sections 8.6–8.8

17. Construct a simple example to show that an arbitrary similarity transformation can worsen the conditioning of the eigenvalues of the transformed matrix.
18. If $B = U^* A U$, where U is unitary, and $U^*(A + \Delta A)U = B + \Delta B$, then show that $\|\Delta B\|_2 = \|\Delta A\|_2$.
19. (a) Prove that A is normal if there exists a unitary matrix U such that $U^* A U = D$, where D is diagonal.

(b) Prove that a matrix A has a set of n orthonormal eigenvectors if and only if A is normal.

(c) Prove that a normal matrix A is unitary if and only if its eigenvalues are on the unit circle; that is, for each eigenvalue λ of A, we have $|\lambda| = 1$.

(d) How does the real Schur form of a normal matrix look?

(e) Using the eigenvector-sensitivity theorem (Theorem 8.8.1), show that if A is normal, then the eigenvector x_k corresponding to the eigenvalue λ_k is well conditioned if λ_k is well separated from the other eigenvalues.

20. Show both theoretically and experimentally that two of the eigenvectors of the matrix

$$A = \text{diag}(1 + \epsilon, 1 - \epsilon, 2)$$

where ϵ is a very small positive number, are ill conditioned. However, the eigenvalues are well conditioned.

21. Show that the unitary similarity transformation preserves the 2-norm condition number of an eigenvalue.
22. Prove the Bauer–Fike theorem (Theorem 8.7.1) using Geršgorin's first theorem.
23. (a) Given ϵ, a small positive number, and

$$A = \begin{pmatrix} 1 & 1 \\ 0 & 1+\epsilon \end{pmatrix}$$

find the eigenvector-matrix X such that $X^{-1}AX$ is diagonal; hence show that the eigenvalues of A are ill conditioned. Verify the ill-conditioning of the eigenvalues of A computationally by constructing a small perturbation to A and finding the eigenvalues of the perturbed matrix.

(b) Consider

$$A = \begin{pmatrix} 12 & 11 & 10 & \cdots & \cdots & 2 & 1 \\ 11 & 11 & 10 & \cdots & \cdots & 2 & 1 \\ 0 & 10 & 10 & \cdots & \cdots & 2 & 1 \\ \vdots & \ddots & \ddots & \ddots & & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & \cdots & 0 & 1 & 1 \end{pmatrix}$$

Show that the largest eigenvalues of A are well conditioned, whereas the smallest ones are very ill conditioned, by computing the eigenvalues using MATLAB.

Sections 8.9 and 8.10

24. Prove that if R is upper triangular and Q is upper Hessenberg, then RQ is upper Hessenberg.
25. Construct an example to show that for an arbitrary upper Hessenberg matrix H, the Q-matrix in

$$H = QR$$

need not be an upper Hessenberg. However, if R has nonzero diagonal entries, then Q in $H = QR$ is always upper Hessenberg.

26. Prove that
    (a) The QR factorization of an $n \times n$ Hessenberg matrix requires $0(n^2)$ flops.
    (b) The QR factorization of a symmetric tridiagonal matrix requires only $0(n)$ flops.
27. Apply three iterations of the single-shift QR iteration to each of the following matrices and observe the convergence or nonconvergence of the subdiagonal entries.

(a) $A = \begin{pmatrix} 1 & 2 & 0 \\ 2 & 3 & 4 \\ 0 & 4 & 1 \end{pmatrix}$

(b) $A = \begin{pmatrix} 4 & 5 & 6 \\ 1 & 0 & 1 \\ 0 & -2 & 2 \end{pmatrix}$

28. Perform one step of the single-shift QR iteration with the matrix

$$H = \begin{pmatrix} 1 & 1 & 1 \\ 2 & 10 & 6 \\ 0 & 0.1 & 1 \end{pmatrix}$$

29. *Implicit Single-Shift QR:* Consider one step of the single-shift QR iteration:

$$H_k - h_{nn}^{(k)}I = Q_kR_k, \quad H_{k+1} = R_kQ_k + h_{nn}^{(k)}I$$

or (simply)

$$H - \lambda I = QR, \qquad \overline{H} = RQ + \lambda I, \qquad (\lambda \text{ is real})$$

(a) Prove that the first column of Q is a multiple of the first column of $H - \lambda I$ and, therefore, contains only two nonzero entries.
(b) Denote the first column of $H - \lambda I$ by $h_1 = (h_{11} - \lambda, h_{21}, 0, \ldots, 0)^T$. Find a Givens rotation P_0 such that P_0h_1 is a multiple of e_1. Show that the first column of P_0 is the same as the first column of Q, except possibly for the sign.
(c) Form $H' = P_0HP_0^T$. Using Givens rotations $J(2, 3, \theta), J(3, 4, \theta), \ldots, J(n-1, n, \theta)$ transform H' to an upper Hessenberg matrix H_1'. Show that the matrix

$$\tilde{Q} = P_0^T J^T(2, 3, \theta) \ldots J^T(n-1, n, \theta)$$

has the same first column as Q. Finally, conclude from the implicit Q theorem that the Hessenberg matrix H_1' is essentially the same as $\overline{H}$.

Steps (a) through (c) constitute one step of the implicit single-shift QR iteration. Apply one step of the implicit single-shift QR iteration to the symmetric tridiagonal matrix

$$\begin{pmatrix} 2 & -1 & \cdots & 0 \\ -1 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & -1 \\ 0 & \cdots & -1 & 2 \end{pmatrix}$$

30. Construct one step of the explicit double-shift QR iteration (real arithmetic) and one step of the implicit double-shift QR iteration for the matrix

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 4 & 5 & 6 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & -2 & 2 \end{pmatrix}$$

and show that the final Hessenberg matrices are (essentially) the same.

31. *LR Iteration:* Analogous to the basic QR iteration algorithm (Section 8.9.1), develop an LR iteration algorithm, based on LU decomposition of A, making the necessary assumptions. Why is this algorithm not to be preferred over the QR iteration algorithm?

32. Considering the structures of the matrices P_i, $i = 0, 1, \ldots, n-2$ in the implicit double-shift QR iteration step (Exercise 29), show that it requires only $4n^2$ flops to implement this step.
33. Show that the matrices H_s and H_{s+2} in the double-shift QR iteration have the same eigenvalues.
34. Prove the following: Let $H = H_0$. Generate the sequence $\{H_k\}$:

$$H_k - \mu_k I = Q_k R_k, \qquad H_{k+l} = R_k Q_k + \mu_k I$$

Then

$$\prod_{i=1}^{n}(H - \mu_i I) = (Q_1 \ldots Q_n)(R_n \ldots R_1)$$

Sections 8.11 and 8.12

35. (a) Let

$$A = \begin{pmatrix} 1 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{pmatrix}$$

Without computing the eigenvalues show that $|\lambda| < 4$ for each eigenvalue λ of A. Show that there are exactly two eigenvalues greater than 2 and two less than 2. Apply the Sturm sequence-bisection algorithm (Algorithm 8.11.1) to compute the eigenvalue close to 2.

(b) Apply the inverse Hessenberg iteration algorithm to A to compute the eigenvector associated with the eigenvalue close to 2.

(c) Compute the eigenvalues of A by applying the symmetric QR iteration with the Wilkinson shift.

36. (a) Prove that the eigenvalues of an unreduced real symmetric tridiagonal matrix are real and distinct.

(b) Prove that if λ is an eigenvalue of multiplicity k of an unreduced symmetric tridiagonal matrix T, then at least $(k-1)$ subdiagonal entries of T must be zero.

37. (a) Develop a QR-type algorithm to compute the eigenvalues of a symmetric positive definite matrix A, based upon the Cholesky decomposition (Section 6.4.6).

(b) Test your algorithm with the matrix A of Exercise 35.

38. Let

$$A = \begin{pmatrix} \alpha & \beta \\ \beta & \gamma \end{pmatrix}$$

Prove that the eigenvalue δ of A closest to γ is given by

$$w = \gamma - \frac{\operatorname{sign}(r)\beta^2}{|r| + \sqrt{r^2 + \beta^2}}$$

where $r = (\alpha - \gamma)/2$. Explain why this formula is better than the one given in Section 8.11.2.

39. Let $A = A_1 + iA_2$ be a Hermitian matrix. Then prove that

$$B = \begin{pmatrix} A_1 & -A_2 \\ A_2 & A_1 \end{pmatrix}$$

is symmetric. How are the eigenvalues and eigenvectors of A related to those of B?

40. Take the 50×50 symmetric tridiagonal matrix T with 2 along the diagonal and -1 along the subdiagonal and superdiagonal. Apply Algorithm 8.12.2 to T with $j = 1, 2, \ldots, 20$. Find approximations to a few extreme eigenvalues of T using Theorems 8.12.1 and 8.12.2.

Exercises 41 through 45 are on the divide-and-conquer method (Section 8.11.3).

41. Prove that if the ith component u_i of u is zero, then

(a) the ith column of $\hat{D}$ is $d_i e_i$

(b) the ith row of $\hat{D}$ is $d_i e_i^T$

(c) d_i is an eigenvalue of $\hat{D}$ with the associated eigenvector e_i

42. Prove that if k eigenvalues of D are equal, then the eigenvalue problem of $\hat{D}$ can be deflated by deleting $(k-1)$ rows and columns.
43. Assume that

(a) $\rho \neq 0$,

(b) the eigenvalues of D are arranged in $d_1 < d_2 < \cdots < d_n$, and

(c) none of the components of u is zero.

Then, prove that each interval (d_i, d_{i+1}) contains exactly one eigenvalue of $\hat{D} = D + \rho u u^T$.

44. Prove that if the assumptions of Exercise 43 hold, then $D - \lambda I$ is nonsingular and $u^T q \neq 0$.
45. By drawing the graphs of the secular equation or otherwise, prove that

(a) $d_1 < \lambda_1 < d_2 < \lambda_2 < \cdots < \lambda_n < d_n + \rho$, if $\rho > 0$

(b) $d_1 + \rho < \lambda_1 < d_2 < \lambda_2 \cdots < d_{n-1} < \lambda_n < d_n$, if $\rho > 0$.

46. Prove Theorem 8.12.1.

MATLAB PROGRAMS AND PROBLEMS

1. Write a MATLAB program to compute the dominant eigenvalue of a matrix using the power method

[lambda1] = **power**(A, $x0$, N)

where N is the number of iterations to be performed. Print the result of each iteration.

(a) Modify the program **power** to incorporate a shift sigma

[lambda1] = **powershift**(A, $x0$, sigma, N)

(b) Apply **power** and **powershift** to the following matrices and compare the speed of convergence.

Test Data

$$A = \begin{pmatrix} 3 & 2 & 3 \\ 0 & 0.99 & 1 \\ 0 & 0 & 2.9 \end{pmatrix}$$

A = a randomly generated matrix of order 15

A = the Wilkinson bidiagonal matrix of order 20

2. (a) Using **linsyspp** from Chapter 6 or the MATLAB command '\', write a MATLAB program called **invitr** to implement the inverse iteration algorithm

   [x] = **invitr**(A, $x0$, sigma, N),

   N is the number of iterations to be performed.

   (b) Using **linsyspp** from Chapter 6 or the MATLAB command '\', write a MATLAB program called **powersmall** to compute the smallest eigenvalue (in magnitude) of a matrix A

   [lambdan] = **powersmall**(A, $x0$, N),

   N is the number of iterations to be performed.

Test Data and Experiment for 2(a) and 2(b)

1. Take the 20×20 symmetric tridiagonal matrix appearing in the buckling problem of Section 8.3.2. Apply **power** to compute the dominant eigenvalue lambda1 by choosing $x0$ arbitrarily.
2. Now compute the smallest eigenvalue in magnitude, **lambdan**, by using
   (a) **powershift** with sigma = lambda1 and $x0$ as in 1.
   (b) **powersmall** with the same $x0$ as used to implement **power**.
3. Compare the speed of (a) and (b) in 2.
4. Find the smallest critical load that will buckle the beam
5. Taking sigma = lambdan, find the eigenvector corresponding to the smallest eigenvalue λ_n using **invitr**.

3. Using **power**, **invitr**, **houszero** (from Chapter 5), and **phousmul** (from Chapter 4), write a MATLAB program to implement the Householder deflation algorithm that computes the subdominant eigenvalue of a matrix:

   [lambda2] = **housdeflt**(a)

Test Data

A single-sex, cohort population model can be represented by the following system of difference equations:

$$p_0(k+1) = \alpha_0 p_0(k) + \alpha_1 p_1(k) + \cdots + \alpha_n p_n(k)$$
$$p_{i+1}(k+1) = \beta_i p_i(k), \qquad i = 0, 1, \ldots, n-1$$

or in matrix form,

$$p_{k+1} = A p_k$$

Here α_i is the birth rate of the ith age group and β_i is the rate of survival of that group (Luenberger 1979, 170). Let $n = 50$.

Taking $\alpha_0 = 0, \alpha_1 = \alpha_2 = \cdots = \alpha_n = 1$, and $\beta_i = 1, i = 0, 1, \ldots, n-1$, determine by using **power**, **invitr**, and **housdeflt** if there is a long-term growth in the population; if this is so, what is the final population distribution and how fast is the original distribution approaching the final distribution? Consult the example on population study in Section 8.5.2.

4. Plot the Geršgorin disks of each of the matrices A in Exercise 4, Section 8.4.
5. *(The purpose of this exercise is to study how the eigenvalues of a matrix A are affected by conditioning of the transforming matrix.)*
   (a) For each of the following matrices, construct a matrix X of appropriate order that is upper triangular with all the entries equal to 1 except for a few very small diagonal entries. Then compute the eigenvalues of A and those of $(X)^{-1}AX$ with MATLAB commands **eig** and **inv** using format "long" in MATLAB:

$$A = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0.9 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \qquad A = \begin{pmatrix} 0 & 0 & 2 \\ 1 & 0 & -5 \\ 0 & 1 & 4 \end{pmatrix}$$

   A = the Wilkinson bidiagonal matrix of order 20

   (b) Repeat part (a) by taking X as a Householder matrix of appropriate order.
   (c) Compare the results of (a) and (b).
6. (a) Compute the eigenvalues of the following matrices using:
   (i) MATLAB commands **poly** and **roots**.
   (ii) MATLAB command **eig**.

$$A = \begin{pmatrix} 0 & 0 & 2 \\ 1 & 0 & -5 \\ 0 & 1 & 4 \end{pmatrix}, \qquad A = \begin{pmatrix} 1 & 1 & 1 \\ 0 & .19 & 1 \\ 0 & 0 & 0.9999 \end{pmatrix}$$

   A = the Wilkinson bidiagonal matrix of order 20

   A = random matrix of order 50

   (b) Compare your results in (i) and (ii) for each matrix.

7. *(The purpose of this exercise is to study the sensitivities of the eigenvalues of some well-known matrices with ill-conditioned eigenvalues.)*

Perform the following on each of the matrices in the test data:

(a) Using the MATLAB command $[V, D] = \mathbf{eig}(A)$, find the eigenvalues and the matrix of right eigenvectors. Then find the matrix of left eigenvectors W as shown in Section 8.7.2.

(b) Compute $s_i = w_i^T v_i$, $i = 1, \ldots, n$ where w_i and v_i are the ith columns of W and V.

(c) Compute c_i = the condition number of the ith eigenvalue $= 1/s_i$, $i = 1, 2, \ldots, n$.

(d) Perturb the $(n, 1)$th entry of A by $\epsilon = 10^{-5}, 10^{-7}, 10^{-10}$. Then compute the eigenvalues $\hat{\lambda}_i$, $i = 1, \ldots, n$ of the perturbed matrix using the MATLAB command **eig**.

(e) Make a table of the following form for each matrix. Also plot $|\lambda_i - \hat{\lambda}_i|$, $|\hat{\lambda}_i|$, $i = 1, \ldots, n$, for each perturbation.

λ_i	$\hat{\lambda}_i$	$\lvert\lambda_i - \hat{\lambda}_i\rvert$	Cond (V)	c_i

(f) Write your conclusions.

Test Data

1. A = the Wilkinson bidiagonal matrix of order 20
2. A = the transpose of the Wilkinson bidiagonal matrix of order 20
3. $$A = \begin{pmatrix} 12 & 11 & 10 & \cdots & \cdots & 2 & 1 \\ 11 & 11 & 10 & \cdots & \cdots & 2 & 1 \\ 0 & 10 & 10 & \cdots & \cdots & 2 & 1 \\ \vdots & \ddots & \ddots & \ddots & & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & \cdots & 0 & 1 & 1 \end{pmatrix}$$

8. Study the sensitivities of the eigenvectors of the following matrices by actually computing the eigenvectors of the original and perturbed matrices using the MATLAB command $[V, D] = eig(A)$, $[V, D] = eig(\hat{A})$, where $\hat{A}$ is the matrix obtained from A by perturbing $(n, 1)$th entry by $\epsilon = 10^{-5}, 10^{-7}, 10^{-10}$.

(a) $$A = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0.999 & 0 & 0 \\ 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0.0005 \end{pmatrix}$$

(b) $A = \text{diag}(1, 0.9999, 1, 0.9999, 1)$

(c) Randomly generated matrices of order 10, 15, 20, 25, 30, 50

9. Write a MATLAB program called **qritrb** to implement the basic QR iteration using **givqr** from Chapter 5:

(a) $[A] = \mathbf{qritrb}(A, N)$; N is the number of iterations.

(b) Modify the program to implement the Hessenberg QR iteration $[H] = \mathbf{qritrh}(H, N)$; N is the number of iterations.

(c) Compare the speed of the programs by actually computing the flop count and elapsed time on random matrices of order 10, 15, 20, 25, and 30.

10. (a) Write a MATLAB program to compute one step of the explicit double-shift QR iteration $[A] = \mathbf{qritrdse}(A, N)$, N is the number of iterations to be performed.

(b) Write a MATLAB program to compute one step of the implicit double-shift QR iteration $[A] = \mathbf{qritrdsi}(A, N)$, N is the number of iterations to be performed.

(c) Compare the results in (a) and (b) and conclude that they are essentially the same. Test your programs on random matrices of order 10, 15, 20, 25, 30, and 50.

11. Write a MATLAB program to deflate the last k rows and k columns of a Hessenberg matrix.

$$[\text{hprime}] = \mathbf{deflat}(H, k)$$

Test your program with a randomly generated matrix of order 20 with different values of k. Note that for $k = 1$, hprime will be of order $n - 1$, for $k = 2$ hprime will be of order $n - 2$, and so on.

12. Using **qritrdsi** and **deflat**, write a MATLAB program to determine the real Schur form of a Hessenberg matrix A,

$$[h] = \mathbf{rsf}(h, eps)$$

where eps is the tolerance.

Test: Generate randomly a 20×20 Hessenberg matrix and make the following table using **rsf**.

Iteration	h_{21}	h_{32}	h_{43}	h_{54}	$\cdots$	$\cdots$	$h_{20,19}$

13. (a) Write a MATLAB program called **polysymtri** to compute the characteristic polynomial p_i of the ith leading principal minor $(i = 1, 2, \ldots n)$ of an $n \times n$ unreduced symmetric tridiagonal matrix T, based on the recursion in

Section 8.11.1:

[valpoly] = **polysymtri**(T, *lambda*, i)

(b) Using **polysymtri**, write a MATLAB program called **signagree** that finds the number of eigenvalues of T greater than a given real number μ, based on Theorem 8.11.7:

[number] = **signagree**(T, *meu*)

(c) Using **polysmtri** and **signagree**, implement the bisection algorithm of Algorithm 8.11.1:

[lambda] = **bisection**(T, m, n)

Compute λ_{n-m+1} for $m = 1, 2, 3, \ldots$ using **bisection** and then compare your results with those obtained by using **eig**(T).

Test Data
T = the symmetric tridiagonal matrix arising in the buckling problem in Section 8.3.2, with $n = 20$.

14. Write a MATLAB program called **lanczossym** to implement the reformulated symmetric Lanczos algorithm of Section 8.11.3. Using **lanczossym**, find the first 10 eigenvalues of a randomly generated symmetric matrix of order 100. (To generate a symmetric matrix B, generate A first; then take $B = A + A^T$.) Plot the exact eigenvalue and the computed eigenvalues.

Note: The programs **power**, **invitr**, **qritrb**, **qritrh**, **qritrdsi**, etc., are in MATCOM, but it is a good idea to write your own codes.

15. (a) Write a MATLAB program **dc** to implement the divide-and-conquer method (Algorithm 8.11.3):

[D, V] = **dc**(T)

where D is a diagonal matrix containing the eigenvalues of the symmetric matrix T and V is the matrix of eigenvectors. Run your program with a randomly generated symmetric matrix T of order 100.

(b) Compute the eigenvalues and eigenvectors of T using MATLAB command **eig**.

[V_1, D_1] = **eig**(T)

(c) Compare the speed of the programs by computing timings of these programs.

Exercises 16 and 17 are on applications of the real Schur form (RSF).

16. *Solution of the Sylvester and Lyapunov matrix equations:* Consider the problem of solving the Sylvester matrix equation in control theory:

$$AX - XB = C$$

Here A, B, and C are given, and X is to be found. Assume A is of order $m \times m$ and B is of order $n \times n (m \geq n)$.

(The Lyapunov matrix equation $AX - XA^T = C$ is a special case.) This problem can be solved by transforming A to an upper Hessenberg matrix and B to real Schur form (RSF), as follows.

Step 1: The problem is first reduced to the equivalent problem

$$A'Y - YB' = C'$$

where A' is an upper Hessenberg matrix, and B' is in real Schur form.

Step 2: Y in the reduced problem is then found by solving a series of Hessenberg linear systems.

Step 3: The solution X of the original problem is finally retrieved from the solution Y of the reduced problem by orthogonal (or unitary) matrix multiplications.

Work out the details of the Steps 1 through 3 and then write a MATLAB program. Test your program with a randomly generated matrix of order 20 by using MATLAB command **lyap** (the method is based on the papers by Bartels and Stewart 1972, and Golub, Nash, and Van Loan 1979).

17. *Block Diagonalization:* Let $Q^T AQ = T$ be in real Schur form with $T_{11}, \ldots, T_{pp}$ as diagonal blocks with disjoint spectra. Develop an algorithm, using the Sylvester matrix equation, to find a matrix Y such that $Y^{-1}TY = \text{diag}(T_{11}, \ldots, T_{pp})$.

Write a MATLAB program and test your algorithm with a matrix of order 100.

Suggestions for Further Reading

Most books on vibration discuss eigenvalue problems arising in vibration of structures. However, almost all eigenvalue problems here are generalized eigenvalue problems; as a matter of fact, they are symmetric definite problems.

For references of the well-known books in vibration, see Section 9.10. The books by Inman (1994) and by Thomson (1993) are, in particular, very useful and important books in this area.

For learning more about how the eigenvalue problem arises in other areas of engineering, see the books on numerical methods in engineering by Chapra and Canale (1988) and by Peter O'Neil (1991). There are other engineering books (too numerous to list here), especially in the areas of electrical, mechanical, civil, and chemical engineering, containing discussions on eigenvalue problems in engineering. The real Schur form (RSF) and the Hessenberg forms of a matrix is an important tool in numerically effective solutions of many important control problems, such as solutions of the Lyapunov matrix equation (Bartels and Stewart, 1972), the Sylvester matrix equation (Golub, Nash, and Van Loan, 1979), algebraic Riccati equations (Laub, 1979, Van Dooren, 1981a, Byers, 1986, Ammar and Mehrmann, 1991, Mehrmann, 1988, and Bunse-Gerstner and Mehrmann, 1986). For details, see the book *Computational Methods for Linear Control Systems* by Petkov, Christov, and Konstantinov (1991), the IEEE reprint by Patel et al. (1994), and the book *Numerical Methods in Control Theory*, by B. N. Datta (in preparation) and the recent survey paper of Datta (1994).

Some references of how eigenvalue problems (especially large and sparse eigenvalue problems) arise in other areas of science and engineering such as power systems, physics, chemistry, structural mechanics, and oceanography are given in the book *Lanczos Algorithms for Large Symmetric Eigenvalue Computations*, vol. 1, by Jane Cullum and Ralph Willoughby (1985). See also the recent book by Saad (1992).

For some generalizations of the Geršgorin disk theorems, see the recent paper by Brualdi and Mallendorf (1994). This paper contains results giving a region of the complex plane for each eigenvalue; for a full description of the Geršgorin disk theorems and applications, see *Matrix Analysis* by Roger Horn and Charles Johnson (1985, Chapter 6). See also Varga (1962) and Johnston (1971).

A nice description of stability theory in dynamic systems is given in *Introduction to Dynamic Systems: Theory, Models, and Applications* by David Luenberger (1979).

For results on eigenvalue bounds, see the papers by Varah (1968b), Kahan, Parlett, and Jiang (1982), and the book by Stewart and Sun (1990).

For computation of Jordan canonical form, see the papers by Golub and Wilkinson (1976), Kågström and Ruhe (1980a and 1980b), and Demmel (1983).

Descriptions of the usual techniques for eigenvalue and eigenvector computations—the power method, the inverse power method, the Rayleigh quotient iteration method, and the QR iteration method—can be found in all numerical linear algebra books: Golub and Van Loan (1983 and 1989), Stewart (1973), Hager (1988), Watkins (1991), and Wilkinson (1965). The Wilkinson (1965) book is, of course, the most authoritative book in this area.

The papers by Varah (1968a, 1970) and Peters and Wilkinson (1979) are important in the context of inverse iteration.

A Fortran program for ordering the eigenvalues of a real upper Hessenberg matrix appears in Stewart (1976b). See also Bai and Demmel (1993a) and Dongarra, Hammarling, and Wilkinson (1992).

An important book in the area of symmetric eigenvalue problem is *The Symmetric Eigenvalue Problem* by Beresford Parlett (1980b).

For a proof of the global convergence of the symmetric QR iteration with Wilkinson shifts, see Lawson and Hanson (1974, 240–7) and Parlett (1980b).

For relationship between Power method and QR iterations, see Watkins (1982, 1993). These two papers, along with a paper of Nanda (1985), provide some theoretical insight into the convergence of the QR iteration algorithm.

Several important papers on the convergence and stability of the QR iteration algorithm are Parlett (1965, 1966, 1968) and Parlett and Le (1993).

A new approach in the theory of implicit shifting in the QR iteration algorithm applied to a Hessenberg matrix has been suggested recently by Miminis and Paige (1991). This approach is different from the one discussed in the book, namely the Francis' approach, and "seems to be more general in that it can be applied to block Hessenberg matrices of any form, as well as to many algorithms which differ from, but are in some ways like, the QR algorithm." See Miminis and Paige (1991) for details.

For computing the accurate eigensystems of scaled diagonally dominant matrices, see Barlow and Demmel (1990). For some recent results of interlacing, see Hill and Parlett (1993).

For a detailed description of the divide-and-conquer method, see Dongarra and Sorensen (1987) and the original paper of Cuppen (1981). For an efficient implementation of the divide-and-conquer method, see Gu and Eisenstat (1994a, 1994b). For a divide-and-conquer approach of the nonsymmetric eigenvalue problem, see Adams and Arbenz (1994). For error analysis of update methods for the symmetric eigenvalue problems, see Barlow (1993). For a method called Homotopy method for the symmetric eigenvalue problem, which is suitable for parallel implementation, see Zeng (1991). See also Li and Zeng (1992) and Li, Zeng, and Cong (1992). The books by Golub and Van Loan (1989), Parlett (1980), Watkins (1991), and Wilkinson (1965) contain descriptions of the Jacobi method. Many papers have been written on the Jacobi method, ranging from an earlier paper of Eberlein (1962) to one of the recent papers by Demmel and Veselic (1992). Golub and Van Loan (1989) contains a thorough list of references.

We have not discussed the procedure for estimating the condition numbers of eigenvalues and eigenvectors. See the papers of Bai, Demmel, and McKenney (1993), Boley (1990), Byers (1984), and Van Loan (1987). For eigenvalue refinement procedures, see Dongarra, Moler, and Wilkinson (1983) and Demmel (1987a).

A basic discussion on Lanczos methods with and without reorthogonalizations, and their applications to solutions of positive definite linear systems and eigenvalue computations, is contained in the book by Golub and Van Loan (1989, Chapter 9). Two authoritative books in this area are: *Lanczos Algorithms for Large Symmetric Eigenvalue Computations*, volumes I and II, by J. K. Cullum and R. A. Willoughby (1985); and *The Symmetric Eigenvalue Problem* by B. N. Parlett (1980b). Another recent important book in this area is *Numerical Methods for Large Eigenvalue Problems* by Y. Saad (1992). Since the pioneering work by Paige in the 1970s, many papers in this area have been published, and this is an active area of research. See Parlett and Scott (1979), Parlett, Simon, and Stringer (1982), Parlett (1989), Saad (1981), Ruhe (1994), Simon (1984), Calvetti, Reichel, and Sorensen (1994).

The doctoral thesis of Chris Paige (1971) and several of his follow-up papers (Paige 1976, 1980, etc.) are also well worth reading. These are considered to be the "seed papers" for further recent active research in this area.

Besides Lanczos algorithm, simultaneous iteration technique is another useful technique to compute a few eigenvalues and eigenvectors of a large and sparse matrix. See the papers by Jennings and Stewart (1975) and Stewart (1976c) in this context.

CHAPTER NINE

The Generalized Eigenvalue Problem

9.1 Introduction

In this chapter we consider the following eigenvalue problem known as the **generalized eigenvalue problem**.

> **Statement of the Generalized Eigenvalue Problem**
> Given $n \times n$ matrices A and B, find scalars λ and nonzero vectors x such that
>
> $$Ax = \lambda Bx$$

Note that the standard eigenvalue problem for the matrix A is a special case of this problem (take $B = I$).

DEFINITION 9.1.1 In the problem $Ax = \lambda Bx$, λ is called an **eigenvalue**, and the vector x is an **eigenvector** associated with λ for the generalized eigenvalue problem $Ax = \lambda Bx$.

It is easy to see that λ is a root of the characteristic equation

$$\det(A - \lambda B) = 0$$

DEFINITION 9.1.2 The matrix $A - \lambda B$ is called a **matrix pencil**. It is conveniently denoted by (A, B). The pair (A, B) is called **regular** if $\det(A - \lambda B)$ is not identically zero. Otherwise, it is called **singular**.

> **A Note on the Use of the Word *Pencil***
> "The rather strange use of the word *pencil* comes from optics and geometry. An aggregate of (light) rays converging to a point does suggest the sharp end of a pencil and, by a natural extension, the term came to be used for any one-parameter family of curves, spaces, matrices, or other mathematical objects." (Parlett 1980b, 302).

The Eigenvalues of a Regular Pencil

We shall now state some important but easily proved facts about the eigenvalues of a regular pencil.

Let (A, B) be a regular pencil. Then

Case 1. If B is nonsingular, then all the eigenvalues of the pair (A, B) are finite and are the same as those of AB^{-1} or of $B^{-1}A$.

Proof: Since B is nonsingular, from $Ax = \lambda Bx$ we have $B^{-1}Ax = \lambda x$.
Thus, λ is an eigenvalue of $B^{-1}A$. Also AB^{-1} and $B^{-1}A$ are similar, because, $AB^{-1} = BB^{-1}AB^{-1}$; therefore, they have the same eigenvalues. ■

Case 2. If B is singular, then the degree of $p(\lambda) = \det(A - \lambda B)$ is less than n. Let it be r. The r zeros of $p(\lambda)$ are the eigenvalues of the pair (A, B). The convention is to set the $(n - r)$ remaining eigenvalues to be ∞.

Example 9.1.1

$$A = \begin{pmatrix} 2 & 1 \\ 0 & 1 \end{pmatrix}, \qquad B = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

$\det(A - \lambda B) = 2 - \lambda$

The eigenvalues of (A, B) are 2 and ∞.

Example 9.1.2

$$A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}, \qquad B = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

Both the eigenvalues of (A, B) are ∞.

The Eigenvalues of a Singular Pencil

If (A, B) is a singular pencil, then, because $\det(A - \lambda B)$ vanishes identically, *any number λ can be an eigenvalue of (A, B)*. We will not consider the singular case here. Most generalized eigenvalue problems arising in practice are regular.

Many engineering applications give rise to generalized eigenvalue problems. *A majority of eigenvalue problems arising in mechanical vibration are generalized eigenvalue problems*. For example, the eigenproblems for vibrations of structures such as buildings and bridges are the so-called symmetric definite generalized eigenvalue problems for the **mass** and **stiffness** matrices: $Kx = \lambda Mx$, where M is the mass matrix

and K is the stiffness matrix. M and K are usually real symmetric and, furthermore, M is symmetric positive definite.

DEFINITION 9.1.3 If A and B are real symmetric matrices and, furthermore, if B is positive definite, then the generalized eigenvalue problem

$$Ax = \lambda Bx$$

is called the **symmetric definite generalized eigenvalue problem**.

This chapter is devoted to the study of numerical solution of the generalized eigenvalue problem, with particular attention to the symmetric definite problem (Sections 9.5 and 9.6). Applications of the symmetric definite generalized eigenvalue problem to the solution of second-order differential equations in mechanical vibration are studied in some depth (Sections 9.6 and 9.7). In addition, in Section 9.2 we discuss the generalized Schur decomposition and in Section 9.3 we cover the QZ algorithm. In Section 9.4 the inverse iteration for eigenvectors of the pencil is described.

The following major tools and concepts developed earlier will be needed for smooth learning and comprehension of this chapter's material: the Householder and Givens methods to create zeros in a vector and the corresponding QR factorization algorithms (Sections 5.4, 5.4.1, 5.4.2, 5.5, 5.5.1), the Cholesky algorithm for solving a symmetric positive definite system (Section 6.4.7), and the inverse iteration algorithm and the Rayleigh quotient iteration algorithm (Section 8.5).

9.2 Generalized Schur Decomposition

As a generalization of the procedure for the standard eigenvalue problem, it is natural to think of solving the generalized eigenvalue problem by simultaneously reducing the matrices A and B to some convenient forms from which generalized eigenvalues can be easily extracted. Theorem 9.3.1 shows the existence of such convenient forms, and Theorem 9.2.1 illustrates the extraction of the generalized eigenvalues once the matrices A and B have been reduced to such forms.

THEOREM 9.2.1 Let A and B be $n \times n$ matrices and the pair (A, B) be regular. Let U_1 and U_2 be unitary matrices such that

$$U_1AU_2 = T_1 = \begin{pmatrix} t_{11} & & \\ & \ddots & * \\ & & \ddots \\ & & & t_{nn} \end{pmatrix}$$

$$U_1BU_2 = T_2 = \begin{pmatrix} t'_{11} & & \\ & \ddots & * \\ & & \ddots \\ & & & t'_{nn} \end{pmatrix}$$

Then the generalized eigenvalues λ_i, $i = 1, \ldots, n$ of the regular pencil (A, B) are given by

$$\lambda_i = t_{ii}/t'_{ii}, \qquad \text{if } t'_{ii} \neq 0$$
$$\lambda_i = \infty, \qquad \text{if } t_{ii} \neq 0, t'_{ii} = 0$$

Proof: From $Ax = \lambda Bx$, we have

$$U_1AU_2U_2^*x = \lambda U_1BU_2U_2^*x$$

Define $y = U_2^*x$. Then from the preceding, we have

$$U_1AU_2y = \lambda U_1BU_2y$$

that is, $T_1y = \lambda T_2y$. Thus, λ is an eigenvalue of the pair (A, B) if and only if it is an eigenvalue of the pair (T_1, T_2). Because T_1 and T_2 are both triangular, $\det(T_1 - \lambda T_2) = \prod_{i=1}^{n}(t_{ii} - \lambda t'_{ii})$, and vanishes when $\lambda_i = t_{ii}/t'_{ii}$. Also, when $t_{ii} \neq 0$ but $t'_{ii} = 0$, λ_i can be set to be ∞. ■

Remarks:

1. Note that $x = U_2y$ is an eigenvector of $(A - \lambda B)$ associated with λ.
2. If A and B are real matrices, then U_1 and U_2 can be chosen such that $U_1AU_2 = T_1$ is in real Schur form.

9.3 The QZ Algorithm

We will describe an analog of the QR iteration, known as the QZ iteration algorithm for computing the eigenvalues, developed by Cleve Moler and G. W. Stewart. Assume that B is nonsingular. Then the basic idea is to apply the QR iteration algorithm to the matrix $C = B^{-1}A$ (or to AB^{-1}) without explicitly forming the matrix C, because if B is nearly singular, then it is not desirable to form B^{-1}. In this case the elements of C will be much larger than those of A and B, and the eigenvalues of C will be computed inaccurately.

If AB^{-1} or $B^{-1}A$ is not to be computed explicitly, then the next best alternative, of course, is to transform A and B to some reduced forms and then extract the generalized eigenvalues λ_i from these reduced forms. The simultaneous reduction of A and B to triangular forms by equivalence is guaranteed by the following theorem.

THEOREM 9.3.1 The Generalized Real Schur Decomposition Given two $n \times n$ real matrices A and B, there exist orthogonal matrices Q and Z such that $Q^TAZ = A'$ is an upper real Schur matrix and $Q^TBZ = B'$ is an upper triangular matrix:

$$A' = Q^TAZ: \quad A' = \text{an upper real Schur matrix}$$
$$B' = Q^TBZ: \quad B' = \text{an upper triangular matrix}$$

The pair (A', B') is said to be in **generalized real Schur form**. The pair (A', B') has the same eigenvalues as the pair (A, B).

We shall now give a constructive proof of this theorem. The reduction to the generalized real Schur form is achieved in two stages.

Stage I. A and B are reduced to an upper Hessenberg and an upper triangular matrix, respectively, by simultaneous orthogonal equivalence:

$$A \equiv Q^T AZ: \quad \text{an upper Hessenberg matrix}$$
$$B \equiv Q^T BZ: \quad \text{an upper triangular matrix}$$

Stage II. The Hessenberg-triangular pair (A, B) is reduced further to the **generalized real Schur form** by applying implicit QR iteration to AB^{-1}.

This process is known as the **QZ algorithm**.

We now briefly sketch these two stages. For details, see Moler and Stewart (1973). For notational convenience, we will denote a Givens rotation $J(i, j, \theta)$ by J_{ij} throughout the chapter.

9.3.1 Reduction to Hessenberg-Triangular Form

Let A and B be two $n \times n$ matrices. Then

1. Find an orthogonal matrix U such that

$$B \equiv U^T B: \quad \text{upper triangular}$$

Form

$$A \equiv U^T A$$

(in general, A will be full).

2. Reduce A to Hessenberg form while preserving the triangular structure of B.

This step is achieved as follows. To start with, we have

$$A \equiv U^T A = \begin{pmatrix} * & * & \cdots & * \\ * & * & \cdots & * \\ \vdots & \vdots & & \vdots \\ * & * & \cdots & * \\ * & * & \cdots & * \end{pmatrix}$$

$$B \equiv U^T B = \begin{pmatrix} * & * & \cdots & \cdots & * \\ 0 & * & \cdots & \cdots & * \\ 0 & 0 & \ddots & \cdots & * \\ \vdots & \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots 0 & * \end{pmatrix}$$

First, the $(n, 1)$th entry of A is made zero by applying a rotation $Q_{n-1,n}$ in the $(n-1, n)$ plane:

$$A \equiv Q_{n-1,n}A = \begin{pmatrix} * & * & \cdots & * \\ * & * & \cdots & * \\ \vdots & \vdots & & \vdots \\ * & * & \cdots & * \\ 0 & * & \cdots & * \end{pmatrix}$$

This transformation, when applied to B to the left, will fill in the $(n, n-1)$ position:

$$B \equiv Q_{n-1,n}B = \begin{pmatrix} * & * & \cdots & \cdots & * \\ 0 & * & \cdots & \cdots & * \\ 0 & 0 & * & \cdots & * \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & * & * \end{pmatrix}$$

The rotation $Z_{n-1,n}$ is now applied to the right of B to make the $(n, n-1)$ entry of B zero. Fortunately, this rotation, when applied to the right of A, does not destroy the zero produced earlier. Schematically, we have

$$B \equiv B\, Z_{n-1,n} = \begin{pmatrix} * & * & * & \cdots & * \\ 0 & * & * & \cdots & * \\ 0 & 0 & * & \cdots & * \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & * \end{pmatrix}$$

$$A \equiv AZ_{n-1,n} = \begin{pmatrix} * & * & * & \cdots & * \\ * & * & * & \cdots & * \\ * & * & * & \cdots & * \\ \vdots & \vdots & \vdots & & \vdots \\ * & * & * & \cdots & * \\ 0 & * & * & \cdots & * \end{pmatrix}$$

The entries $(n-1, 1), (n-2, 1), \ldots, (3, 1)$ of A are now successively made zero. For each entry to be made zero, three rotations are to be applied in succession, as follows. First, an appropriate rotation is applied to the left of A to make the entry zero. Second, the same rotation is applied to the left of B. Finally, another appropriate rotation is applied to the right of B to recover the undesirable fill-in in B. At the end of the first step, the matrix A is Hessenberg in its first column, whereas B remains upper triangular:

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

The zeros are now produced on the second column of A in the appropriate places while retaining the triangular structure of B in the analogous manner.

The process is continued until the matrix A is an upper Hessenberg matrix and B is upper triangular.

Example 9.3.1

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 3 & 4 \\ 1 & 3 & 3 \end{pmatrix}, \qquad B = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & 2 \end{pmatrix}$$

Step 1: Form Q_{23} to make a_{31} zero:

$$Q_{23} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0.7071 & 0.7071 \\ 0 & -0.7071 & 0.7071 \end{pmatrix}$$

$$A \equiv A^{(1)} = Q_{23}A = \begin{pmatrix} 1 & 2 & 3 \\ 1.4142 & 4.2426 & 4.9497 \\ 0 & 0 & -0.7071 \end{pmatrix}$$

Step 2: Update B:

$$B \equiv B^{(1)} = Q_{23}B = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0.7071 & 2.8284 \\ 0 & -0.7071 & 0 \end{pmatrix}$$

Step 3: Form Z_{23} to make b_{32} zero:

$$Z_{23} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix}$$

$$B \equiv B^{(1)}Z_{23} = Q_{23}BZ_{23} = \begin{pmatrix} 1 & 1 & -1 \\ 0 & 2.8284 & -0.7071 \\ 0 & 0 & 0.7071 \end{pmatrix}$$

Step 4: Update A:

$$A \equiv A^{(1)}Z_{23} = Q_{23}AZ_{23} = \begin{pmatrix} 1 & 3 & -2 \\ 1.4142 & 4.9497 & -4.2426 \\ 0 & -0.7071 & 0 \end{pmatrix}$$

A is in upper Hessenberg and B is in upper triangular form.

9.3.2 Reduction to the Generalized Schur Form

At the beginning of this process, we have A and B as an upper Hessenberg and an upper triangular matrix, respectively, obtained from stage 1. We can assume without loss of generality that the matrix A is an unreduced upper Hessenberg matrix. *The basic idea now is to apply an implicit QR step to AB^{-1} without ever completely forming this matrix explicitly.*

Thus a QZ step, analogous to an implicit QR step (see Algorithm 8.9.1), will be as follows:

- Compute the first column of $N = (C - \alpha_1 I)(C - \alpha_2 I)$, where $C = AB^{-1}$ and α_1 and α_2 are suitably chosen shifts.
- Find a Householder matrix Q_1 such that $Q_1 N e_1$ is a multiple of e_1.
- Form $Q_1 A$ and $Q_1 B$.
- Simultaneously transform $Q_1 A$ to an upper Hessenberg matrix A_1, and $Q_1 B$ to an upper triangular matrix B_1:

$$A_1 \equiv Q'(Q_1 A)Z: \quad \text{an upper Hessenberg}$$
$$B_1 \equiv Q'(Q_1 B)Z: \quad \text{an upper triangular}$$

Using the implicit Q theorem (Theorem 5.5.3) we can show (Exercise 5) that the matrix $A_1 B_1^{-1}$ is essentially the same as that we would have obtained by applying an implicit QR step directly to AB^{-1}.

Application of a few QZ steps sequentially will then yield a quasi-triangular matrix $R = Q^T AZ$ and an upper triangular $T = Q^T BZ$, from which the eigenvalues can be easily computed. The preceding four steps constitute one *single iteration* of the QZ algorithm. We now show how to implement these steps.

Computation of the First Column of $(C - \alpha_1 I)(C - \alpha_2 I)$

The real bottleneck in implementing the whole algorithm is in computing the first column of $(C - \alpha_1 I)(C - \alpha_2 I)$ without forming $C = AB^{-1}$ explicitly. Fortunately, this can be done. First, note that because A is upper Hessenberg and B is upper triangular, this first column contains at most three nonzero entries in the first three places:

$$n_1 = Ne_1 = (C - \alpha_1 I)(C - \alpha_2 I)e_1 = \begin{pmatrix} x \\ y \\ z \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

To compute x, y, z, all we need to know is the first two columns of C, which can be obtained just by inverting the 2×2 leading principal submatrix of B^{-1}; *the whole B^{-1} does not need to be computed* (Exercise 3). Thus, if c_1 and c_2 are the first two columns of $C = AB^{-1}$, then

$$(c_1, c_2) = (a_1, a_2)\begin{pmatrix} b_{11} & b_{12} \\ 0 & b_{22} \end{pmatrix}^{-1}, \text{ where } a_1 \text{ and } a_2 \text{ are the first two columns of } A.$$

Computations of A_1 and B_1

Since the first column n_1 of $N = (C - \alpha_1 I)(C - \alpha_2 I)$ has at most three nonzero entries, the Householder matrix Q_1 that transforms n_1 to a multiple of e_1 has the form

$$Q_1 = \begin{pmatrix} \hat{Q}_1 & 0 \\ 0 & I_{n-3} \end{pmatrix}$$

where $\hat{Q}_1$ is a 3×3 Householder matrix. The matrices Q_1A and Q_2B will then have the following forms, as illustrated here with $n = 6$. $\otimes$ denotes a possible fill-in.

$$A \equiv Q_1A = \begin{pmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \\ \otimes & * & * & * & * & * \\ 0 & 0 & * & * & * & * \\ 0 & 0 & 0 & * & * & * \\ 0 & 0 & 0 & 0 & * & * \end{pmatrix}$$

$$B \equiv Q_1B = \begin{pmatrix} * & * & * & * & * & * \\ \otimes & * & * & * & * & * \\ \otimes & \otimes & * & * & * & * \\ 0 & 0 & 0 & * & * & * \\ 0 & 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & 0 & 0 & * \end{pmatrix}$$

That is, both the Hessenberg form A and the triangular form of B are now lost, in that there are now both an unwanted nonzero in the (3, 1) position of Q_1A and the unwanted nonzeros in the (2, 1), (3, 1), and (3, 2) positions of Q_1B. The job now will be to cleverly chase the unwanted nonzero entries and make them zero using orthogonal transformations.

To do this, a Householder matrix Z_1 is first constructed to create zeros in the positions (3, 1) and (3, 2) of B, and then a Givens rotation Z_2 is constructed to create a zero in (2, 1) position of B. A is next updated.

$$B \equiv Q_1BZ_1Z_2 = \begin{pmatrix} * & * & * & * & * & * \\ 0 & * & * & * & * & * \\ 0 & 0 & * & * & * & * \\ 0 & 0 & 0 & * & * & * \\ 0 & 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & 0 & 0 & * \end{pmatrix}$$

$$A \equiv Q_1AZ_1Z_2 = \begin{pmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \\ \otimes & * & * & * & * & * \\ \otimes & \otimes & * & * & * & * \\ 0 & 0 & 0 & * & * & * \\ 0 & 0 & 0 & 0 & * & * \end{pmatrix}$$

(Note that we now have unwanted zeros in the (3, 1), (4, 1), and (4, 2) positions of A.)

Next, a Householder matrix Q_2 is created to introduce zeros in the (3, 1) and (4, 1) positions of A; B is then updated. We now have

$$A \equiv Q_2Q_1AZ_1Z_2 = \left(\begin{array}{c|ccccc} * & * & * & * & * & * \\ \hline * & * & * & * & * & * \\ 0 & * & * & * & * & * \\ 0 & * & * & * & * & * \\ 0 & 0 & 0 & * & * & * \\ 0 & 0 & 0 & 0 & * & * \end{array}\right)$$

$$B \equiv Q_2Q_1BZ_1Z_2 = \left(\begin{array}{c|ccccc} * & * & * & * & * & * \\ \hline 0 & * & * & * & * & * \\ 0 & * & * & * & * & * \\ 0 & * & * & * & * & * \\ 0 & 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & 0 & 0 & * \end{array}\right)$$

The submatrices of A and B as just shown now have the same structures as those of the original matrices, namely, of the matrices Q_1A and Q_1B. The reduction process is now continued with these submatrices until the Hessenberg triangular structure of (A, B) is restored.

In view of this discussion, let's now summarize one iteration of the QZ algorithm.

ALGORITHM 9.3.1 One iteration step of the QZ Algorithm Given A, an unreduced upper Hessenberg matrix, and B, an upper triangular matrix, the following steps constitute one iteration step of the QZ algorithm; that is, these steps construct orthogonal matrices Q and Z such that Q^TAZ is an upper Hessenberg matrix and Q^TBZ is upper triangular.

Step 1: Choose the shifts α_1 and α_2.

Step 2: Compute the first column of $N = (C - \alpha_1 I)(C - \alpha_2 I)$, where $C = AB^{-1}$, without explicitly forming B^{-1}: Let (c_1, c_2) be the first two columns of C. Then

$$(c_1, c_2) = (a_1, a_2)\begin{pmatrix} b_{11} & b_{12} \\ 0 & b_{22} \end{pmatrix}^{-1}$$

The three nonzero entries of the first column of N are given by:

$$x = (c_{11} - \alpha_1)(c_{11} - \alpha_2) + c_{12}c_{21}$$
$$y = c_{21}(c_{11} - \alpha_2) + c_{21}(c_{22} - \alpha_1)$$
$$z = c_{21}c_{32}$$

The first column of N is $n_1 = \begin{pmatrix} x \\ y \\ z \\ 0 \\ \vdots \\ 0 \end{pmatrix}$

Step 3: Find a Householder matrix Q_1 such that $Q_1n_1 = \begin{pmatrix} * \\ 0 \\ \vdots \\ 0 \end{pmatrix}$

Step 4: Form Q_1A and Q_1B

Step 5: Transform Q_1A and Q_1B, respectively, to an upper Hessenberg matrix A_1 and a triangular matrix B_1 by orthogonal equivalence in the way shown

previously, by taking advantage of the special structures of the matrices, creating orthogonal matrices Q_2 through Q_{n-1} and Z_1 through Z_{n-1}.

Obtaining the Transforming Matrices

The transforming matrices Q and Z are obtained as follows. The matrix Q:

$$Q = Q_1 Q_2 Q_3 \cdots Q_{n-1}$$

The matrix Z:

$$Z = Z_1 Z_2 \cdots Z_{n-1}$$

Note that Q has the same first row as Q_1.

Flop count. One QZ step requires about $13n^2$ flops. If Q and Z are to be accumulated, then an additional $5n^2$ and $8n^2$ flops, respectively, are required.

Choosing the Shifts

The double shifts α_1 and α_2 at a QZ step can be taken as the eigenvalues of the lower 2×2 submatrix of $C = AB^{-1}$. *The 2×2 lower submatrix of C can be computed without explicitly forming B^{-1}* (Exercise 4).

ALGORITHM 9.3.2 The QZ Algorithm Given $n \times n$ real matrices A and B.

Step 1: Transform (A, B) to a Hessenberg triangular pair by orthogonal equivalence:

$A \equiv Q^T AZ$: an upper Hessenberg

$B \equiv Q^T BZ$: an upper triangular

Step 2: Iterate with Algorithm 9.3.1 to produce $\{A_k\}$ and $\{B_k\}$, choosing shifts for each iteration as described above.

Step 3: Monitor the convergence of the sequences $\{A_k\}$ and $\{B_k\}$:

$\{A_k\} \to R$, quasi-triangular

$\{B_k\} \to T$, triangular

Flop count. The implementation of Step 1 to 3 requires about $15n^3$ flops. The formations of Q and Z, if required, need, respectively, another $8n^3$ and $10n^3$ flops (from experience it is known that about two QZ steps per eigenvalue are adequate with the Francis shift strategy).

Roundoff properties. The QZ iteration algorithm is as stable as the QR iteration algorithm. It can be shown that the computed $\hat{R}$ and $\hat{S}$ satisfy

$$Q_0^T(A + E)Z_0 = \hat{R}$$
$$Q_0^T(B + F)Z_0 = \hat{S}$$

where Q_0 and Z_0 are orthogonal and

$$\|E\| \cong \mu\|A\|$$
$$\text{and} \quad \|F\| \cong \mu\|B\|$$

μ is the machine precision.

9.4 The Inverse Iteration for Eigenvectors of the Pencil (A, B)

Once an approximate eigenvalue λ is computed, the corresponding eigenvector v can be computed using the inverse iteration as before.

ALGORITHM 9.4.1

Step 1: Choose an initial eigenvector v_0.

Step 2: For $k = 1, 2, \ldots$ do until convergence

Solve $(A - \lambda B)\hat{v}_k = Bv_{k-1}$

$v_k = \hat{v}_k / \|\hat{v}_k\|_2$

MATCOM Note: Algorithm 9.4.1 has been implemented in MATCOM program INVITRGN.

A Remark on Solving $(A - \lambda B)\hat{v}_k = Bv_{k-1}$

In solving $(A - \lambda B)\hat{v}_k = Bv_{k-1}$, substantial savings can be made by exploiting the Hessenberg-triangular structure to which the pair (A, B) is reduced as a part of the QZ algorithm. Note that for a given λ, the matrix $A - \lambda B$ is a Hessenberg matrix. Thus, at each iteration only a Hessenberg system needs to be solved.

Example 9.4.1

$$A = \begin{pmatrix} 3 & -1.5 & 0 \\ -1.5 & 3 & -1.5 \\ 0 & -1.5 & 1.5 \end{pmatrix}$$

$$B = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 4 \end{pmatrix}$$

λ_1 = a generalized eigenvalue of $(A - \lambda B)$

$= 1.9508$

$$v_0 = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

$k = 1$: Solve for v_1:

$$(A - \lambda_1 B)\hat{v}_1 = Bv_0$$

$$\hat{v}_1 = 10^4 \begin{pmatrix} 1.7027 \\ -1.0236 \\ 0.2435 \end{pmatrix}$$

$$v_1 = \hat{v}_1 / \|\hat{v}_1\| = \begin{pmatrix} 0.8507 \\ -0.5114 \\ 0.1217 \end{pmatrix}$$

9.5 The Symmetric Definite Generalized Eigenvalue Problem

Many practical applications give rise to symmetric definite generalized eigenvalue problems of the form

$$Ax = \lambda Bx$$

and

$$ABx = \lambda x$$

where A and B are symmetric, and at least one of them is positive definite. We will assume that B is positive definite and consider only the problem $Ax = \lambda Bx$. When B is positive definite, we can write

$$ABx = \lambda x$$

in the form

$$(A - \lambda B^{-1})(Bx) = 0$$

Since B^{-1} is also positive definite, the problem $ABx = \lambda x$ is of the same form as the problem $Ax = \lambda Bx$.

The symmetric definite problem $Ax = \lambda Bx$ is routinely solved in vibration analysis of structures.

> **Frequencies, Modes, and Mode Shapes**
> The eigenvalues are related to the natural frequencies, and "the size and sign of each element of an eigenvector determines the shape of the vibration at any instant of time." The eigenvectors are, therefore, referred to as mode shapes, or simply as modes.
>
> "The language of modes, mode shapes, and natural frequencies forms the basis for discussing vibration phenomena of complex systems. An entire industry has been formed around the concept of modes" (Inman 1994, 196).

We start with an important (but not surprising) property of the symmetric definite pencil.

THEOREM 9.5.1 The symmetric definite pencil $(A - \lambda B)$ has real eigenvalues and linearly independent eigenvectors.

Proof: Since B is symmetric positive definite, it admits the Cholesky decomposition:

$$B = LL^T$$

So, from

$$Ax = \lambda Bx$$

we have

$$Ax = \lambda LL^T x$$

which gives

$$L^{-1}A(L^T)^{-1}L^T x = \lambda L^T x$$

or

$$Cy = \lambda y, \qquad \text{where } y = L^T x$$

The matrix $C = L^{-1}A(L^T)^{-1}$ is *symmetric*; therefore, λ is real. The assertion about the eigenvectors is obvious, because a symmetric matrix has a set of n independent eigenvectors (Theorem 8.2.8). ■

> **An Interval Containing the Eigenvalues of a Symmetric Definite Pencil**
> The eigenvalues of the symmetric definite pencil $A - \lambda B$ lie in the interval $[-\|B^{-1}A\|, \|B^{-1}A\|]$ (see Exercise 26).

Orthogonality of the Eigenvectors

If the eigenvectors $y_1, y_2, \ldots, y_n$ of C are chosen to be orthonormal, then it is easy to see that the eigenvectors x_i are B-orthonormal; that is

$$x_i^T Bx_i = 1, \qquad i = 1, \ldots, n$$
$$x_i^T Bx_j = 0, \qquad i \neq j$$

Moreover, if x_i is a normalized eigenvector, $x_i^T Ax_i = c_{ii}$, $i = 1, \ldots, n$ and

$$x_i^T Ax_j = 0, \qquad i \neq j$$

To prove $x_i^T Bx_j = 0$, we note that

$$x_i^T Bx_j = x_i^T LL^T x_j = (L^T x_i)^T L^T x_j = y_i^T y_j = 0$$

(since $y_1, \ldots, y_n$ are orthogonal). The other relations can similarly be proved.

Computational Algorithms

9.5.1 The QZ Method for the Symmetric Definite Pencil

The QZ algorithm described in the previous section for the regular pencil $A - \lambda B$ can, of course, be applied to a symmetric definite pencil. However, the drawback here is that both the symmetry and definiteness of the problem will be lost, in general. We describe now a specialized algorithm for a symmetric definite pencil.

9.5.2 The Cholesky QR Algorithm

The proof of Theorem 9.5.1 immediately reveals a computational method for finding the generalized eigenvalues of a symmetric definite pencil.

ALGORITHM 9.5.1 The Cholesky QR Algorithm for the Symmetric Definite Pencil The following algorithm computes the eigenvalues λ_i and the corresponding eigenvectors x_i, $i = 1, 2, \ldots, n$ of the symmetric definite pencil (A, B).

Step 1: Find the Cholesky factorization of B, using Algorithm 6.4.4:

$$B = LL^T$$

Step 2: Form $C = L^{-1}A(L^T)^{-1}$ by taking advantage of the symmetry of A.

Step 3: Compute the eigenvalues λ_i, $i = 1, 2, \ldots, n$ and the orthonormal eigenvectors y_i, $i = 1, \ldots, n$, of the symmetric matrix C, using the symmetric QR iteration algorithm (Algorithms 8.11.2 and 8.10.1). [$\lambda_1, \lambda_2, \ldots, \lambda_n$ are the eigenvalues of the pair (A, B)].

Step 4: Compute the eigenvectors x_i corresponding to the eigenvalue λ_i of the pencil (A, B) by solving: $L^T x_i = y_i$, $i = 1, \ldots, n$.

MATCOM Note: Algorithm 9.5.1 has been implemented in MATCOM program CHOLQR.

Stability of the Algorithm

When B is well conditioned, the algorithm is stable. However, if B is ill conditioned or nearly singular, so is L^{-1}; then the matrix C cannot be computed accurately. Therefore, in this case, the eigenvalues and the eigenvectors will be inaccurate.

Specifically, it can be shown (Golub and Van Loan MC 1989, 469) that a computed eigenvalue $\tilde{\lambda}$ obtained by the algorithm is the exact eigenvalue of the matrix $(L^{-1}A(L^T)^{-1} + E)$, where $\|E\|_2 \approx \mu\|A\|_2\|B^{-1}\|_2$. Thus, ill-conditioning of B will severely affect the computed eigenvalues, even if the eigenvalues are themselves well conditioned.

As we will see in the next section, in many practical applications only a few of the smallest generalized eigenvalues are of interest. These eigenvalues can sometimes be computed reasonably accurately, even when B is ill conditioned, by using the ordered RSF of B in place of the Cholesky decomposition, as follows.

ALGORITHM 9.5.2 The Schur QR Algorithm for the Symmetric Definite Pencil The following algorithm computes the eigenvalues and the corresponding eigenvectors of a symmetric definite pencil (A, B) using the ordered real Schur form of B. The smallest eigenvalues are expected to be accurately determined.

Step 1: Compute the real Schur form of B and order the eigenvalues of B from smallest to largest:

$$U^T BU = D = \text{diag}(d_1, \ldots, d_n)$$

Step 2: Form

$$L = UD^{\frac{1}{2}} = U\,\text{diag}\left(\sqrt{d_1}, \ldots, \sqrt{d_n}\right)$$

Step 3: Form

$$C = L^{-1}A(L^T)^{-1}$$

Step 4: Compute the eigenvalues and the eigenvectors z_i of C using symmetric QR iteration. [The eigenvalues of C are the eigenvalues of the pencil (A, B)].

Step 5: Compute the eigenvectors x_i of (A, B) by solving for x_i:

$$D^{\frac{1}{2}}U^T x_i = z_i, \qquad i = 1, 2, \ldots, n$$

MATCOM Note: Algorithm 9.5.2 has been implemented in MATCOM program SCHURQR.

Example 9.5.1 A Comparison Between Algorithms 9.5.1 and 9.5.2

Let

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \end{pmatrix}, \qquad B = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0.00001 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$\text{Cond}(B) = 10^5$.

Let

$$B_1 = \begin{pmatrix} 10^{-8} & 10^{-4} & 10^{-7} \\ 10^{-4} & 1.00000001 & 0.0011 \\ 10^{-7} & 0.0011 & 2.000001 \end{pmatrix}$$

$\text{Cond}(B_1) = O(10^{16})$.

Performing all the computations using 16-digit arithmetic, we find that all three eigenvalues of (A, B) were computed accurately by both the algorithms. However, with the pair (A, B_1) it was a different story.

Algorithm 9.5.2 computed the two smallest eigenvalues of (A, B_1), namely, -2.998800250010 and 0, accurately (up to the full precision); on the other hand, Algorithm 9.5.1 gave -2.81799699955530 and 0.09960728145265 as the two computed smallest eigenvalues. The reason, of course, is that B_1 is severely ill-conditioned.

9.5.3 Diagonalization of the Symmetric Definite Pencil

Simultaneous Diagonalization of *A* and *B*

The Cholesky QR iteration algorithm of the symmetric definite pencil gives us a method to find a nonsingular matrix X that transforms A and B simultaneously to diagonal forms by congruence. In fact, *X is a matrix of eigenvectors of the definite pencil* (A, B). This can be seen as follows.

From Step 4 of Algorithm 9.5.1, we have

$$x_i = (L^T)^{-1}y_i, \qquad i = 1, \ldots, n$$

Let $X = (x_1, x_2, \ldots, x_n)$, and $Y = (y_1, y_2, \ldots, y_n)$.
Then $X^T AX = Y^T L^{-1}A(L^{-1})^T Y = Y^T CY = \text{diag}(c_1, c_2, \ldots, c_n) = D$.
And $X^T BX = Y^T L^{-1}B(L^{-1})^T Y = Y^T L^{-1}(LL^T)(L^{-1})^T Y = Y^T Y = I$.
Finally, from the preceding two equations, we have $AX = (X^T)^{-1}Y^T CY = BXD$.

We can now *restate* Algorithm 9.5.1 in the context of simultaneous diagonalization of A and B.

ALGORITHM 9.5.3 Simultaneous Diagonalization of a Symmetric Definite Pencil Given a symmetric matrix A and a symmetric positive definite matrix B, the following algorithm computes a nonsingular matrix X such that $X^T BX$ is the identity matrix and $X^T AX$ is a diagonal matrix.

Step 1: Compute the Cholesky factorization of B (Algorithm 6.4.4):

$$B = LL^T$$

Step 2: Form $C = L^{-1}A(L^T)^{-1}$ by taking advantage of the symmetry of A (C is symmetric).

Step 3: Applying the symmetric QR iteration algorithm (Algorithm 8.11.2) to C, find an orthogonal matrix Y such that $Y^T CY$ is a diagonal matrix.

Step 4: Form $X = (L^{-1})^T Y$

Flop count. The algorithm for simultaneous diagonalization requires $7n^3$ flops.

MATCOM Note: Algorithm 9.5.3 has been implemented in MATCOM program SIMDIAG.

Example 9.5.2 Consider

$$B = \begin{pmatrix} 10 & 1 & 1 \\ 1 & 10 & 1 \\ 1 & 1 & 10 \end{pmatrix}, \qquad A = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \end{pmatrix}$$

A is symmetric and B is symmetric positive definite.

1. The Cholesky decomposition of $B = LL^T$:

$$L = \begin{pmatrix} 3.1623 & 0 & 0 \\ 0.3162 & 3.1464 & 0 \\ 0.3162 & 0.2860 & 3.1334 \end{pmatrix}$$

2. Form $C = L^{-1}A(L^{-1})^T$:

$$C = \begin{pmatrix} 0.1000 & 0.1910 & 0.2752 \\ 0.1910 & 0.2636 & 0.3320 \\ 0.2752 & 0.3320 & 0.3864 \end{pmatrix}$$

3. Find an orthogonal Y such that

$$Y^T CY = \text{diag}(c_1, \ldots, c_n)$$

$$Y = \begin{pmatrix} 0.4220 & -0.8197 & -0.3873 \\ 0.5684 & -0.0936 & 0.8174 \\ 0.7063 & 0.5651 & -0.4262 \end{pmatrix}$$

4. Form

$$X = (L^{-1})^T Y = \begin{pmatrix} 0.09409 & -0.2726 & -0.1361 \\ 0.1601 & -0.0462 & 0.2722 \\ 0.2254 & 0.1803 & -0.1361 \end{pmatrix}$$

5. Verify

$$X^T AX = \text{diag}(0.8179, -0.0679, 0)$$
$$X^T BX = \text{Identity} = \text{diag}(1, 1, 1)$$

Modal Matrix and Normal Modes

In vibration problems involving the mass and stiffness matrices—that is, when K is the stiffness matrix and M is the mass matrix in the symmetric definite generalized eigenvalue problem

$$Kx = \lambda Mx$$

the matrix P that simultaneously diagonalizes M and K, respectively, to the identity and a diagonal matrix, is called the **modal matrix**, and the columns of the matrix P are called the **normal modes**.

Generalized Rayleigh Quotient

The Rayleigh quotient defined for a symmetric matrix A in Chapter 8 (Section 8.5.3) can easily be generalized for the symmetric definite pair (A, B).

DEFINITION 9.5.1 The number

$$\lambda = \frac{x^T Ax}{x^T Bx} \qquad (\|x\|_2 = 1)$$

is called the generalized Rayleigh quotient. It can be shown that $\lambda_n \le \frac{x^T Ax}{x^T Bx} \le \lambda_1$, where λ_n and λ_1 are, respectively, the smallest and largest eigenvalue of (A, B).

It can be used to compute approximations to eigenvalues λ_k and eigenvectors x_k for the symmetric definite generalized eigenvalue problem, as shown in the following algorithm.

ALGORITHM 9.5.4 Generalized Rayleigh Quotient Iteration

Step 1: Choose x_0 such that $\|x_0\| = 1$.

Step 2: For $k = 0, 1, \ldots$ do until convergence occurs

a: Compute

$$\lambda_k = \frac{x_k^T Ax_k}{x_k^T Bx_k}$$

b: Solve for $\hat{x}_{k+1}$:

$$(A - \lambda_k B)\hat{x}_{k+1} = Bx_k$$

c: Normalize $\hat{x}_{k+1}$:

$$x_{k+1} = \frac{\hat{x}_{k+1}}{\|\hat{x}_{k+1}\|_2}$$

MATCOM Note: Algorithm 9.5.4 has been implemented in MATCOM program GENRAYQT.

9.6 Symmetric Definite Generalized Eigenvalue Problems Arising in Vibrations of Structures

As we stated earlier, almost all eigenvalue problems arising in vibrations of structures are symmetric definite generalized eigenvalue problems. In Chapter 8 (Section 8.3) we have given some examples. In this section we will give some more.

9.6.1 A Case Study: The Vibration of a Spring-Mass System

Consider the system of three spring masses shown in Figure 9.1 with spring constants k_1, k_2, and k_3. (The stiffness matrices are very often determined using physical characteristics of the systems. The eigenvalues and eigenvectors are then used to study the behavior of the system.) We would like to study the configuration of the system corresponding to the different modes of vibration.

1: Formulate the Problem as a Symmetric Definite Generalized Eigenvalue Problem

Let the displacements of the masses from static equilibrium positions be defined by the generalized coordinates y_1, y_2, and y_3, respectively.

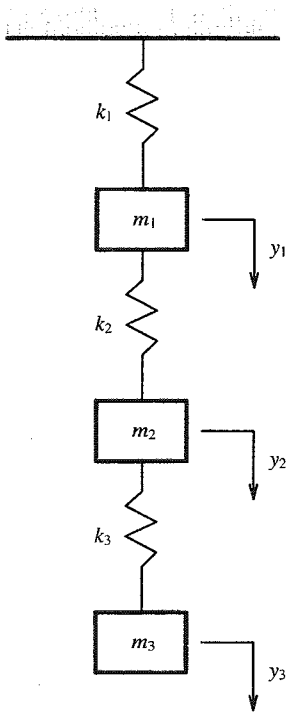


FIGURE 9.1 Three Degrees of Freedom

The equations of motion for the system can be written as

$$\begin{aligned} m_1\ddot{y}_1 + (k_1 + k_2)y_1 - k_2y_2 &= 0 \\ m_2\ddot{y}_2 - k_2y_1 + (k_2 + k_3)y_2 - k_3y_3 &= 0 \\ m_3\ddot{y}_3 - k_3y_2 + k_3y_3 &= 0 \end{aligned}$$

or, in matrix form,

$$\begin{pmatrix} m_1 & 0 & 0 \\ 0 & m_2 & 0 \\ 0 & 0 & m_3 \end{pmatrix}\begin{pmatrix} \ddot{y}_1 \\ \ddot{y}_2 \\ \ddot{y}_3 \end{pmatrix} + \begin{pmatrix} k_1 + k_2 & -k_2 & 0 \\ -k_2 & k_2 + k_3 & -k_3 \\ 0 & -k_3 & k_3 \end{pmatrix}\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

that is,

$$M\ddot{y} + Ky = 0$$

where $M = \text{diag}(m_1, m_2, m_3)$ and

$$K = \begin{pmatrix} k_1 + k_2 & -k_2 & 0 \\ -k_2 & k_2 + k_3 & -k_3 \\ 0 & -k_3 & k_3 \end{pmatrix}$$

Assuming harmonic motion, we can write

$$\begin{aligned} y_1 &= x_1\, e^{i\omega t} \\ y_2 &= x_2\, e^{i\omega t} \\ y_3 &= x_3\, e^{i\omega t} \end{aligned}$$

where x_1, x_2, and x_3 are, respectively, the *amplitudes* of the masses, m_1, m_2, and m_3 and ω denotes the *natural frequency*. Substituting these expressions for y_1, y_2, and y_3 into the equations of motion and noting that

$$\ddot{y}_k = -\omega^2 x_k e^{i\omega t}, \qquad k = 1, 2, 3$$

we have

$$\begin{aligned} -m_1x_1\omega^2 + (k_1 + k_2)x_1 - k_2x_2 &= 0 \\ -m_2x_2\omega^2 - k_2x_1 + (k_2 + k_3)x_2 - k_3x_3 &= 0 \\ -m_3x_3\omega^2 - k_3x_2 + k_3x_3 &= 0 \end{aligned}$$

which can be written in matrix form as

$$\begin{pmatrix} k_1 + k_2 & -k_2 & 0 \\ -k_2 & k_2 + k_3 & -k_3 \\ 0 & -k_3 & k_3 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \omega^2 \begin{pmatrix} m_1 & 0 & 0 \\ 0 & m_2 & 0 \\ 0 & 0 & m_3 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

or

$$Kx = \lambda Mx$$

where $\lambda = \omega^2$. *The eigenvalues* $\lambda_i = \omega_i^2$, $i = 1, 2, 3$, *are the squares of the natural frequencies of the first, second, and third modes of vibration, respectively.*

2: Solve the Generalized Eigenvalue Problem Using the Cholesky QR Algorithm

Let's take $m_1 = 20{,}000$, $m_2 = 30{,}000$, $m_3 = 40{,}000$, and $k_1 = k_2 = k_3 = 10^9 \times 1.5$. Then

$$M = \text{diag}(20{,}000, 30{,}000, 40{,}000)$$

$$K = 10^9 \begin{pmatrix} 3 & -1.5 & 0 \\ -1.5 & 3 & -1.5 \\ 0 & -1.5 & 1.5 \end{pmatrix}$$

$$L = \begin{pmatrix} 141.4214 & 0 & 0 \\ 0 & 173.2051 & 0 \\ 0 & 0 & 200.000 \end{pmatrix}$$

$$C = 10^5 \begin{pmatrix} 1.5 & -0.6124 & 0 \\ -0.6124 & 1 & -0.4330 \\ 0 & -0.4330 & 0.3750 \end{pmatrix}$$

The eigenvalues of (K, M) = the eigenvalues of C are $10^5(1.9508, 0.8382, 0.0860)$.

The natural frequencies are $10^2(4.4168, 2.8951, 0.9273)$. The eigenvectors are

$$\begin{pmatrix} 0.0056 \\ -0.0034 \\ 0.0008 \end{pmatrix}, \begin{pmatrix} -0.0040 \\ -0.0035 \\ 0.0028 \end{pmatrix}, \begin{pmatrix} 0.0017 \\ 0.0031 \\ 0.0040 \end{pmatrix}$$

These eigenvectors can be used to determine different configurations of the system for different modes; that is, they can be used to see how the structures vibrate when excited at specific natural frequencies.

Remark: In practical computations, it is much better to keep K in factored form, $K = BB^T$, where

$$B = \begin{pmatrix} \sqrt{k_1} & -\sqrt{k_2} & 0 \\ 0 & \sqrt{k_2} & -\sqrt{k_3} \\ 0 & 0 & \sqrt{k_3} \end{pmatrix}$$

without forming K explicitly.

9.6.2 A Case Study: The Vibration of a Building

Consider a four-story reinforced concrete building, as shown in Figure 9.2. The floors and roof, which are fairly rigid, are represented by lumped masses m_1 to m_4 having a horizontal motion caused by shear deformation of columns, and k_1 to k_4 are equivalent spring constants of columns that act as springs in parallel.

We would like to study the configuration when the building is vibrating in its first two modes (corresponding to two smallest eigenvalues).

1. Formulate the problem as a *symmetric definite generalized eigenvalue problem* in terms of mass and stiffness matrices:

$$Kx = \lambda Mx$$

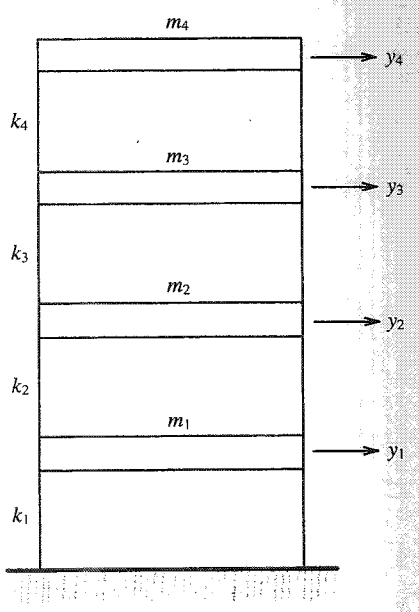


FIGURE 9.2 Schematic of a Four-Story Building

As in the previous example, M is diagonal and K is tridiagonal:

$$M = \text{diag}(m_1, m_2, m_3, m_4)$$

$$K = \begin{pmatrix} k_1 + k_2 & -k_2 & 0 & 0 \\ -k_2 & k_2 + k_3 & -k_3 & 0 \\ 0 & -k_3 & k_3 + k_4 & -k_4 \\ 0 & 0 & -k_4 & k_4 \end{pmatrix}$$

Taking $m_1 = 5 \times 10^7$, $m_2 = 4 \times 10^7$, $m_3 = 3 \times 10^7$, $m_4 = 2 \times 10^7$, and $k_1 = 10 \times 10^{14}$, $k_2 = 8 \times 10^{14}$, $k_3 = 6 \times 10^{14}$, $k_4 = 4 \times 10^{14}$, we have

$$K = 10^{14} \begin{pmatrix} 1.8 & -8 & 0 & 0 \\ -8 & 14 & -6 & 0 \\ 0 & -6 & 10 & -4 \\ 0 & 0 & -4 & 4 \end{pmatrix}$$

and

$$M = 10^7 \text{diag}(5, 4, 3, 2)$$

2. Find the eigenvalues and eigenvectors using the Cholesky QR algorithm.

$$L = 10^3 \begin{pmatrix} 7.0711 & 0 & 0 & 0 \\ 0 & 6.3246 & 0 & 0 \\ 0 & 0 & 5.4772 & 0 \\ 0 & 0 & 0 & 4.4721 \end{pmatrix}$$

$$C = 10^7 \begin{pmatrix} 3.6 & -1.7889 & 0 & 0 \\ -1.7889 & 3.5000 & -1.7321 & 0 \\ 0 & -1.7321 & 3.3333 & -1.6330 \\ 0 & 0 & -1.6330 & 2.000 \end{pmatrix}$$

The eigenvalues of C, which are the eigenvalues of the pencil (K, M), are

$$10^7 \begin{pmatrix} 6.1432 \\ 1.8516 \\ 0.3435 \\ 4.0950 \end{pmatrix}$$

The approximate eigenvectors of (K, M) associated with these eigenvalues are:

$$10^{-3} \begin{pmatrix} 0.0666 \\ -0.1058 \\ 0.0977 \\ -0.0472 \end{pmatrix}, \quad 10^{-3} \begin{pmatrix} -0.0785 \\ -0.0858 \\ 0.0104 \\ 0.1403 \end{pmatrix},$$

$$10^{-3} \begin{pmatrix} 0.0370 \\ 0.0753 \\ 0.1091 \\ 0.1318 \end{pmatrix}, \quad 10^{-3} \begin{pmatrix} 0.0896 \\ -0.02777 \\ -0.1085 \\ 0.1036 \end{pmatrix}$$

The eigenvectors corresponding to the two smallest eigenvalues $10^7(0.3435)$ and $10^7(1.8516)$ are

$$10^{-3} \begin{pmatrix} 0.0370 \\ 0.0753 \\ 0.1091 \\ 0.1318 \end{pmatrix}, \quad 10^{-3} \begin{pmatrix} -0.0785 \\ -0.0858 \\ 0.0104 \\ 0.1403 \end{pmatrix}$$

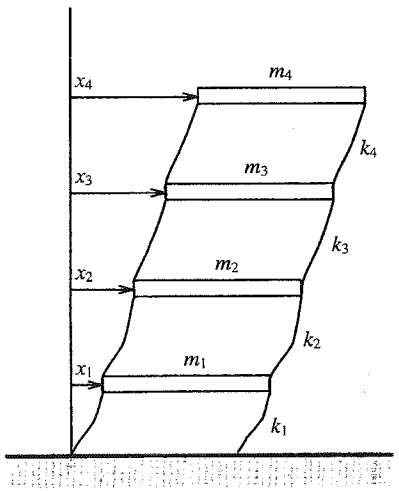


FIGURE 9.3 First Mode of Vibration of Four-Story Building

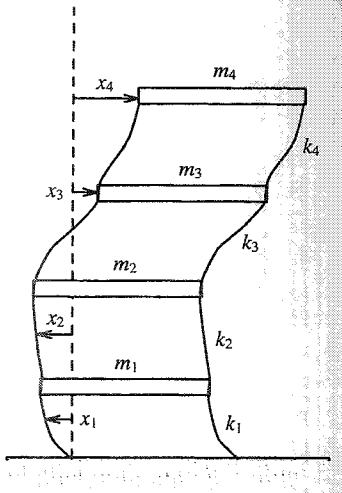


FIGURE 9.4 Second Mode of Vibration of Four-Story Building

The first two modes of vibration are shown in Figures 9.3 and 9.4, respectively.

9.6.3 Forced Harmonic Vibration

In the previous two examples, we considered vibration of systems without any external force. Consider now a system with two degrees of freedom excited by a harmonic force $F_1 \sin \omega t$, as shown in Figure 9.5.

Then the differential equations of motion for the system become

$$m_1 \ddot{y}_1 = -k(y_1 - y_2) - ky_1 + F_1 \sin \omega t$$
$$m_2 \ddot{y}_2 = k(y_1 - y_2) - ky_2$$

or

$$\begin{pmatrix} m_1 & 0 \\ 0 & m_2 \end{pmatrix} \begin{pmatrix} \ddot{y}_1 \\ \ddot{y}_2 \end{pmatrix} + \begin{pmatrix} 2k & -k \\ -k & 2k \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} F_1 \\ 0 \end{pmatrix} \sin \omega t \tag{9.6.1}$$

Assuming a solution of the form

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \sin \omega t$$

and substituting into the equations, we get

$$(2k - m_1\omega^2)x_1 - kx_2 = F_1$$
$$-kx_1 + (2k - m_2\omega^2)x_2 = 0$$

In matrix form,

$$\begin{pmatrix} (2k - m_1\omega^2) & -k \\ -k & (2k - m_2\omega^2) \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} F_1 \\ 0 \end{pmatrix}$$

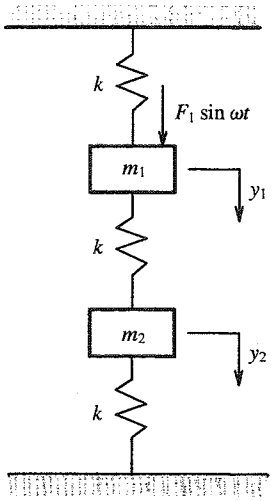


FIGURE 9.5 Forced Vibration of a 2-Degree-of-Freedom System

These equations can be easily solved to obtain explicit expressions for the amplitudes x_1 and x_2:

$$x_1 = \frac{(2k - m_2\omega^2)F_1}{m_1 m_2(\omega_1^2 - \omega^2)(\omega_2^2 - \omega^2)} \tag{9.6.2}$$

$$x_2 = \frac{kF_1}{m_1 m_2(\omega_1^2 - \omega^2)(\omega_2^2 - \omega^2)} \tag{9.6.3}$$

where ω_1 and ω_2 are the modal frequencies. For the special case when $m_1 = m_2 = m$, ω_1 and ω_2 are given by

$$\omega_1 = \sqrt{\frac{k}{m}} \quad \text{and} \quad \omega_2 = \sqrt{3\frac{k}{m}}$$

and x_1 and x_2 are given by

$$x_1 = \frac{(2k - m\omega^2)F_1}{m^2(\omega_1^2 - \omega^2)(\omega_2^2 - \omega^2)} \tag{9.6.4}$$

$$x_2 = \frac{kF_1}{m^2(\omega_1^2 - \omega^2)(\omega_2^2 - \omega^2)} \tag{9.6.5}$$

From this, it follows immediately that whenever ω is equal to or close to ω_1 or ω_2, the amplitude becomes arbitrarily large, signaling the occurrence of *resonance*. Note that in this case, the denominator is zero or close to it. This situation is very alarming to an engineer.

In other words, if the frequency of the imposed periodic force is equal or nearly equal to one of the natural frequencies of a system, resonance results, a situation that is dangerous.

Example 9.6.1 To demonstrate the consequences of excitation at or near resonance, let us consider an airplane landing on a rough runway. The airplane is shown schematically in Figure 9.6. The fuselage and engine are assumed to have a combined mass m_1. The wings are modeled by lumped masses m_2 and m_3 and stiffness k_2 and k_3; k_1 represents the combined stiffness of the landing gear and tires. The masses of the wheels and landing gear are assumed negligible compared to the mass of the fuselage and the wings. The system is modeled in Figure 9.7.

The runway is modeled by a sinusoidal curve, as shown in Figure 9.6. Let the contour be described by $y = y_0 \sin \omega t$, and let the airplane be subjected to a forcing input of $f_1 \sin \omega t$, where $f_1 = k_1 y_0$. The equations of motion for the 3-degree-of-freedom system so described are given by

$$\begin{bmatrix} m_1 & 0 & 0 \\ 0 & m_2 & 0 \\ 0 & 0 & m_3 \end{bmatrix} \begin{Bmatrix} \ddot{y}_1 \\ \ddot{y}_2 \\ \ddot{y}_3 \end{Bmatrix} + \begin{bmatrix} k_1 + k_2 + k_3 & -k_2 & -k_3 \\ -k_2 & k_2 & 0 \\ -k_3 & 0 & k_3 \end{bmatrix} \begin{Bmatrix} y_1 \\ y_2 \\ y_3 \end{Bmatrix} = \begin{Bmatrix} f_1 \sin \omega t \\ 0 \\ 0 \end{Bmatrix},$$

where y_1, y_2, and y_3 represent motion relative to the ground.

Let

$$\begin{aligned} k_1 &= 1.7 \times 10^5 \text{ N/m} \\ k_3 = k_2 &= 6 \times 10^5 \text{ N/m} \\ m_1 &= 1300 \text{ kg} \\ m_2 &= m_3 = 300 \text{ kg} \end{aligned}$$

The natural frequencies obtained by solving the generalized eigenvalue problem $Kx = \lambda Mx = \omega^2 Mx$ are given by:

$$\begin{aligned} \omega_1 &= 9.39 \text{ rad/s} \\ \omega_2 &= 44.72 \text{ rad/s} \\ \omega_3 &= 54.46 \text{ rad/s} \end{aligned}$$

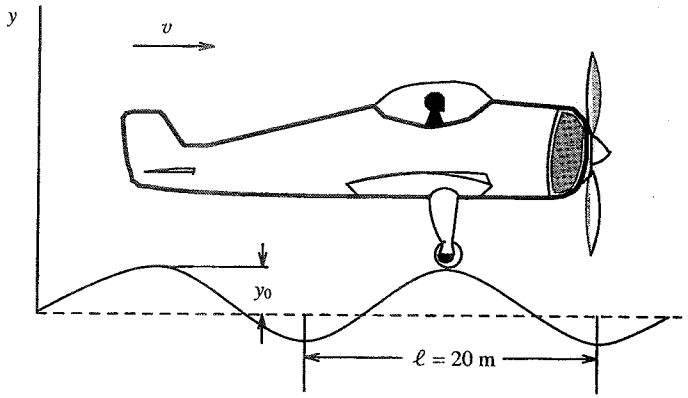


FIGURE 9.6 Airplane Landing on a Runway

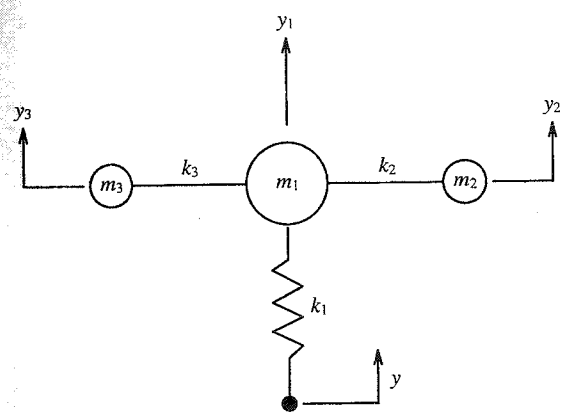


FIGURE 9.7 Model of the Airplane

The forcing frequency ω is related to the landing velocity v by $v = \omega\ell/2\pi$. So if $\omega = \omega_1$,

$$v = \frac{\omega_1 \ell}{2\pi} = \frac{9.39 \times 20 \text{ m/s}}{2\pi} = 29.8 \text{ m/s} = 107 \text{ km/h}$$

Thus, if the landing velocity is 107 km/h, or close to it, there is danger of excitation at or near the resonance.

9.7 Applications to Decoupling and Model Reduction

We have already seen how generalized eigenvalue problems arise in applications. In this section we mention a few more engineering applications. These include decoupling of a second-order system of differential equations and model reduction.

9.7.1 Decoupling of a Second-Order System

Case 1: The Undamped System

As we have seen before, vibration problems in structural analysis give rise to a homogeneous system of differential equations of second order of the form:

$$M\ddot{y} + Ky = 0 \tag{9.7.1}$$

where

$$\ddot{y} = \frac{d^2y}{dt^2}$$

and

$$y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

is an n-vector. The matrices M and K are, as usual, the mass and stiffness matrices. Assuming that these matrices are symmetric and M is positive definite, we now show how the simultaneous diagonalization technique (Algorithm 9.5.3) can be profitably employed to solve this system of second-order equations.

The idea is to decouple the system into n uncoupled equations so that each of these uncoupled equations can be solved using a standard technique. Let P be the modal matrix such that

$$P^T MP = I \tag{9.7.2}$$

$$P^T KP = \Lambda = \text{diag}(\omega_1^2, \ldots, \omega_n^2) \tag{9.7.3}$$

Let $y = Pz$, so the homogeneous system

$$M\ddot{y} + Ky = 0$$

becomes

$$MP\ddot{z} + KPz = 0$$

Next premultiplying by P^T we have

$$P^T MP\ddot{z} + P^T KPz = 0$$

or

$$\ddot{z} + \Lambda z = 0$$

The homogeneous system therefore decouples:

$$\ddot{z}_i + \omega_i^2 z_i = 0, \qquad i = 1, 2, \ldots, n$$

or

$$\begin{aligned} \ddot{z}_1 + \omega_1^2 z_1 &= 0 \\ \ddot{z}_2 + \omega_2^2 z_2 &= 0 \\ &\vdots \\ \ddot{z}_n + \omega_n^2 z_n &= 0 \end{aligned} \tag{9.7.4}$$

where

$$z = \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{pmatrix}, \qquad \Lambda = \text{diag}(\omega_1^2, \omega_2^2, \ldots, \omega_n^2)$$

The solution of the original system can now be obtained by solving these n decoupled equations using standard techniques and then recovering the original solution y from

$$y = Pz$$

Thus, if the solutions of the transformed system (9.7.4) are given by

$$z_i = A_i \cos \omega_i t + B_i \sin \omega_i t, \qquad i = 1, 2, \ldots, n$$

then the solutions of the original system (9.7.1) will be

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = P \begin{bmatrix} A_1 \cos \omega_1 t + B_1 \sin \omega_1 t \\ A_2 \cos \omega_2 t + B_2 \sin \omega_2 t \\ \vdots \\ A_n \cos \omega_n t + B_n \sin \omega_n t \end{bmatrix} \tag{9.7.5}$$

The constants A_i and B_i are to be determined from the initial conditions, such as

$$y_i|_{t=0} = \text{displacement at time } t = 0$$
$$\dot{y}_i|_{t=0} = \text{initial velocity}$$

Example 9.7.1 Consider the system with three masses in the example of the Section 9.6.1. Suppose that the system, when released from rest at $t = 0$, is subjected to a displacement.

We would like to find the undamped time response of the system. The initial conditions are

$$y_0 = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}, \qquad \dot{y}_0 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

Because the initial velocities are zeros, we obtain

$$\dot{y}_0 = P \begin{pmatrix} B_1\omega_1 \\ B_2\omega_2 \\ B_3\omega_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

These give $B_1 = B_2 = B_3 = 0$.

Again, at $t = 0$, we have

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = P \begin{pmatrix} A_1 \\ A_2 \\ A_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

Recall that, for this problem, the matrix of eigenvectors

$$P = \begin{pmatrix} 0.0056 & -0.0040 & 0.0017 \\ -0.0034 & -0.0035 & 0.0031 \\ 0.0008 & 0.0028 & 0.0040 \end{pmatrix}$$

Then the solution of the linear system

$$P \begin{pmatrix} A_1 \\ A_2 \\ A_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

is

$$A_1 = 6.1816$$
$$A_2 = 50.6264$$
$$A_3 = 705.2650$$

Substituting these values of A_1, A_2, and A_3 and the values of ω_1, ω_2, and ω_3 obtained (in Section 9.6.1) in

$$y = Pz$$

or

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = P \begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix} = P \begin{pmatrix} A_1 \cos \omega_1 t \\ A_2 \cos \omega_2 t \\ A_3 \cos \omega_3 t \end{pmatrix}$$

we obtain the values of y_1, y_2, and y_3 that give the undamped time response of the systems subject to the given initial conditions.

Case 2: The Damped Systems

Some damping, such as that due to air resistance or fluid and solid friction, is present in all real systems. Let us now consider damped homogeneous systems.

Let D be the damping matrix. Then the equations of motion of the damped system become

$$M\ddot{y} + D\dot{y} + Ky = 0 \tag{9.7.6}$$

If D is a linear combination of M and K; that is,

$$D = \alpha M + \beta K \tag{9.7.7}$$

where α and β are constants, then the system is said to have **proportional**, or **Rayleigh, damping**. Let P be the modal matrix. Then we have

$$\begin{aligned} P^T DP &= \alpha P^T MP + \beta P^T KP \\ &= \alpha I + \beta \Lambda \end{aligned}$$

Let $z = P^T y$. Then the preceding homogeneous damped equations are transformed to n decoupled equations:

$$\ddot{z}_i + (\alpha + \beta\omega_i^2)\dot{z}_i + \omega_i^2 z_i = 0, \qquad i = 1, 2, \ldots, n \tag{9.7.8}$$

In engineering practice it is customary to define *modal damping ratio* of the ith mode by

$$\zeta_i = \frac{\alpha + \beta\omega_i^2}{2\omega_i}$$

ζ_i is usually taken as a small number between 0 and 1. The most common values are $0 \le \zeta \le 0.05$ (Inman, 1994, 196). However, in some applications, such as in the design of flexible structures, $\{\zeta_i\}$ are taken to be as low as 0.005. On the other hand, for an automatic shock absorber, a value as high as $\zeta = 0.5$ is possible.

With the above definition of ζ_i, the decoupled equations (9.7.8) become

$$\ddot{z}_i + 2\zeta_i\omega_i\dot{z}_i + \omega_i^2 z_i = 0, \qquad i = 1, 2, \ldots, n$$

The solutions of these equations are then given by

$$z_i = e^{-\zeta_i\omega_i t}\left(A_i \cos \omega_i\sqrt{1 - \zeta_i^2}\,t + B_i \sin \omega_i\sqrt{1 - \zeta_i^2}\,t\right), \qquad i = 1, 2, \ldots, n$$

where the constants A_i and B_i are to be determined from the given initial equations.

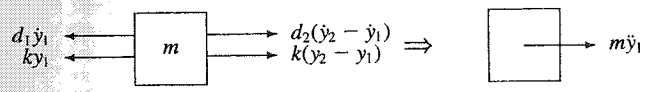
The original system can now be solved by solving these n decoupled equations separately and then recovering the original solution y from $y = Pz$.

Remark: Note that for **simultaneous diagonalization** of M, K, and D, proportional damping is just a special condition. A more general condition is $DM^{-1}K = KM^{-1}D$ (see Inman 1989).

Example 9.7.2 Usefulness of Proportional Damping Consider the system with 2 degrees of freedom (DOF) shown in Figure 9.8. We will show here why proportional damping is useful.

The equations of motion of the system are developed by considering a free body diagram for each mass.

For mass m:



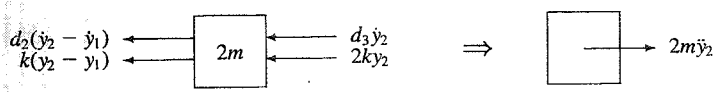
The equation of motion for this mass is

$$-d_1\dot{y}_1 + d_2(\dot{y}_2 - \dot{y}_1) - ky_1 + k(y_2 - y_1) = m\ddot{y}_1$$

or

$$m\ddot{y}_1 + (d_1 + d_2)\dot{y}_1 - d_2\dot{y}_2 + 2ky_1 - ky_2 = 0$$

For mass $2m$:



The equation of motion for this mass is

$$-d_2(\dot{y}_2 - \dot{y}_1) - d_3\dot{y}_2 - k(y_2 - y_1) - 2ky_2 = 2m\ddot{y}_2$$

or

$$2m\ddot{y}_2 - d_2\dot{y}_1 + (d_2 + d_3)\dot{y}_2 - ky_1 + 3ky_2 = 0$$

Thus, for the whole system we have

$$\begin{pmatrix} m & 0 \\ 0 & 2m \end{pmatrix}\begin{pmatrix} \ddot{y}_1 \\ \ddot{y}_2 \end{pmatrix} + \begin{pmatrix} d_1 + d_2 & -d_2 \\ -d_2 & d_2 + d_3 \end{pmatrix}\begin{pmatrix} \dot{y}_1 \\ \dot{y}_2 \end{pmatrix} + \begin{pmatrix} 2k & -k \\ -k & 3k \end{pmatrix}\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

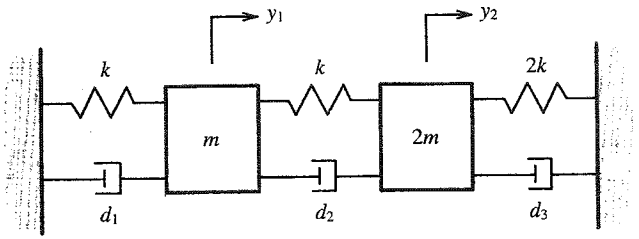


FIGURE 9.8

Let $k = 2$ and $m = 5$, and assume that $D = \alpha M + \beta K$. For this 2-DOF system, the mass matrix is

$$M = \begin{pmatrix} m & 0 \\ 0 & 2m \end{pmatrix} = \begin{pmatrix} 5 & 0 \\ 0 & 10 \end{pmatrix}$$

the stiffness matrix is

$$K = \begin{pmatrix} 2k & -k \\ -k & 3k \end{pmatrix} = \begin{pmatrix} 4 & -2 \\ -2 & 6 \end{pmatrix}$$

and the damping matrix is

$$D = \begin{pmatrix} d_1 + d_2 & -d_2 \\ -d_2 & d_2 + d_3 \end{pmatrix}$$

Consider two cases.

Case 1: $d_1 = 1, d_2 = 1, d_3 = 2$:

$$\begin{pmatrix} 2 & -1 \\ -1 & 3 \end{pmatrix} = \alpha\begin{pmatrix} 5 & 0 \\ 0 & 10 \end{pmatrix} + \beta\begin{pmatrix} 4 & -2 \\ -2 & 6 \end{pmatrix}$$

$$2 = 5\alpha + 4\beta \tag{9.7.9}$$

$$-1 = -2\beta \tag{9.7.10}$$

$$3 = 10\alpha + 6\beta \tag{9.7.11}$$

Equations (9.7.9), (9.7.10), and (9.7.11) are uniquely satisfied with $\beta = \frac{1}{2}$ and $\alpha = 0$. So, this is a case of proportional damping.

Case 2: $d_1 = 2, d_2 = 4, d_3 = 1$:

$$\begin{pmatrix} 6 & -4 \\ -4 & 5 \end{pmatrix} = \alpha\begin{pmatrix} 5 & 0 \\ 0 & 10 \end{pmatrix} + \beta\begin{pmatrix} 4 & -2 \\ -2 & 6 \end{pmatrix}$$

$$6 = 5\alpha + 4\beta \tag{9.7.12}$$

$$-4 = -2\beta \tag{9.7.13}$$

$$5 = 10\alpha + 6\beta \tag{9.7.14}$$

Equations (9.7.12), (9.7.13), and (9.7.14) cannot all be satisfied with any set of values of α and β. So, proportional damping is not possible. Furthermore, it is easy to check that, in this case, the general condition of simultaneous diagonalization, namely $DM^{-1}K = KM^{-1}D$, is violated.

Thus, in the first case, the equations of motion can be decoupled using $y = Pz$, obtaining real mode shapes and real natural frequencies. However, in the second case, such decoupling is not possible.

Damped Systems Under Force Excitation

When a damped system is subject to an external force F, the equations of motion are given by

$$M\ddot{y} + D\dot{y} + Ky = F(t) = \begin{Bmatrix} F_1(t) \\ F_2(t) \\ \vdots \\ F_n(t) \end{Bmatrix} \tag{9.7.15}$$

Assuming that M is symmetric positive definite, K is symmetric, and proportional damping, it is easy to see from our previous discussion that the preceding equations can be decoupled using simultaneous diagonalization.

Let $P = (p_{ij})$ be the modal matrix. Then

$$P^T F = \begin{pmatrix} p_{11} & p_{21} & \cdots & p_{n1} \\ p_{12} & p_{22} & \cdots & p_{n2} \\ \vdots & \vdots & & \vdots \\ p_{1i} & p_{2i} & \cdots & p_{ni} \\ \vdots & \vdots & & \vdots \\ p_{1n} & p_{2n} & \cdots & p_{nn} \end{pmatrix} \begin{pmatrix} F_1 \\ F_2 \\ \vdots \\ F_i \\ \vdots \\ F_n \end{pmatrix}$$

The decoupled equations will then be given by

$$\ddot{z}_i + 2\zeta_i\omega_i\dot{z}_i + \omega_i^2 z_i = p_{1i}F_1 + p_{2i}F_2 + \cdots + p_{ni}F_n$$

or

$$\ddot{z}_i + 2\zeta_i\omega_i\dot{z}_i + \omega_i^2 z_i = E_i(t) \tag{9.7.16}$$

where $E_i(t) = \sum_{j=1}^{n} p_{ji}F_j$, $i = 1, 2, \ldots, n$.

The function $E_i(t)$ is called the **exciting**, or **forcing function** of the ith mode. If each force F_i is written as

$$F_i = f_i s(t)$$

then

$$E_i(t) = s(t)\sum_{j=i}^{n} p_{ji}f_j$$

DEFINITION 9.7.1 The expression

$$\sum_{j=1}^{n} p_{ji}f_j$$

is called the **mode participation factor** for the ith mode.

Once the decoupled equations are solved for z_i, we can recover the solutions of the original equations from

$$y = Pz$$

or

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = P \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{pmatrix}$$

The solutions of the decoupled equations

$$\ddot{z}_i + 2\zeta_i\omega_i\dot{z}_i + \omega_i^2 z_i = E_i(t)$$

depend upon the nature of the force $F(t)$. For example, when the force is a shock-type force, such as an earthquake, we are normally interested in maximum responses, and the maximum values of $z_1, z_2, \ldots, z_n$ can be obtained from the responses of a single equation with one degree of freedom (see the example in Section 9.7.3).

9.7.2 The Reduction of a Large Model

Many applications give rise to a *very large* system of second-order differential equations:

$$M\ddot{y} + D\dot{y} + Ky = F(y, t) \tag{9.7.17}$$

It is, therefore, infinite-dimensional in theory but of very large dimension in practice. A finite element–generated reduced-order model can be a large second-order system. The dimension of the problem in such an application can be *several thousand*. Naturally, the solution of a large system will lead to a solution of a very large generalized eigenvalue problem. Unfortunately, effective numerical techniques for computing a large number of eigenvalues and eigenvectors of a generalized eigenvalue problem are virtually nonexistent and not very well developed (see Section 9.9). It is, therefore, natural to think of solving the problem by constructing a reduced-order model with the help of a few eigenvalues and eigenvectors that are feasible to compute. Such a thought is based on an assumption that in many instances the response of the structure depends mainly on a first few eigenvalues (low frequencies); usually the higher modes do not get excited.

We now show how the computations can be simplified by using only the knowledge of a few eigenvalues and eigenvectors.

Suppose that, under the usual assumption that M and K are symmetric and of order n and M is positive definite, we were able to compute only the first few normal modes, perhaps m of them, where $m \ll n$. Let the matrix of these normal modes be $P_{n\times m}$. Then

$$P^T MP = I_{m\times m}$$
$$P^T KP = \Lambda_{m\times m} = \operatorname{diag}(\omega_1^2, \ldots, \omega_m^2)$$

Setting $y = Pz$ and assuming proportional damping, the system of n differential equations (9.7.17) then reduces to m equations:

$$\ddot{z}_i + 2\zeta_i\omega_i\dot{z}_i + \omega_i^2 z_i = E_i(t), \qquad i = 1, 2, \ldots, m$$

where E_i is the ith coordinate of the vector $P^T F$. Once this small number of equations is solved, the displacement of any masses under the external force can be computed from

$$y_i = Pz_i, \qquad i = 1, \ldots, n$$

Sometimes only the maximum value of the displacement is of interest.

Several vibration groups in industry and the military use the following approximation to obtain the maximum value of y_i (Thomson 1992):

$$|y_i|_{\max} = |p_1 z_{1(\max)}| + \sqrt{\sum_{j=2}^{m} |p_j z_{j(\max)}|^2}$$

where p_i is the ith column of P evaluated at z_i.

9.7.3 A Case Study: The Potential Damage to a Building Due to an Earthquake

Suppose we are interested in finding the absolute maximum responses of the four-story building considered in the example of Section 9.6.2 when the building will be subjected to a strong earthquake, using the known response of the building due to a previous earthquake. We will use only the first two modes (the modes corresponding to the two lowest frequencies) in our calculations.

In Figure 9.9, the y_i, $i = 1, \ldots, 4$, denote the displacement of each floor relative to the moving support, whose motion is denoted by y_0.

The decoupled normal mode equations in modal form in this case can be written as:

$$\ddot{z}_i + 2\zeta_i\omega_i\dot{z}_i + \omega_i^2 z_i = -E_i\ddot{y}_0, \qquad i = 1, 2$$

where

$$\ddot{y}_0 = \text{absolute acceleration of the moving support}$$

$$E_i = \sum_{j=1}^{4} p_{ji}m_j = \text{mode participation factor of the chosen mode } p_i \text{ due to support excitation}$$

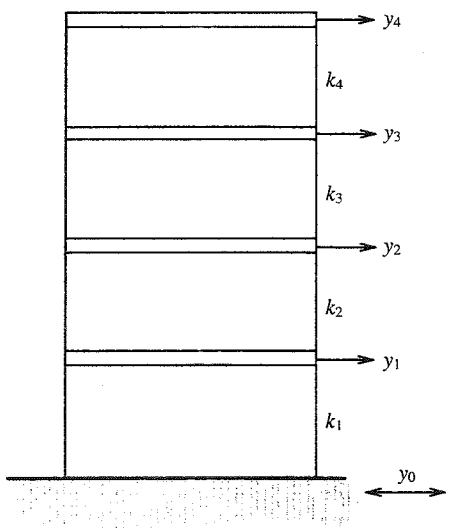


FIGURE 9.9 Building Subject to an Earthquake

p_{ji} are the coordinates of the participating mode P_i; that is,

$$P = (p_1, p_2) = \left[\begin{pmatrix} p_{11} \\ p_{21} \\ p_{31} \\ p_{41} \end{pmatrix}, \begin{pmatrix} p_{12} \\ p_{22} \\ p_{32} \\ p_{42} \end{pmatrix}\right]$$

where p_1 and p_2 are the two chosen participating modes.

Let R_1 and R_2 denote the maximum relative responses of $z_{1(\max)}$ and $z_{2(\max)}$ obtained from previous experience. Then we can take

$$z_{1(\max)} = E_1 R_1$$
$$z_{2(\max)} = E_2 R_2$$

This observation immediately gives

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix}_{\max} = E_1 R_1 \begin{pmatrix} p_{11} \\ p_{21} \\ p_{31} \\ p_{41} \end{pmatrix} + E_2 R_2 \begin{pmatrix} p_{12} \\ p_{22} \\ p_{32} \\ p_{42} \end{pmatrix}$$

Using the data of the example in Section 9.6.2, we have

$$p_1 = \text{the eigenvector corresponding to the smallest eigenvalue}$$
$$= 10^{-3}\begin{pmatrix} 0.0370 \\ 0.0753 \\ 0.1091 \\ 0.1318 \end{pmatrix}$$

$$p_2 = \text{the eigenvector corresponding to the second smallest eigenvalue}$$
$$= 10^{-3}\begin{pmatrix} -0.0785 \\ -0.0858 \\ 0.0104 \\ 0.1403 \end{pmatrix}$$

$$E_1 = m_1 p_{11} + m_2 p_{21} + m_3 p_{31} + m_4 p_{41} = 10^4(1.0772)$$
$$E_2 = m_1 p_{12} + m_2 p_{22} + m_3 p_{32} + m_4 p_{42} = 10^3(-4.2417)$$

Assume that $R_1 = 1.5$ in. and $R_2 = 0.25$ in. We obtain

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix}_{\max} = 10^{-3} \cdot 10^4(1.0772)(1.5)\begin{pmatrix} 0.0370 \\ 0.0753 \\ 0.1091 \\ 0.1318 \end{pmatrix}$$
$$+10^{-3} \cdot 10^3(-4.2417)(.25)\begin{pmatrix} -0.0785 \\ 0.0858 \\ 0.0104 \\ 0.1403 \end{pmatrix}$$

$$= \begin{pmatrix} 0.5979 \\ 1.2169 \\ 1.7636 \\ 2.1292 \end{pmatrix} + \begin{pmatrix} 0.0833 \\ 0.0910 \\ -0.0110 \\ -0.1488 \end{pmatrix}$$

$$= \begin{pmatrix} 0.6812 \text{ in.} \\ 1.3079 \text{ in.} \\ 1.7526 \text{ in.} \\ 1.9805 \text{ in.} \end{pmatrix}$$

Thus, the absolute maximum displacement (relative to the moving support) of the first floor is 0.6812 in., that of the second floor is 1.3079 in., and so forth. See Figure 9.10.

The absolute maximum relative displacements are obtained by adding the terms using their absolute values:

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix}_{\text{(abs. max.)}} = \begin{pmatrix} 0.6812 \text{ in.} \\ 1.3079 \text{ in.} \\ 1.7746 \text{ in.} \\ 2.2780 \text{ in.} \end{pmatrix}$$

Note: The contribution to the second participating mode to responses is small in comparison with the contribution of the first mode.

Average Maximum Relative Displacement of the Masses

The absolute maximum relative displacement of the masses provides us with an upper bound for the largest relative displacements the masses can have and thus helps us to choose design parameters. Another practice for such a measure in engineering literature has been to use the **root sum square** of the same terms, giving the "average" maximum relative displacement values:

$$(y_i)_{\text{average max}} = \sqrt{(E_1R_1p_{i1})^2 + (E_2R_2p_{i2})^2 + \cdots + (E_kR_kP_{ik})^2}$$

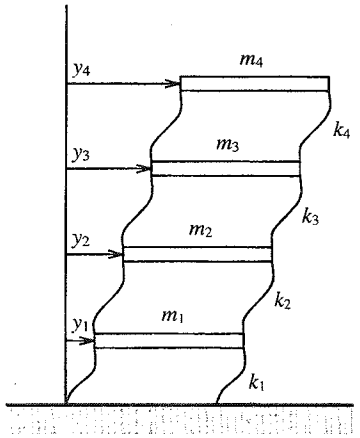


FIGURE 9.10 Absolute Maximum Displacement

For the preceding example, $k = 2$, and the average maximum relative displacements are given by

$$(y_1)_{\text{average max}} = \sqrt{(E_1R_1p_{11})^2 + (E_2R_2p_{12})^2} = 0.9975 \text{ in.}$$

$$(y_2)_{\text{average max}} = \sqrt{(E_1R_1p_{21})^2 + (E_2R_2p_{22})^2} = 1.5610 \text{ in.}$$

and so on.

9.8 The Quadratic Eigenvalue Problem

So far we have considered the second-order system:

$$M\ddot{y} + D\dot{y} + Ky = 0 \tag{9.8.1}$$

under the assumption of proportional or modal damping. This assumption lets us use the simultaneous diagonalization on M, K, and D (see Section 9.7.1, Case 2), which, of course, as we saw in Section 9.5, amounts to solving a generalized eigenvalue problem.

In case of general damping, assuming that the solutions are of the form $y = xe^{\lambda t}$, where x is a constant vector, we will, however, have the quadratic eigenvalue problem

$$(\lambda^2 M + \lambda D + K)x = 0 \tag{9.8.2}$$

This problem has $2n$ eigenvalues, and there are $2n$ eigenvectors corresponding to them. We describe two existing approaches to solve this quadratic eigenvalue problem and show how to extract frequencies and modal damping ratios from the eigenvalues. We assume, as before, that M is symmetric positive definite.

Approach 1: Reduction to a Standard Eigenvalue Problem

We can rewrite (9.8.2) as

$$\begin{pmatrix} 0 & I \\ -M^{-1}K & -M^{-1}D \end{pmatrix} \begin{pmatrix} x \\ \lambda x \end{pmatrix} = \lambda \begin{pmatrix} x \\ \lambda x \end{pmatrix}$$

or

$$Au = \lambda u \tag{9.8.3}$$

where

$$A = \begin{pmatrix} 0 & I \\ -M^{-1}K & -M^{-1}D \end{pmatrix}, \qquad u = \begin{pmatrix} x \\ \lambda x \end{pmatrix} \tag{9.8.4}$$

Thus we have reduced the quadratic eigenvalue problem (9.8.2) to the $2n \times 2n$ standard eigenvalue problem (9.8.3), where A and u are given by (9.8.4).

If the second-order system is the model of a vibrating structure, then the natural frequencies and the modal damping ratios can be computed from the eigenvalues as follows.

Computing Frequencies and Damping Ratios

Let $\lambda_k = \alpha_k + i\beta_k$; that is, α_k and β_k are, respectively, the real and imaginary parts of the complex eigenvalue λ_k. Then the **natural frequencies** ω_k are given

$$\omega_k = \sqrt{\alpha_k^2 + \beta_k^2}, \qquad k = 1, 2, \ldots, n \tag{9.8.5}$$

(Inman 1994, 462). Note that the eigenvalues λ_k occur in complex conjugate pairs.

The modal damping ratios are given by

$$\zeta_k = \frac{-\alpha_k}{\sqrt{\alpha_k^2 + \beta_k^2}} \tag{9.8.6}$$

Approach 2: Reduction to a Symmetric Generalized Eigenvalue Problem

There are several other ways to rewrite (9.8.2). One such way is

$$\begin{pmatrix} D & K \\ K & 0 \end{pmatrix}\begin{pmatrix} \lambda x \\ x \end{pmatrix} = \lambda \begin{pmatrix} -M & 0 \\ 0 & K \end{pmatrix}\begin{pmatrix} \lambda x \\ x \end{pmatrix}$$

or

$$Bz = \lambda Cz \tag{9.8.7}$$

where

$$B = \begin{pmatrix} D & K \\ K & 0 \end{pmatrix}, \qquad C = \begin{pmatrix} -M & 0 \\ 0 & K \end{pmatrix}, \qquad z = \begin{pmatrix} \lambda x \\ x \end{pmatrix} \tag{9.8.8}$$

The equation (9.8.7) is a symmetric generalized eigenvalue problem.

Here both B and C are, in general, indefinite; therefore, the eigenvectors can be complex. The natural frequencies and modal damping ratios can be extracted from the eigenvalues as in Approach 1.

Summary. The quadratic eigenvalue problem

$$(\lambda^2 M + \lambda D + K)x = 0$$

can be either reduced to the **standard nonsymmetric eigenvalue problem**

$$Au = \lambda u$$

where

$$A = \begin{pmatrix} 0 & I \\ -M^{-1}K & -M^{-1}D \end{pmatrix}, \qquad u = \begin{pmatrix} x \\ \lambda x \end{pmatrix}$$

or to the **symmetric generalized eigenvalue problem**

$$Bz = \lambda Cz$$

where

$$B = \begin{pmatrix} D & K \\ K & 0 \end{pmatrix}, \qquad C = \begin{pmatrix} -M & 0 \\ 0 & K \end{pmatrix}, \qquad z = \begin{pmatrix} \lambda x \\ x \end{pmatrix}$$

Once the eigenvalues and eigenvectors are computed, the natural frequencies and the modal damping ratios of a vibrating structure whose mathematical model is the second-order system (9.8.1) can be computed from the eigenvalues of (9.8.3) or of (9.8.7).

A Word of Caution About Using Approach 1: If M is ill conditioned, the explicit formation of the matrix A given by (9.8.4) by actually computing M^{-1} will be computationally disastrous. A will be computed inaccurately, as will the eigenvalues and eigenvectors. Also note that A is nonsymmetric.

MATCOM Note: Approach 1 and Approach 2 have been implemented, respectively, in MATCOM programs QUADEIG1 and QUADEIG2.

Example 9.8.1 Find the eigenvalues and eigenvectors of the quadratic eigenvalue problem (9.8.2) with

$$M = \text{diag}(1, 2), \qquad D = \begin{pmatrix} 2 & 2 \\ 2 & 2 \end{pmatrix}, \qquad K = \begin{pmatrix} 1 & 2 \\ 2 & 0 \end{pmatrix}$$

Approach 1: Form

$$A = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -1 & -2 & -2 & -2 \\ -1 & 0 & -1 & -1 \end{pmatrix}$$

The eigenvalues of A and the corresponding eigenvectors are -2, 1, -1, and -1.

$$\begin{pmatrix} 0.4000 \\ 0.2000 \\ -0.8000 \\ -0.4000 \end{pmatrix} \begin{pmatrix} 0.5000 \\ -0.5000 \\ 0.5000 \\ -0.5000 \end{pmatrix} \begin{pmatrix} 0.6696 \\ 0.2273 \\ -0.6696 \\ -0.2273 \end{pmatrix} \begin{pmatrix} 0.3486 \\ -0.6152 \\ -0.3486 \\ 0.6152 \end{pmatrix}$$

Thus, the eigenvalues and the eigenvectors of the quadratic eigenvalue problem (9.8.2) are -2, 1, -1, and -1; and

$$\begin{pmatrix} 0.4000 \\ 0.2000 \end{pmatrix}, \begin{pmatrix} 0.5000 \\ -0.5000 \end{pmatrix}, \begin{pmatrix} 0.6696 \\ 0.2273 \end{pmatrix}, \begin{pmatrix} 0.3486 \\ -0.6152 \end{pmatrix}$$

Approach 2: Form

$$B = \begin{pmatrix} 2 & 2 & 1 & 2 \\ 2 & 2 & 2 & 0 \\ 1 & 2 & 0 & 0 \\ 2 & 0 & 0 & 0 \end{pmatrix}, \qquad C = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & -2 & 0 & 0 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 2 & 0 \end{pmatrix}$$

The eigenvalues and the eigenvectors of the generalized eigenvalue problem $Bz = \lambda Cz$ are

$$1, -2, -1, -1, \quad \text{and} \quad \begin{pmatrix} -0.5000 \\ 0.5000 \\ -0.5000 \\ 0.5000 \end{pmatrix}, \begin{pmatrix} -0.8000 \\ -0.4000 \\ 0.4000 \\ 0.2000 \end{pmatrix}, \begin{pmatrix} -0.5000 \\ -0.5000 \\ 0.5000 \\ 0.5000 \end{pmatrix}, \begin{pmatrix} 0.6747 \\ 0.2117 \\ -0.6747 \\ -0.2117 \end{pmatrix}$$

Thus, the eigenvalues of the quadratic eigenvalue problem (9.8.2) are $1, -2, -1, -1$; and the eigenvectors are

$$\begin{pmatrix} -0.5000 \\ 0.5000 \end{pmatrix}, \begin{pmatrix} 0.4000 \\ 0.2000 \end{pmatrix}, \begin{pmatrix} 0.5000 \\ 0.5000 \end{pmatrix}, \begin{pmatrix} -0.6747 \\ -0.2117 \end{pmatrix}$$

9.9 Symmetric Generalized Eigenvalue Problems for Large and Structured Matrices

As we have seen in several previous case studies, in many practical situations the matrices A and B are structured: *Tridiagonal and banded cases are quite common.* Unfortunately, the Cholesky algorithm, when applied to such structured problems, will very often destroy the sparsity. Even though A and B are banded, the matrix $C = L^{-1}A(L^T)^{-1}$ will, in general, be full. Thus, simultaneous diagonalization techniques are not practical for large and sparse matrices.

9.9.1 The Sturm Sequence Method for Tridiagonal *A* and *B*

We now present a method that can be used when A and B are both symmetric tridiagonal and B is positive definite. The method is a generalization of the Sturm sequence algorithm for the symmetric eigenvalue problem described earlier in Chapter 8 and takes advantage of the tridiagonal forms of A and B. Let

$$A = \begin{pmatrix} \alpha_1 & \beta_1 & & 0 \\ \beta_1 & \ddots & \ddots & \\ & \ddots & \ddots & \beta_{n-1} \\ 0 & & \beta_{n-1} & \alpha_n \end{pmatrix}, \qquad B = \begin{pmatrix} \alpha_1' & \beta_1' & & 0 \\ \beta_1' & \ddots & \ddots & \\ & \ddots & \ddots & \beta_{n-1}' \\ 0 & & \beta_{n-1}' & \alpha_n' \end{pmatrix}$$

Define the sequence of polynomials $\{p_k(\lambda)\}$ given by

$$p_0(\lambda) = 1 \tag{9.9.1}$$

$$p_1(\lambda) = \alpha_1 - \lambda\alpha_1' \tag{9.9.2}$$

$$p_r(\lambda) = (\alpha_r - \lambda\alpha_r')p_{r-1}(\lambda) - (\beta_{r-1} - \lambda\beta_{r-1}')^2 p_{r-2}(\lambda) \tag{9.9.3}$$

$$r = 2, 3, \ldots, n$$

Then it can be shown (Exercise 22) that these polynomials form a Sturm sequence.

The eigenvalues of the pencil $(A - \lambda B)$ are then given by the zeros of $p_n(\lambda)$. The zeros can be found by *bisection* or any other suitable root-finding method for polynomials. For a proof of the algorithm, see Wilkinson (1965, 340–341).

MATCOM Note: The preceding method has been implemented in MATCOM program GENSTURM.

Example 9.9.1

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{pmatrix}, \qquad B = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 5 \end{pmatrix}$$

$$p_0(\lambda) = 1, \qquad p_1(\lambda) = 1 - \lambda$$

$$\begin{aligned} p_2(\lambda) &= (2 - 4\lambda)(1 - \lambda) - (0 - \lambda)^2 \cdot 1 \\ &= 3\lambda^2 - 6\lambda + 2 \\ p_3(\lambda) &= (3 - 5\lambda)(3\lambda^2 - 6\lambda + 2) - (-\lambda)^2(1 - \lambda) \\ &= -14\lambda^3 + 38\lambda^2 - 28\lambda + 6 \end{aligned}$$

The roots of $p_3(\lambda) = 0$ are 1.6708, 0.6472, and 0.3964. It can easily be checked that these are the generalized eigenvalues of the pencil $(A - \lambda B)$.

Computing an Eigenvector

Once an eigenvalue is computed, the corresponding eigenvector can be computed using the inverse iteration by taking full advantage of the tridiagonal forms of A and B as follows:

Let y_0 be the initial approximation of an eigenvector corresponding to a computed eigenvalue λ. Then

For $i = 0, 1, 2, \ldots$, do until convergence occurs.

Step 1: Solve for x_{i+1}: $(A - \lambda B)x_{i+1} = y_i$
Step 2: Form $y_{i+1} = Bx_{i+1}$.

Remarks:

1. About two iterations per eigenvector are adequate.
2. Note that the system $(A - \lambda B)x_{i+1} = y_i$ can be solved by taking advantage of the tridiagonal forms of A and B.

9.9.2 The Lanczos Algorithm for the Pencil $A - \lambda B$

In this section, we show how the symmetric Lanczos method described in Chapter 8 for a matrix A can be extended to the symmetric definite generalized eigenvalue problem $Ax = \lambda Bx$. The Lanczos method to be described here is useful for estimating a few extremal eigenvalues of a large and sparse symmetric definite generalized eigenvalue problem. Remember, large symmetric definite generalized eigenvalue problems are common in practice and, like most practical problems, they are sparse. The Lanczos method, which relies only on matrix–vector multiplications, is suitable to preserve sparsity.

Let $B = LL^T$ be the Cholesky factorization of B.

ALGORITHM 9.9.1 The Lanczos Algorithm for the Symmetric Definite Pencil Given a symmetric matrix A and a symmetric positive definite matrix B, the following *Lanczos* algorithm constructs a symmetric tridiagonal matrix

$$T_j = \begin{pmatrix} \alpha_1 & \beta_1 & & 0 \\ \beta_1 & \ddots & \ddots & \\ & \ddots & \ddots & \beta_{j-1} \\ 0 & & \beta_{j-1} & \alpha_j \end{pmatrix}$$

and an orthonormal matrix $V_j = \{v_1, v_2, \ldots, v_j\}$ such that

$$V_j^T A V_j = T_j$$
$$V_j^T B V_j = I_{j\times j}$$

Step 1: Choose v_1 such that $v_1^T B v_1 = 1$

Step 2: Set $\beta_0 = 1$, $r_0 = v_1$, $v_0 = 0$

Step 3: For $i = 1, 2, \ldots, j$ do

If $i > 1$, solve: $Bv_i = r_{i-1}/\beta_{i-1}$

$$\alpha_i = v_i^T(Av_i - Bv_{i-1})$$
$$r_i = Av_i - \beta_{i-1}Bv_{i-1} - \alpha_i Bv_i$$
$$\beta_i = \|L^{-1}r_i\|_2$$

Notes:

1. The Lanczos vectors in this case are **B-orthonormal**:

$$v_i^T B v_i = 1$$
$$v_i^T B v_j = 0, \qquad i \neq j$$

2. For $j = n$:

$$V^T B V = I_{n\times n}$$
$$V^T A V = T_{n\times n}, \qquad \text{a symmetric tridiagonal matrix}$$

3. If (θ, s) is an eigenpair of T_j, then $(\theta, V_j s)$ is the corresponding Ritz pair for the pencil (A, B).

Computational Remarks

1. Note that $L^{-1}r_i$ can be computed by solving the lower triangular system

$$Ly_i = r_i$$

Also,

$$Bv_i = \frac{r_{i-1}}{\beta_{i-1}}$$

is equivalent to

$$L^T v_i = L^{-1}\left(\frac{r_{i-1}}{\beta_{i-1}}\right), \qquad \text{because } B = LL^T$$

Thus, to implement the scheme, we need (1) a Cholesky factorization routine for a sparse matrix, (2) routines for sparse triangular systems, and (3) a routine for matrix–vector multiplication.

2. At each iteration, the vector βV_i should be saved for use in the next iteration.

MATCOM Note: Algorithm 9.9.1 has been implemented in MATCOM program LANSYMGN.

9.9.3 Estimating the Eigenvalues

Fact: For large enough j the extreme eigenvalues of the tridiagonal matrices

$$T_j = \begin{pmatrix} \alpha_1 & \beta_1 & & 0 \\ \beta_1 & \ddots & \ddots & \\ & \ddots & \ddots & \beta_{j-1} \\ 0 & & \beta_{j-1} & \alpha_j \end{pmatrix}$$

will approximate some eigenvalues of the pencil $(A - \lambda B)$.

Example 9.9.2

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \end{pmatrix}$$

$$B = \text{diag}(1, 2, 3)$$
$$L = \text{diag}(1, 1.4142, 1.7321)$$
$$\beta_0 = 1, \qquad v_1 = (1, 0, 0)^T, \qquad r_0 = v_1$$

(Note that $v_1^T B v_1 = 1$.)

$i = 1$:

$$\alpha_1 = v_1^T(Av_1 - Bv_0) = 1$$
$$r_1 = \begin{pmatrix} 0 \\ 2 \\ 3 \end{pmatrix}$$
$$\beta_1 = 2.2361$$

$i = 2$:

$$v_2 = \begin{pmatrix} 0 \\ 0.4472 \\ 0.4472 \end{pmatrix}$$
$$\alpha_2 = v_2^T(Av_2 - Bv_1) = 3.2000$$
$$T_2 = \begin{pmatrix} \alpha_1 & \beta_1 \\ \beta_1 & \alpha_2 \end{pmatrix} = \begin{pmatrix} 1 & 2.2361 \\ 2.2361 & 3.2000 \end{pmatrix}$$

The eigenvalues of T_2 are -0.3920, 4.5919. The eigenvalues of (A, B) are 4.6013, -0.4347, and 0. (Note that the largest eigenvalue of T_2, 4.5919, is a reasonable

approximation of the largest eigenvalue 4.6013 of the pencil $(A - \lambda B)$.)

$i = 3$:

$$v_3 = \begin{pmatrix} 0 \\ -0.5477 \\ -0.3651 \end{pmatrix}$$

$$\alpha_3 = v_3^T(Av_3 - Bv_2) = -0.0333$$

Thus,

$$T_3 = T = \begin{pmatrix} \alpha_1 & \beta_1 & 0 \\ \beta_1 & \alpha_2 & \beta_2 \\ 0 & \beta_2 & \alpha_3 \end{pmatrix} = \begin{pmatrix} 1 & 2.2361 & 0 \\ 2.2361 & 3.2000 & 0.2449 \\ 0 & 0.2449 & -0.0333 \end{pmatrix}$$

The eigenvalues of T are -0.4347, 0.000, and 4.6013. The eigenpairs of T_2 are

$$\left(-0.3920, \begin{pmatrix} 0.8489 \\ -0.5285 \end{pmatrix}\right) \quad \text{and} \quad \left(4.5919, \begin{pmatrix} 0.5285 \\ 0.8489 \end{pmatrix}\right)$$

The *Ritz pairs* are:

$$\left(-0.3920, \begin{pmatrix} 0.8489 \\ -0.2363 \\ -0.2363 \end{pmatrix}\right), \qquad \left(4.5919, \begin{pmatrix} 0.5285 \\ 0.3796 \\ 0.3796 \end{pmatrix}\right)$$

9.10 The Lanczos Method for Eigenvalues of the Pencil $A - \lambda B$ in an Interval

In several important applications, such as in vibration and structural analysis, we are frequently interested in computing a specified number of eigenvalues in certain parts of the spectrum. We give a brief outline of a Lanczos algorithm for the generalized symmetric eigenvalue problem to do this. The algorithm was devised by Ericsson and Ruhe (1980).

ALGORITHM 9.10.1 The Lanczos Method for Eigenvalues in an Interval (the Ericsson–Ruhe Algorithm) Given (A, B), A symmetric and B symmetric positive definite, the following algorithm computes the generalized eigenvalues λ_k of the pencil $(A - \lambda B)$ in a specified interval $[\alpha, \beta]$.

Denote

$S(\nu, \delta)$ = the set of eigenvalues in (ν, δ)

$C(\nu, \delta)$ = the set of converged eigenvalues in (ν, δ)

Set

$$\mu_1 = \alpha, \qquad C = \phi \qquad \text{(null set)}$$

For $i = 1, 2, \ldots$ do until $|C(\alpha, \beta)| = |S(\alpha, \beta)|$

Step 1: Factor $(A - \mu_i B)$:

$$A - \mu_i B = LDL^T$$

Count the number of negative diagonal entries of D. Set that count equal to $|S(\alpha, \mu_i)|$.

Step 2: Choose a random vector v_1 such that $v_1^T \beta v_1 = 1$.

Step 3: Apply the symmetric Lanczos algorithm to the matrix

$$W^T(A - \mu_i B)^{-1} W$$

where

$$B = WW^T$$

with v_1 as the starting Lanczos vector.

Step 4: Let $\nu_1, \nu_2, \ldots, \nu_r$ be the converged eigenvalues from Step 3. Then add $\mu_i + 1/\nu_s$ to C, $s = 1, \ldots, r$.

Step 5: If $|C(\alpha, \mu_i)| < |S(\alpha, \mu_i)|$, then take a new vector v_1 orthogonal to converged eigenvectors and repeat steps 3–4. Else determine a new shift μ_{i+1}, and continue.

Remarks: Note that in applying the symmetric Lanczos to

$$W^T(A - \mu_i B)^{-1} W$$

we need a matrix–vector product of the form

$$y = W^T(A - \mu_i B)^{-1} Wx$$

which can be obtained as follows:

Step 1: Compute $z = Wx$

Step 2: Solve for p:

$$(A - \mu_i B)p = z$$

which can be done by making use of the factorization in Step 1 of the algorithm.

Step 3: Compute $y = W^T p$

For details of this algorithm, its implementation, and its proof, see the paper by Ericsson and Ruhe (1980).

9.11 Review and Summary

This chapter has been devoted to the study of the most commonly arising eigenvalue problem in engineering, namely, the generalized eigenvalue problem involving two matrices: $Ax = \lambda Bx$.

1. *Existence results:* Schur and real Schur analogues of the ordinary eigenvalue problem for the generalized eigenvalue problem exist (Theorems 9.2.1 and 9.3.1).
2. *The QZ algorithm:* The most widely used algorithm for the generalized eigenvalue problem is the QZ algorithm, which constructs the generalized real Schur form of (A, B). The algorithm comes in two stages. In stage I, the pair (A, B) is reduced to a Hessenberg triangular pair. In stage II, the Hessenberg triangular pair is further reduced to the generalized real Schur

form using implicit QR iteration to AB^{-1}. B^{-1} is never formed explicitly. The algorithm requires about $15n^3$ flops and is backward stable.

3. *The symmetric generalized eigenvalue problem:* Almost all eigenvalue problems arising in structural and vibration engineering are of the form

$$Kx = \lambda Mx$$

where M is symmetric positive definite and K is symmetric. This is called the symmetric definite generalized eigenvalue problem. Because of the importance of this problem, it was studied in some depth here.

Several case studies from vibration engineering were presented in Section 9.6 to show how this problem arises in important practical applications. These include

(a) Vibration of a free spring-mass system
(b) Vibration of a building
(c) Forced harmonic vibration of a spring-mass system

The natural frequencies and amplitudes of a vibrating system are related, respectively, to the eigenvalues and eigenvectors. If the frequency of the imposed periodic force becomes equal or nearly equal to one of the natural frequencies of the system, then resonance occurs, and the situation is quite alarming.

The collapses of the Tacoma Narrows Bridge in the state of Washington and of Broughton Bridge in England are widely believed to be related to such a phenomenon (see Chapter 8).

(a) The QZ method can, of course, be used to solve a symmetric definite generalized eigenvalue problem. However, both symmetry and definiteness will be lost, in general.
(b) A symmetry-preserving method is the Cholesky QR algorithm. This was described in Section 9.5.1. The accuracy obtained by this algorithm can be severely impaired if the matrix B in

$$Ax = \lambda Bx$$

is ill conditioned.

A variation of this method due to Wilkinson that computes the smallest eigenvalues with reasonable accuracy was described in this section. The method constructs an ordered real Schur form of B rather than the Cholesky factorization.

4. *Simultaneous diagonalization and applications:* The Cholesky QR algorithm for the symmetric definite problem

$$Ax = \lambda Bx$$

basically constructs a nonsingular matrix P that transforms A and B simultaneously to diagonal forms by congruence:

$$P^T AP = \text{a diagonal matrix}$$
$$P^T BP = I$$

This is called simultaneous diagonalization of A and B. In vibration and other engineering applications, this decomposition is called modal decomposition, and the matrix P is called a modal matrix.

The technique of simultaneous diagonalization is a very useful technique in engineering practice. Its applications include

(a) Decoupling of a second-order system of differential equations

$$M\ddot{y} + D\dot{y} + ky = 0$$

into n independent equations

$$\ddot{z}_i + (\alpha + \beta\omega_i^2)\dot{z}_i + \omega_i^2 z_i = 0, \qquad i = 1, 2, \ldots, n$$

if $DM^{-1}K = KM^{-1}D$.

(b) Making of a reduced-order model from a very large system of second-order systems.

Decoupling and model reduction are certainly very useful approaches for handling a large second-order system of differential equations. Unfortunately, however, the simultaneous diagonalization technique is not practical for large and sparse problems. On the other hand, many practical problems, such as the design of large space structures, give rise to very large and sparse symmetric definite eigenvalue problems.

The simultaneous diagonalization technique preserves symmetry but destroys the other exploitable properties offered by the data of the problem, such as sparsity. (Note that most practical large problems are sparse, and maintaining sparsity is a major concern to algorithm developers in order to economize the storage and computing requirements.)

5. *The quadratic eigenvalue problem:* The eigenvalue problem $(\lambda^2 M + \lambda D + K)x = 0$ is discussed in Section 9.8. We showed that the problem can be reduced either to a standard $2n \times 2n$ nonsymmetric problem (9.8.3) or to a $2n \times 2n$ symmetric generalized eigenvalue problem (9.8.7). We also showed how to extract the frequencies and modal damping ratios of a vibrating system governed by a second-order system once the eigenvalues are obtained (Equations 9.8.5 and 9.8.6).
6. *The Sturm sequence and Lanczos methods:* We gave an outline of the Sturm sequence method for the generalized eigenvalue problem with tridiagonal matrices A and B. The symmetric definite tridiagonal problems arise in several applications. We also gave a very brief description of the symmetric Lanczos method for a generalized eigenvalue problem.

The chapter concludes with a Lanczos algorithm due to Ericsson and Ruhe for finding a specified number of generalized eigenvalues in an interval of a symmetric definite generalized eigenvalue problem (Section 9.10).

7. *The sensitivity analysis of the generalized eigenvalue problem:* We have not discussed this aspect at all. Roughly speaking, if x and y are unit-length eigenvectors such that $Ax = \lambda Bx$ and $y^* = \lambda y^* B$, then λ is considered to be ill conditioned if

$$(y^* Ax)^2 + (y^* Bx)^2$$

is small. For details, see Stewart (1978).

EXERCISES

Section 9.2

1. Show that when A and B have a common null vector, then the generalized characteristic equation is identically zero:

$$\det(A - \lambda B) = 0$$

2. (a) Prove that if A and B are $n \times n$ matrices, then $\det(A - \lambda B)$ is a polynomial of degree at most n.
   (b) Prove that the degree of $\det(A - \lambda B)$ is equal to n if and only if B is nonsingular.

Sections 9.3 and 9.4

3. Show that computation of the matrix Q_1 in the initial QZ step depends only on the inversion of the 2×2 leading principal submatrix of the triangular matrix B.
4. Show that the shifts α_1 and α_2 in a QZ step, which are the eigenvalues of the lower 2×2 principal submatrix of $C = AB^{-1}$, can be computed without forming the complete B^{-1}. *Hint*: Computation depends only on the lower-right 3×3 submatrix of B^{-1}.
5. Using the implicit Q theorem, show that the matrix Q in the QZ step has the same first row as Q_1.
6. Work out the flop count for
   (a) the Hessenberg-triangular reduction in Section 9.3.1 with and without accumulations of the transforming matrices.
   (b) One iteration of the QZ algorithm (Algorithm 9.3.1).
   (c) Reduction of (A, B) to the generalized Schur form (Algorithm 9.3.2).
7. Work out an algorithm based on Gaussian elimination with partial pivoting for the Hessenberg triangular reduction. What is the flop count?
8. How are the eigenvectors of the pair (A, B) related to those of the orthogonally equivalent pair $(\tilde{A}, \tilde{B})$, where

$$\tilde{A} = UAV$$
$$\tilde{B} = UBV$$

Section 9.5

9. Rework the Cholesky QR algorithm (Algorithm 9.5.1) of the symmetric definite pencil by using the Schur decomposition of B:

$$V^T BV = D$$

10. Consider the Hessenberg triangular reduction of (A, B), with B singular. Show that in this case, if the dimension of the null space AB is k, then the Hessenberg triangular structure takes the form

$$A = \begin{pmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{pmatrix}, \qquad B = \begin{pmatrix} 0 & B_{12} \\ 0 & B_{22} \end{pmatrix}$$

where A_{11} is a $k \times k$ upper triangular matrix, A_{22} is upper Hessenberg, and B_{22} is $(n-k) \times (n-k)$ upper triangular and nonsingular. How does this help in the reduction process to the generalized Schur form?

11. Work out the flop count for the Cholesky QR algorithm of the symmetric definite pencil (Algorithm 9.5.1).
12. Let A and B be positive definite. Show that the algorithm for simultaneous diagonalization (Algorithm 9.5.3) requires about $7n^3$ flops.
13. (Generalized Orthogonal Iteration): Consider the following iterative algorithm, known as the generalized orthogonal iteration:
    (a) Choose an $n \times m$ orthonormal matrix Q_0 such that $Q_0^T Q_0 = I_{m \times m}$.
    (b) For $k = 1, 2, \ldots$, do
        (i) Solve for Z_k: $BZ_k = AQ_{k-1}$
        (ii) Find QR factorization of Z_k: $Z_k = Q_k R_k$

    Apply this algorithm to

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \\ 1 & 3 & 4 & 5 \\ 1 & 4 & 5 & 6 \end{pmatrix}, \qquad B = \begin{pmatrix} 10 & 1 & 1 & 1 \\ 1 & 10 & 1 & 1 \\ 1 & 1 & 10 & 1 \\ 1 & 1 & 1 & 10 \end{pmatrix}$$

14. Given

    (a) $A = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}, \qquad B = \begin{pmatrix} 10 & 1 & 0 \\ 1 & 10 & 1 \\ 0 & 1 & 10 \end{pmatrix}$

    (b) $A = \begin{pmatrix} 10 & 1 & 1 \\ 1 & 10 & 1 \\ 1 & 1 & 1 \end{pmatrix}, \qquad B = \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \end{pmatrix}$

    Find the eigenvalues and eigenvectors for each of the above pair using
    (i) The QZ iteration followed by inverse iteration (Algorithm 9.3.2 and Algorithm 9.4.1).
    (ii) The generalized Rayleigh quotient iteration (Algorithm 9.5.4).
    (iii) Techniques of simultaneous diagonalization (Algorithm 9.5.3).

Sections 9.6 through 9.9

15. Suppose that a bridge trestle has a natural frequency of 5 Hz (known from an earlier test). Suppose also that it deflects about 2 mm at midspan under a vehicle of 90,000 kg. What are the natural frequencies of the bridge and the vehicle?
16. For the equation of motion under the force $F(t) = F_1 e^{i\omega t}$,

$$m\ddot{y} + ky = F_1 e^{i\omega t}$$

    find the steady-state amplitude of vibration and determine the situation that can give rise to resonance.
17. Consider the following spring-mass problem:

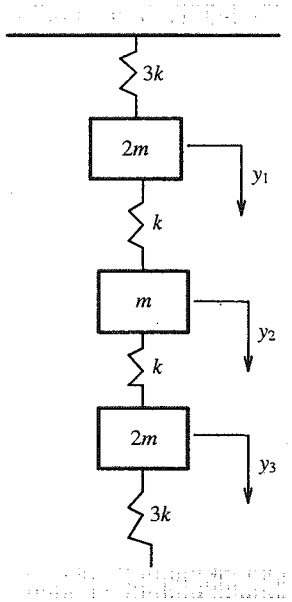


FIGURE 9.11

(a) Determine the equations of motion.
(b) Set up the generalized eigenvalue problem in the form

$$Kx = \lambda Mx$$

and then determine the natural frequencies and modes of vibration.
(c) Find the configuration of the system in each mode.

18. Consider the four-story building depicted in Figure 9.12:
Given

$$m_1 = 1.0 \times 10^5 \text{ kg}$$
$$m_2 = 0.8 \times 10^5 \text{ kg}$$
$$m_3 = 0.5 \times 10^5 \text{ kg}$$
$$m_4 = 0.6 \times 10^5 \text{ kg}$$
$$k_1 = 15 \times 10^8 \text{ N/m}, \; k_2 = 12 \times 10^8 \text{ N/m}$$
$$k_3 = 15 \times 10^8 \text{ N/m}, \; k_4 = 10 \times 10^8 \text{ N/m}$$

Find the maximum amplitude of each floor for a base horizontal displacement of 3 mm with a time period of 0.25 s, assuming zero intial conditions.

Hint: $\ddot{y}_o = \dfrac{-4\pi^2 y_o}{T^2}$

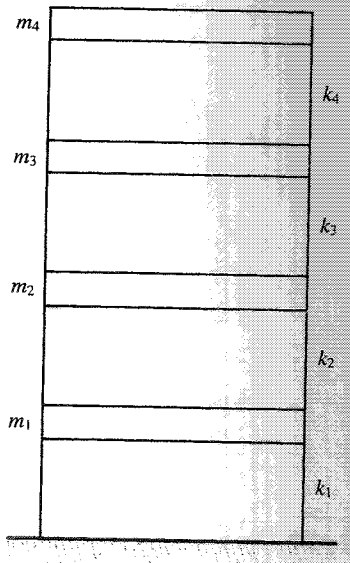


FIGURE 9.12

19. Consider the following diagram of an automobile suspension with vibration absorber (taken from the book *Linear Vibrations* by P. C. Müller and W. O. Schiehlen, 1985, 226).

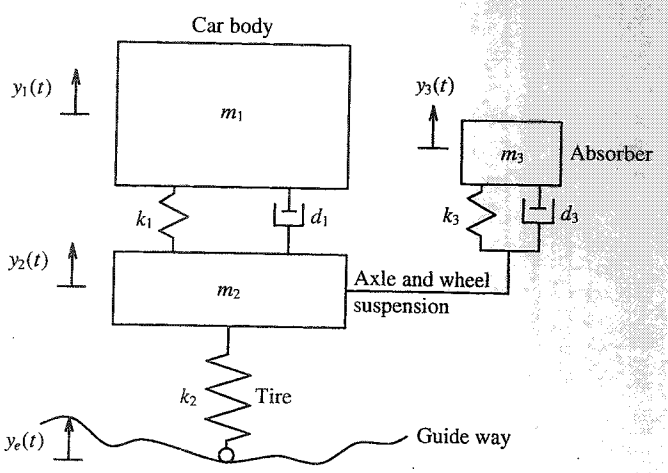


FIGURE 9.13 Automobile Suspension with Vibration Absorber

Given

$m_1 = 1200$ kg
$m_2 = 80$ kg
$m_3 = 20$ kg
$k_1 = 300$ N/cm
$k_2 = 3200$ N/cm
$k_3 = 600$ N/cm

Find the various amplitude frequency responses with different damping values of the absorber: $d_3 = 0, 300, 600, 1000$ Ns/m. (Assume that $d_1 = 0$.)

Sections 9.8 and 9.9

20. Find the eigenvalues of the quadratic pencil $(M\lambda^2 + K\lambda + D)x = 0$ using both Approach 1 and Approach 2 of Section 9.8, and compare the results.

$$M = \text{diag}(1, 2, 3), \qquad K = \begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{pmatrix}, \qquad D = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

21. Deduce the equations (9.8.5) and (9.8.6) for frequencies and the modal damping ratio.
22. Show that the polynomials defined by (9.9.1)–(9.9.3) form a Sturm sequence.
23. Find the eigenvalues for the pair (A, B), where

$$A = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix}, \qquad B = \begin{pmatrix} 10 & 1 & 0 \\ 1 & 10 & 1 \\ 0 & 1 & 10 \end{pmatrix}$$

using the Sturm sequence method described in Section 9.9.1.

24. Estimate the eigenvalues of the pair (A, B) of Exercise 13 using the symmetric Lanczos algorithm (Algorithm 9.9.1).
25. Find the eigenvalues of the pair (A, B) of Exercise 23 in the interval $(0, 0.3)$.
26. Prove that the eigenvalues of the symmetric definite pencil $A - \lambda B$ lie in the interval $[-\|B^{-1}A\|, \|B^{-1}A\|]$. Apply this result to the pair (A, B) in Exercise 23.
27. Consider the problem of finding λ's such that $\det(\lambda^k I + \lambda^{k-1}A_{k-1} + \cdots + A_0)$ is singular, where A_p, $p = 0, 1, \ldots$ are matrices. Show that this problem is equivalent to finding the eigenvalues of the block companion matrix

$$C = \begin{pmatrix} 0 & I & 0 & \cdots & \cdots & 0 \\ 0 & 0 & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & 0 \\ \vdots & & & \ddots & \ddots & 0 \\ \vdots & & & & 0 & I \\ -A_0 & -A_1 & \cdots & \cdots & \cdots & -A_{k-1} \end{pmatrix}$$

Verify the assertion by constructing an example of your own with three matrices A_0, A_1, and A_2 each of order 3 (use MATLAB).

MATLAB PROGRAMS AND PROBLEMS

Note: You will need **backsub** from MATCOM.

1. Write a MATLAB program called **hesstri** to reduce a pair of matrices (A, B) to a (Hessenberg-triangular) pair:

   $[H, T] =$ **hesstri**(A, B)

   Test your program by randomly generating A and B, each of order 15.
2. (a) Write a MATLAB program called **qzitri** to implement one iteration of the QZ algorithm:

   $[A1, B1] =$ **qzitri** (A, B)

   (b) Now apply one iteration of **qritrdsi**. (Double-shift implicit QR iteration from Chapter 8) to $C = AB^{-1}$:

   $[C] =$ **qritrdsi** (C)

   (c) Compute $D = A1 * B1^{-1}$, where $(A1, B1)$ is the Hessenberg triangular pair obtained in step (a).

   (d) Compare C and D to verify that they are the same.

Test Data

A = A 15×15 randomly generated unreduced upper Hessenberg matrix.
B = A 15×15, upper triangular matrix with all entries equal to 1, except five diagonal entries each equal to 10^{-5}.

3. Using **lynsyspp** from Chapter 6 or the MATLAB Command '\' and **hesstri**, write a MATLAB program called **invitrgn** to compute an eigenvector u corresponding to a given approximation to an eigenvalue λ of the pencil (A, B):

   $[u] =$ **invitrgn**(A, B, λ, V_o)

   Test your program using randomly generated matrices A and B, each of order 15, and then compare the result with that obtained by running the MATLAB command

   $[U, D] =$ **eig**(A, B)
4. (The purpose of this exercise is to compare the accuracy of different ways to find the eigenvalues and eigenvectors of a symmetric definite pencil $(K - \lambda M)$.)

   (a) Using the MATLAB commands **chol**, **inv**, **eig**, and '\' (or **backsub** from Chapter 3), write a MATLAB program called **cholqr**, to implement the Cholesky QR algorithm for the symmetric definite pencil $(K - \lambda M)$ (Algorithm 9.5.1).

   $[V1, D1] =$ **cholqr** (K, M)

   where $V1$ is a matrix containing the eigenvectors as its columns and $D1$ is a diagonal matrix containing the eigenvalues of the symmetric definite pencil $K - \lambda M$.

(b) Run **eig**(K, M) from MATLAB to compute $(V2, D2)$:

$$[V2, D2] = \textbf{eig}(K, M)$$

(c) Run **eig**$(inv(M) * K)$ from MATLAB to compute $(V3, D3)$:

$$[V3, D3] = \textbf{eig}\,(inv(M) * K)$$

(d) Compare the results of (a), (b), and (c).

Test data

$$M = \text{diag}(m, m, \ldots, m)_{50\times 50}$$

$$K = \begin{pmatrix} k & -k & 0 & 0 & \cdots & 0 \\ -k & 2k & -k & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & -k \\ 0 & \cdots & \cdots & 0 & -k & 2k \end{pmatrix}_{50\times 50}$$

$$m = 100$$

$$k = 10^8$$

5. Using **chol, inv, eig** from MATLAB, write a MATLAB program called **simdiag** to simultaneously diagonalize a pair of matrices (M, K), where M is symmetric positive definite and K is symmetric (Algorithm 9.5.3):

$$(I, D) = \textbf{simdiag}\,(M, K)$$

Test data

Use M and K from the case-study on the vibration of a building in Section 9.6.2.

6. Using **simdiag**, solve Example 9.7.1 from the book completely.
7. [The purpose of this exercise is to compare different approaches for solving the quadratic eigenvalue problem $(\lambda^2 M + \lambda D + K)x = 0$.]

(a) (Approach 1 of Section 9.8) Form the matrix

$$A = \begin{pmatrix} 0 & I \\ -M^{-1}K & -M^{-1}D \end{pmatrix}$$

by using MATLAB commands **inv**, **eig**, and **zeros**. Then compute the eigenvalues and eigenvectors of A using MATLAB command $[V_1, D_1] = \textbf{eig}(A)$.

(b) (Approach 2 of Section 9.8) Use the program **cholqr** to compute the eigenvalues

$$[V2, D2] = \textbf{cholqr}(A, B)$$

where

$$A = -\begin{pmatrix} D & K \\ K & 0 \end{pmatrix} \qquad B = \begin{pmatrix} M & 0 \\ 0 & K \end{pmatrix}$$

(c) Use the MATLAB command $[V3, D3] = eig(A, B)$, where the matrices A and B are given as in part (b).

Compare the results of these three approaches with respect to accuracy, flop count, and elapsed time. (The eigenvalues and the eigenvectors obtained by (c) can be taken as the accurate ones.)

Test data

Use the same M and K as in problem 4 and $D = 10^{-4} * M$.

8. Find the natural frequencies and modal damping ratios using results (9.8.5) and (9.8.6) of the book with the same M, K, and D as in Problem 7.
9. Write a MATLAB program, based on Algorithm 9.9.2, called **lansymgn** to transform the symmetric definite pair (A, B) to (TJ, IJ), where TJ is a $j \times j$ symmetric tridiagonal matrix and IJ is the $j \times j$ identity matrix, $j \leq n$.

$$[alpha, beta] = \textbf{lansymgn}\,(A, B)$$

where alpha is a vector containing the diagonal entries of TJ and beta is a vector containing the off-diagonal entries of TJ.

Test data and the experiment

Take $B = M$, $A = K$, where M and K are the same as given in Exercise 7. Run **lansymgn** using $j = 10, 20, 30, 40$, and 50. Tell how the extreme eigenvalues of TJ for each value of j approximate the eigenvalues of the pencil $(A - \lambda B)$. Find the eigenvalues using the MATLAB command

$$[U, D] = eig(A, B)$$

Note: Some of the programs you have been asked to write are in MATCOM.

Suggestions for Further Reading

Almost all books in vibration and structural engineering discuss implicitly or explicitly how the generalized eigenvalue problems arise in these applications. Some of the well-known books in the literature of vibration are:

Inman, Daniel J. *Vibration with Control, Measurement, and Stability*. Englewood Cliffs, N.J.: Prentice Hall, 1989.

Inman, Daniel J. *Engineering Vibration*. Englewood Cliffs, N.J.: Prentice Hall, 1994.

James, M. L., G. M. Smith, J. C. Wolford, and P. W. Whaley. *Vibration of Mechanical and Structural Systems*, 2d ed. New York: Harper & Row, 1989.

Müller, P. C., and W. O. Schiehlen. *Linear Vibrations*, Dordrecht, Netherlands: Martinus Nijhoff Publications, 1985.

Steidel, Robert F., Jr. *An Introduction to Mechanical Vibrations*, 2d ed. New York: Wiley, 1979.

Thomson, W. T. *Theory of Vibrations with Applications*, 4th ed. Englewood Cliffs, N.J.: Prentice Hall, 1992.

The QZ iteration algorithm has been discussed in some detail in the books by Golub and Van Loan (1989, Chapter 7), Stewart (1973, Chapter 7), and Watkins (1991, Chapter 5). (The original paper of Moler and Stewart (1973) is worth reading in this context.) Our treatment of this algorithm follows closely the one given in Stewart (1973).

For results on perturbation analysis of the generalized eigenvalue problem, see Stewart (1978, 1979b), Boley (1990), and Stewart and Sun (1990).

For convergence theory of Rayleigh quotient iteration with an arbitrary starting vector x_0, see Parlett (1980b, 319–320) and the recent paper of Pantazis and Szyld (1994). For applications of the symmetric definite generalized eigenvalue problem to earthquake engineering, see the book *Introduction to Earthquake Engineering* (Okamoto, 1984).

For applications of the generalized eigenvalue problem to the solution of the algebraic Riccati equations, see Arnold and Laub (1984), Pappas, Laub, and Sandell (1980), Van Dooren (1981a, 1981b), and Mehrmann (1988); for applications to other control problems, see Inman (1989).

A technique more efficient than the Cholesky QR iteration method for computing the generalized eigenvalues of a symmetric definite pencil for banded matrices has been proposed by Crawford (1973). The technique of Crawford is being implemented in the latest version of LAPACK. See also the recent work of Wang and Zhao (1991) and Kaufman (1993). For Geršgorin theory of the generalized eigenvalue problem, see Stewart (1975).

Williams and Laub (1992) have considered the simultaneous triangularizations of matrices M, D, and K of the second-order system $M\ddot{y} + D\dot{y} + Ky = 0$.

For aspects of balancing of the generalized eigenvalue problem, see Ward (1981).

Chapter 15 of Parlett's book *The Symmetric Eigenvalue Problem* (1980b) is a rich source of knowledge in the area of the generalized eigenvalue problems. See also Saad (1992), and Stewart (1976a).

The Lanczos Algorithm (Algorithm 9.9.1) needs to be reformulated in case B is singular. See Saad (1992, 298–299) for such a reformulated Lanczos algorithm (Algorithm 9.3). For block Lanczos algorithms for symmetric generalized eigenvalue problems, see Grimes, Lewis, and Simon (1994). There has been some further work on Algorithm 9.10.1 since it was proposed in 1980; notably in that the matrix B does not need to be factorized. See again Grimes, Lewis, and Simon (1994) for details. See also Ruhe (1994).

For more on generalized Schur decomposition of an arbitrary pencil, see Demmel and Kågström (1993).

The quadratic eigenvalue problems (QEP) have hardly been discussed in the books. However, many structural problems give rise to QEP. The normal practice is to ignore the damping matrix D or to assume the proportional damping, and then to solve the resulting generalized eigenvalue problems. For a Lanczos method to find a few extremal eigenvalues of a QEP, see Parlett and Chen (1990). Datta and Rincon (1993) have proposed an algorithm to stabilize a control system modeled by a second-order differential equation that does not require the explicit knowledge of eigenvalues and eigenvectors of the associated quadratic pencil. See also Datta (1994) in this context.

CHAPTER TEN

The Singular Value Decomposition (SVD)

10.1 Introduction

We have seen before that every matrix A can be can be decomposed into

$$A = U\Sigma V^*$$

where U and V are unitary and Σ is a diagonal matrix. If A is real, U and V are real orthogonal. *We will assume that A is real throughout this chapter.* This decomposition is called the *singular-value decomposition*, or the SVD of A. The SVD is closely related to the eigendecomposition of AA^T.

The SVD has a long and fascinating history. The names of at least five classical and celebrated mathematicians—E. Beltrami (1835–1899), C. Jordan (1838–1921), J. Sylvester (1814–1897), E. Schmidt (1876–1959), and H. Weyl (1885–1955)—can be associated with the development of the theory of the SVD. Some details of the contributions of these mathematicians to the SVD can be found in an interesting recent paper by G. W. Stewart (1993b). Also, the book by Horn and Johnson (1991) contains a nice history of the SVD.

In recent years the SVD has become a computationally viable tool for solving a wide variety of problems arising in many practical applications. The crux of using the SVD in these applications is in the fact that these applications require knowledge of the rank of a matrix, approximations of a matrix using matrices of lower rank, the orthonormal bases for the row and column spaces of a matrix as well as for their orthogonal complements, and orthogonal projections onto these subspaces. In most cases these computations need to be done under the presence of impurities in the data (called noise), and, as we will see in later sections in this chapter, the SVD is very effective for these computations. However, the ability to handle these problems effectively using the SVD certainly depends on how effectively the SVD can be computed. To this end, the applied scientific and engineering communities owe a great deal to contemporary numerical analysts such as Gene H. Golub, William Kahan, and C. Reinsch, who showed us how to compute the SVD in an efficient and numerically stable way.

This chapter is devoted to the study of the SVD. Important properties, some applications, and the existing computational methods will be discussed as follows: the SVD existence theorem (Section 10.2), a relationship between singular values and the eigenvalues (Section 10.3), an application of the SVD to the fetal ECG (Section 10.4), sensitivity of singular values (Section 10.5), applications of the SVD to rank-related problems, computing orthonormal bases, matrix approximations, and so on (Section 10.6), computing the variance-covariance matrix (Section 10.7), use of the SVD in least-squares and the pseudoinverse problems (Section 10.8), the Golub–Kahan–Reinsch algorithm and the Demmel–Kahan algorithm (Section 10.9), and the generalized SVD (Section 10.10).

The following major tools and concepts developed earlier will be needed for smooth learning and comprehension of this chapter's material: norm concepts and related results (Section 1.7), rank concept (Section 1.3.3), orthonormal bases (Sections 1.3.6 and 5.6) and projections (Sections 1.3.6 and 1.3.7), creating zeros with Householder and Givens matrices (Sections 5.4 and 5.5), QR factorization of a nonsquare matrix (Section 5.4.2), statements and methods for least-squares problems and the pseudoinverse (Chapter 7), basic results on eigenvalues and eigenvectors (Section 8.2), the Bauer–Fike theorem (Theorem 8.7.1) and general perturbation property of the symmetric eigenvalue problem (Theorem 8.11.2), and the implicit QR iteration (Section 8.9.6).

10.2 The Singular Value Decomposition Theorem

THEOREM 10.2.1 Let A be a real $m \times n$ matrix. Then there exist orthogonal matrices U and V such that

$$U^T A V = \begin{pmatrix} \Sigma_1 & 0 \\ 0 & 0 \end{pmatrix} = \Sigma \tag{10.2.1}$$

where Σ_1 is a nonsingular diagonal matrix. The diagonal entries of Σ are all nonnegative and can be arranged in nonincreasing order. The number of nonzero diagonal entries of Σ equals the rank of A.

Proof: Consider the matrix $A^T A$. It is an $n \times n$ symmetric positive semidefinite matrix; therefore its eigenvalues are nonnegative. Denote the eigenvalues of $A^T A$ by $\lambda_1 = \sigma_1^2, \lambda_2 = \sigma_2^2, \ldots, \lambda_n = \sigma_n^2$. Assume that these have been ordered such that $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$, and

$$\sigma_{r+1} = \sigma_{r+2} = \cdots = \sigma_n = 0$$

We have seen before that a symmetric matrix has a set of orthonormal eigenvectors (Theorem 8.2.8). Denote the set of orthonormal eigenvectors of $A^T A$ corresponding to λ_1 through λ_n by $v_1, \ldots, v_n$; that is, v_1 through v_n are orthonormal and satisfy

$$A^T A v_i = \sigma_i^2 v_i, \qquad i = 1, \ldots, n$$

Then,

$$v_i^T A^T A v_i = \sigma_i^2 \neq 0, \qquad i = 1, \ldots, r \tag{10.2.2}$$

and

$$v_i^T A^T A v_j = 0, \qquad i = 1, \ldots, r; j \neq i \tag{10.2.3}$$

Write

$$V_1 = (v_1, \ldots, v_r)$$
$$V_2 = (v_{r+1}, \ldots, v_n)$$

where v_1 through v_r are the eigenvectors associated with the nonzero eigenvalues λ_1 through λ_r and $v_{r+1}, \ldots, v_n$ correspond to the zero eigenvalues. Then

$$V_2^T A^T A V_2 = V_2^T A^T A(v_{r+1}, v_{r+2}, \ldots, v_n) = V_2^T(0, 0, \ldots, 0) = 0$$

This implies that $AV_2 = 0$, or

$$A v_k = 0, \qquad k = r+1, r+2, \ldots, n \tag{10.2.4}$$

Define now a set of nonzero vectors $\{u_i\}$ by

$$u_i = \frac{1}{\sigma_i} A v_i, \qquad i = 1, \ldots, r \tag{10.2.5}$$

The u_i's, $i = 1, \ldots, r$, then form an orthonormal set, because

$$\begin{aligned} u_i^T u_j &= \frac{1}{\sigma_i}(A v_i)^T \frac{1}{\sigma_j}(A v_j) \\ &= \frac{1}{\sigma_i \sigma_j}(v_i^T A^T A v_j) \\ &= \begin{cases} 0 & \text{when } i \neq j \\ 1 & \text{when } i = j \end{cases} \end{aligned} \tag{10.2.6}$$

Define $U_1 = (u_1, \ldots, u_r)$, and choose $U_2 = (u_{r+1}, \ldots, u_m)$ such that $U = (U_1, U_2)$ is orthogonal. Then, for any $k > r$, we have

$$u_k^T A v_i = \sigma_i u_k^T u_i = 0, \qquad i = 1, \ldots, r \quad \text{(Note that by 10.2.5, } A v_i = \sigma_i u_i.)$$

(by orthogonality of the vectors of U), and

$$u_k^T A v_i = 0, \qquad i = r+1, \ldots, n$$

(by 10.2.4).

Let $V = (V_1, V_2)$. Then

$$U^T A V = \begin{pmatrix} u_1^T \\ u_2^T \\ \vdots \\ u_m^T \end{pmatrix} A(v_1, \ldots, v_n) = \begin{pmatrix} \frac{1}{\sigma_1} v_1^T A^T \\ \frac{1}{\sigma_2} v_2^T A^T \\ \vdots \\ \frac{1}{\sigma_r} v_r^T A^T \\ u_{r+1}^T \\ \vdots \\ u_m^T \end{pmatrix} A(v_1, \ldots, v_n) \tag{10.2.7}$$

$$= \begin{pmatrix} \frac{1}{\sigma_1}\cdot\sigma_1^2 & & & 0 \\ & \frac{1}{\sigma_2}\cdot\sigma_2^2 & & \\ & & \ddots & \\ & & \frac{1}{\sigma_r}\cdot\sigma_r^2 & \\ 0 & & & 0 \end{pmatrix} = \Sigma = \begin{pmatrix} \Sigma_1 & 0 \\ 0 & 0 \end{pmatrix} \qquad (10.2.8)$$

where $\Sigma_1 = \text{diag}(\sigma_1, \ldots, \sigma_r)$

The statement about the rank is obvious, because rank (A) = rank $(U\Sigma V^T)$ = rank $(\Sigma) = r$. ■

The decomposition $A = U\Sigma V^T$ is known as the **singular value decomposition** (SVD) of A.

DEFINITION 10.2.1 The diagonal entries of the matrix Σ are called the **singular values** of A. The numbers $\sigma_1, \sigma_2, \ldots, \sigma_r$ are the positive singular values of A.

DEFINITION 10.2.2 The columns of U are called **left singular vectors** and those of V are called **right singular vectors**.

Uniqueness of the Singular Value Decomposition

There are $k = \min(m, n)$ singular values of A. Let r be the rank of A. Then there are r positive singular values. These are the positive square roots of the nonzero eigenvalues of A^TA (or AA^T). The remaining $(k - r)$, if $r < k$, singular values are zero. Thus, the singular values are unique. However, the singular vectors are not unique. For example, if A has a multiple singular value $\sigma > 0$, then the corresponding columns of the matrix V can be chosen as any orthonormal basis of the space spanned by the eigenvectors associated with the multiple eigenvalue $\lambda = \sigma^2$ of A^TA.

Example 10.2.1 Find the singular values and singular vectors of

$$A = \begin{pmatrix} 1 & 2 \\ 2 & 3 \\ 3 & 4 \end{pmatrix}$$

1. Compute the singular values. The eigenvalues of A^TA are 42.8600 and 0.1400. Therefore, the singular values are

$$\sigma_1 = \sqrt{42.8600} = 6.5468, \qquad \sigma_2 = \sqrt{0.1400} = 0.3742$$

2. Compute V. The matrix V_1 of the eigenvectors associated with the eigenvalues of A^TA is

$$V_1 = \begin{pmatrix} 0.5696 & -0.8219 \\ 0.8219 & 0.5696 \end{pmatrix}$$

Because $r = 2$, $V = V_1$.

3. Compute U

$$u_1 = \frac{1}{\sigma_1} Av_1 = \begin{pmatrix} 0.3381 \\ 0.5506 \\ 0.7632 \end{pmatrix} \qquad u_2 = \frac{1}{\sigma} Av_2 = \begin{pmatrix} 0.8480 \\ 0.1735 \\ -0.5009 \end{pmatrix}$$

Choose u_3 so that $U = (u_1, u_2, u_3)$ is unitary.

$$u_3 = \begin{pmatrix} 0.4082 \\ -0.8165 \\ 0.4082 \end{pmatrix}$$

4. The matrices U, Σ, and V defining the SVD of A are given by

$$U = \begin{pmatrix} 0.3381 & 0.8480 & 0.4082 \\ 0.5506 & 0.1735 & -0.8165 \\ 0.7632 & -0.5009 & 0.4082 \end{pmatrix}_{3\times 3}$$

$$V = \begin{pmatrix} 0.5696 & -0.8219 \\ 0.8219 & 0.5696 \end{pmatrix}_{2\times 2} \qquad \Sigma = \begin{pmatrix} 6.5468 & 0 \\ 0 & 0.3742 \\ 0 & 0 \end{pmatrix}$$

A Convention

For the remainder of this chapter we will assume, without any loss of generality, that $m \geq n$; because if $m < n$, we consider the SVD of A^T, and if the SVD of A^T is $U\Sigma V^T$, then the SVD of A is $V\Sigma^T U^T$.

We will follow the convention that the singular values are in nonincreasing order. Thus $\sigma_1 = \sigma_{\max}$ is the largest singular value and $\sigma_n = \sigma_{\min}$ the smallest singular value. Also, by $\sigma(A)$, we will denote the set of singular values of A.

10.3 A Relationship between the Singular Value Decomposition and Eigendecompositions

In the proof of Theorem 10.2.1 we used the fact that the singular values of A are the nonnegative square roots of the eigenvalues of A^TA (in fact, that's how they have been defined). The following theorem shows how the SVD of A is related to the eigendecompositions of AA^T and A^TA.

THEOREM 10.3.1 Let $A = U\Sigma V^T$ be the singular value decomposition of an $m \times n$ matrix A $(m \geq n)$. Let r be the rank of the matrix A. Then

1. $V^T(A^TA)V = \text{diag}(\sigma_1^2, \sigma_2^2, \ldots, \sigma_r^2, 0, \ldots, 0)_{(n\times n)}$
2. $U^T(AA^T)U = \text{diag}(\sigma_1^2, \sigma_2^2, \ldots, \sigma_r^2, 0, \ldots, 0)_{(m\times m)}$

Proof:

$$\begin{aligned} A^TA &= (V\Sigma^T U^T)(U\Sigma V^T) \\ &= V\Sigma^T\Sigma V^T \\ &= V\Sigma' V^T \end{aligned} \qquad (10.3.1)$$

where Σ' is an $n \times n$ diagonal matrix with $\sigma_1^2, \ldots, \sigma_r^2, 0, \ldots, 0$, as its diagonal entries. Thus,

$$V^T A^T A V = \Sigma' \tag{10.3.2}$$
$$= \operatorname{diag}(\sigma_1^2, \sigma_2^2, \ldots, \sigma_r^2, 0, \ldots, 0)_{n\times n}$$

Similarly,

$$AA^T = U\Sigma\Sigma^T U^T \tag{10.3.3}$$
$$= \operatorname{diag}(\sigma_1^2, \ldots, \sigma_r^2, 0, \ldots, 0)_{m\times m}$$

■

Note: From Theorem 10.3.1, it immediately follows that

1. The right singular vectors $v_1, v_2, \ldots, v_n$ are the eigenvectors of the matrix $A^T A$.
2. The left singular vectors $u_1, u_2, \ldots, u_m$ are the eigenvectors of AA^T.
3. $\sigma_1^2, \ldots, \sigma_r^2$ are the nonzero eigenvalues of both $A^T A$ and AA^T.

Example 10.3.1 Let

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

The SVD of A is

$$A = U\Sigma V^T = \begin{pmatrix} 0.5774 & -0.8165 & 0.0000 \\ 0.5774 & 0.4082 & -0.7071 \\ 0.5774 & 0.4082 & 0.7071 \end{pmatrix} \begin{pmatrix} 2.4495 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} 0.7071 & -0.7071 \\ 0.7071 & 0.7071 \end{pmatrix}$$

It is easily verified that $VA^T AV = \operatorname{diag}(6, 0)$, and $U^T AA^T U = \operatorname{diag}(6, 0, 0)$.

COROLLARY 10.3.1 Let A be a symmetric matrix with its eigenvalues $\lambda_1, \ldots, \lambda_n$. Then the singular values of A are $|\lambda_i|$, $i = 1, \ldots, n$.

Proof: Because

$$A = A^T \tag{10.3.4}$$
$$A^T A = A^2 \tag{10.3.5}$$

Therefore, from Theorem 10.3.1, we see that the n singular values of A are the nonnegative square roots of the n eigenvalues of A^2. Because the eigenvalues of A^2 are $\lambda_1^2, \ldots, \lambda_n^2$, we have the result. ■

COROLLARY 10.3.2 An $n \times n$ matrix A is nonsingular if and only if all its singular values are different from zero.

Proof: We know that $\det(A^T A) = (\det(A))^2$. Thus A is nonsingular if and only if $A^T A$ is nonsingular. Since a matrix is nonsingular if and only if its eigenvalues are all different from zero, the proof is now immediate from Theorem 10.3.1. ■

THEOREM 10.3.2 Let A be an $m \times n$ $(m \geq n)$ matrix and let C be the $(m+n) \times (m+n)$ matrix defined by

$$C = \begin{pmatrix} 0_{m\times m} & A \\ A^T & 0_{n\times n} \end{pmatrix}$$

Let $\sigma_1, \sigma_2, \ldots, \sigma_n$ be the singular values of A. Then the eigenvalues of C are $\sigma_1, \sigma_2, \ldots, \sigma_n, -\sigma_1, -\sigma_2, \ldots, -\sigma_n$, and zeros repeated $m - n$ times.

Proof: Let

$$A = U\Sigma V^T$$

Write

$$U = (\, U_{1(m\times n)} \quad U_{2(m\times(m-n))} \,)$$

$$\Sigma = \begin{pmatrix} \Sigma_{1(n\times n)} \\ 0_{(m-n)\times n} \end{pmatrix}$$

Define

$$P = \begin{pmatrix} \tilde{U}_1 & -\tilde{U}_1 & U_2 \\ \tilde{V} & \tilde{V} & 0_{n\times(m-n)} \end{pmatrix}$$

where

$$\tilde{U}_1 = \frac{1}{\sqrt{2}} U_1$$

and

$$\tilde{V} = \frac{1}{\sqrt{2}} V$$

Then it is easy to see that

$$P^T CP = \begin{pmatrix} \Sigma_1 & 0 & 0 \\ 0 & -\Sigma_1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

which shows that the eigenvalues of C are $\sigma_1, \ldots, \sigma_n, -\sigma_1, \ldots, -\sigma_n$, and zeros repeated $m - n$ times. ■

Example 10.3.2 Let

$$A = \begin{pmatrix} 1 & 2 \\ 2 & 3 \\ 3 & 4 \end{pmatrix}$$

Then, using the results of Example 10.2.1, we have

$$P = \begin{pmatrix} 0.2391 & 0.5996 & -0.2391 & -0.5996 & 0.4082 \\ 0.3894 & 0.1227 & -0.3894 & -0.1227 & -0.8165 \\ 0.5397 & -0.3542 & -0.5397 & 0.3542 & 0.4082 \\ 0.4028 & -0.5812 & 0.4028 & -0.5812 & 0 \\ 0.5812 & 0.4028 & 0.5812 & 0.4028 & 0 \end{pmatrix}$$

$$P^T CP = \operatorname{diag}(6.5468, 0.3742, -6.5468, -0.3742, 0).$$

The positive eigenvalues of C are 6.5468 and 0.3742. These are the nonzero singular values of A (see Example 10.2.1).

10.4 Applications of the SVD

The SVD has become an effective tool in handling various important problems arising in a wide variety of application areas, such as control theory, signal and image-processing, identification and estimation, speech synthesis, pattern recognition, time series analysis, electrical network theory, and biomedical engineering.

The aspects of control theory and identification problems requiring use of the SVD include problems on controllability and observability, realization of state-space models, the H-infinity control, balancing, robust feedback stabilization, model reduction, and so on.

In signal and speech processing the SVD can be regarded as a filter that produces an estimate of a signal from noisy data. For example, when a person speaks, speech is absent about 50% of the time. Thus, if there are background noises coming from a fan, a vibrating machine, and so on, then these disturbances dominate in the microphone signal when speech is absent. In such a situation the ratio of speech signal to the background noise has to be enhanced, and the SVD can be used effectively to do so.

In image-processing applications the SVD is routinely used for image compression, to remove noise in a picture, and so on. For details see Jain (1989).

In biomedical engineering the SVD plays an important role in obtaining a meaningful fetal ECG from that of the mother, see Vandewalle and De Moor (1988).

Furthermore, the SVD is the most effective tool in solving least squares and the generalized least-square problems. In this section we present a real-life application of the SVD in biomedical engineering.

Fetal ECG (Taken from Callaerts, De Moor, Vandewalle, and Sansen (1990)) Consider the problem of taking the electrocardiogram of a fetus by placing the cutaneous electrodes at the heart and the abdomen of the pregnant mother. The maternal ECG (MECG) clearly disturbs the observation of the fetal ECG (FECG), because the contributions of the maternal heart signals are much stronger than those of the fetal heart. The objective then will be to detect the FECG while simultaneously suppressing the MECG respective to noise.

Suppose there are p measurement signals $m_1(t), \ldots, m_p(t)$. Let these measurements be arranged in a vector called the measurement vector, $\boldsymbol{m}(t)$:

$$\boldsymbol{m}(t) = (m_1(t), m_2(t), \ldots, m_p(t))^T \tag{10.4.1}$$

Let there be r source signals $s_1(t), s_2(t), \ldots, s_r(t)$ arranged in the source signal vector, $\boldsymbol{s}(t)$:

$$\boldsymbol{s}(t) = (s_1(t), s_2(t), \ldots, s_r(t))^T \tag{10.4.2}$$

Obviously, the measurement signals are corrupted by an additive noise signal, and there exists a relationship between the measurement signals and the source signals. It can be assumed (Vandewalle and De Moor 1988) that this relationship is linear, and, indeed, each measurement signal $m_i(t)$ can be written as a linear combination of r source signals $s_i(t)$ and additive noise signal $n_i(t)$. This leads to the following equations:

$$\begin{aligned} m_1(t) &= t_{11}s_1(t) + t_{12}s_2(t) + \cdots + t_{1r}s_r(t) + n_1(t) \\ m_2(t) &= t_{21}s_1(t) + t_{22}s_2(t) + \cdots + t_{2r}s_r(t) + n_2(t) \\ &\vdots \\ m_p(t) &= t_{p1}s_1(t) + t_{p2}s_2(t) + \cdots + t_{pr}s_r(t) + n_r(t) \end{aligned} \tag{10.4.3}$$

or

$$\boldsymbol{m}(t) = T\boldsymbol{s}(t) + \boldsymbol{n}(t) \tag{10.4.4}$$

where

$$T = (t_{ij}), \qquad \text{and} \qquad \boldsymbol{n}(t) = (n_1(t), n_2(t), \ldots, n_r(t))^T \tag{10.4.5}$$

The matrix T is called the **transfer matrix** and depends upon the geometry of the body, the positions of the electrodes and sources, and the conductivities of the body tissues.

The problem now is to get an estimate of the source signals $\boldsymbol{s}(t)$ knowing only $\boldsymbol{m}(t)$ and, from that estimate, separate out the estimate of fetal source signals.

Let each measurement consist of q samples. Then the measurements can be stored in a matrix M of order $p \times q$. The matrix equation form of (10.4.4) is $M = TS + N$.

We now show that the SVD of M can be used to get estimates of the source signals. Let

$$M = U\Sigma V^T \tag{10.4.6}$$

be the SVD of M.

Then the $p \times q$ matrix $\hat{S}$ defined by

$$\hat{S} = U^T M \tag{10.4.7}$$

will contain p estimates of the source signals. Next, we need to extract the estimates of the fetal source signals from $\hat{S}$; let this be called $\hat{S}_F$.

Partition the matrix of singular values Σ of M as follows:

$$\Sigma = \begin{pmatrix} \Sigma_M & 0 & 0 \\ 0 & \Sigma_F & 0 \\ 0 & 0 & \Sigma_0 \end{pmatrix} \tag{10.4.8}$$

where Σ_M contains r_m large singular values associated with the maternal heart, Σ_F contains r_f singular values, those smaller ones associated with the fetal heart, and Σ_0 contains the remaining singular values associated with noise, and so on.

Let $U = (U_M, U_F, U_0)$ be a conformable partitioning of U. Then, obviously, we have

$$\begin{aligned} \hat{S} &= U^T M = \begin{pmatrix} U_M^T \\ U_F^T \\ U_0^T \end{pmatrix} M \\ &= \begin{pmatrix} U_M^T M \\ U_F^T M \\ U_0^T M \end{pmatrix} = \begin{pmatrix} \hat{S}_M \\ \hat{S}_F \\ \hat{S}_0 \end{pmatrix} \end{aligned} \tag{10.4.9}$$

Thus, $\hat{S}_F = U_F^T M$.

Once $\hat{S}_F$ is determined, we can also construct a matrix F containing only the contributions of the fetus in each measured signal, as follows:

$$F = U_F \hat{S}_F = \sum_{i=r_m+1}^{r_m+r_f} \sigma_i u_i v_i^T$$

where u_i and v_i are the ith column of U and V and σ_i is the ith singular value of M. The signals in $\hat{S}_F$ are called the principal fetal signals.

The previous method has been automated, and an on-line adaptive algorithm to compute the U-matrix has been designed. For the details of the method and test results, see the paper by Callaerts, De Moor, Vandewalle, and Sansen (1990).

Note that if the SVD of M is given by

$$\begin{aligned} M &= U\Sigma V^T \\ &= (U_1, U_2)\begin{pmatrix} \Sigma_1 & 0 \\ 0 & \Sigma_2 \end{pmatrix}\begin{pmatrix} V_1^T \\ V_2^T \end{pmatrix} \end{aligned}$$

then S can be estimated by $\hat{S} = U_1^T M$.

10.5 Sensitivity of the Singular Values

Because the squares of the singular values of A are just the eigenvalues of the real symmetric matrix A^TA and we have seen before that the eigenvalues of a symmetric matrix are insensitive to small perturbations, the same thing should be expected of the singular values as well.

Indeed, the following important result holds. The result is analogous to a result on the sensitivity of the eigenvalues of a symmetric matrix stated in Chapter 8 (Theorem 8.11.2). The result basically states that, like the eigenvalues of a symmetric matrix, *the singular values of a matrix are well-conditioned.*

THEOREM 10.5.1 Perturbation Theorem for Singular Values Let A and $B = A + E$ be two $m \times n$ matrices ($m \geq n$). Let σ_i, $i = 1, \ldots, n$ and $\tilde{\sigma}_i$, $i = 1, \ldots, n$ be, respectively, the singular values of A and B, in nonincreasing order. Then $|\tilde{\sigma}_i - \sigma_i| \leq \|E\|_2$, for each i.

Proof: The proof is based on the interesting relationship between the singular values of the matrix A and the eigenvalues of the symmetric matrix

$$\tilde{A} = \begin{pmatrix} 0 & A \\ A^T & 0 \end{pmatrix}$$

displayed in Theorem 10.3.2. It was shown there that the nonzero eigenvalues of $\tilde{A}$ are $\sigma_1, \ldots, \sigma_k, -\sigma_1, \ldots, -\sigma_k$, where σ_1 through σ_k are the nonzero singular values of A. The remaining eigenvalues of $\tilde{A}$ are, of course, zero. Define

$$\tilde{B} = \begin{pmatrix} 0 & B \\ B^T & 0 \end{pmatrix}, \qquad \tilde{E} = \begin{pmatrix} 0 & E \\ E^T & 0 \end{pmatrix}$$

We then have $\tilde{B} - \tilde{A} = \tilde{E}$.

The eigenvalues of $\tilde{B}$ and $\tilde{E}$ are related, respectively, to the singular values of B and E in the same way the eigenvalues of $\tilde{A}$ are related to the singular values of A. The result of Theorem (10.5.1) now follows immediately by applying Theorem 8.11.2 to $\tilde{B}$, $\tilde{A}$, and $\tilde{E}$. ■

Example 10.5.1

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{pmatrix}, \qquad E = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0.0002 \end{pmatrix} \tag{10.5.1}$$

The singular values of A are

$$\begin{aligned} \sigma_1 &= 14.5576 \\ \sigma_2 &= 1.0372 \\ \sigma_3 &= 0.0000 \end{aligned} \tag{10.5.2}$$

The singular values of $A + E$ are

$$\begin{aligned} \tilde{\sigma}_1 &= 14.5577 \\ \tilde{\sigma}_2 &= 1.0372 \\ \tilde{\sigma}_3 &= 0.0000 \\ \|E\|_2 &= 0.0002 \\ |\tilde{\sigma}_1 - \sigma_1| &\cong 10^{-4} < \|E\|_2 \\ |\tilde{\sigma}_2 - \sigma_2| &\cong < \|E\|_2 \\ |\tilde{\sigma}_3 - \sigma_3| &\cong < \|E\|_2 \end{aligned} \tag{10.5.3}$$

We now present a result (without proof) on perturbation of the singular values that uses the Frobenius norm instead of the 2-norm. The result of Theorem 10.5.2 is used later in deriving a result on the closeness of A to a matrix of lower rank (Theorem 10.6.2). Proof is similar to that of Theorem 10.5.1. See Horn and Johnson (1985, 419).

THEOREM 10.5.2 Let A, E, σ_i and $\tilde{\sigma}_i$, $i = 1, \ldots, n$, be the same as in Theorem (10.5.1). Then

$$\sqrt{\sum_{i=1}^{n} (\tilde{\sigma}_i - \sigma_i)^2} \leq \|E\|_F$$

10.6 The Singular Value Decomposition and the Structure of a Matrix

The SVD can be very effectively used to compute certain important properties relating to the structure of a matrix, such as its rank (in fact, nearness to rank-deficiency, which is an important practical issue), Euclidean norm, condition number, and orthonormal basis for the null space and range. We discuss these applications in this section.

10.6.1 Norms and the Condition Number

THEOREM 10.6.1 Let $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n$ be the n singular values of an $m \times n$ matrix A. Then

1. $\|A\|_2 = \sigma_1 = \sigma_{\max}$
2. $\|A\|_F = (\sigma_1^2 + \sigma_2^2 + \cdots + \sigma_n^2)^{\frac{1}{2}}$
3. $\|A^{-1}\|_2 = \dfrac{1}{\sigma_n} = \dfrac{1}{\sigma_{\min}}$, when A is $n \times n$ and nonsingular
4. If A is $n \times n$ and nonsingular, then $\text{Cond}_2(A) = \|A\|_2\|A^{-1}\|_2 = \dfrac{\sigma_1}{\sigma_n} = \dfrac{\sigma_{\max}}{\sigma_{\min}}$

Proof:

1. $\|A\|_2 = \|U\Sigma V^T\|_2$
   $= \|\Sigma\|_2 = \max_i(\sigma_i)$
   $= \sigma_1$
   (Note that the 2-norm and F-norm are invariant under orthogonal matrix products.)
2. $\|A\|_F = \|U\Sigma V^T\|_F$
   $= \|\Sigma\|_F$
   $= (\sigma_1^2 + \sigma_2^2 + \cdots + \sigma_n^2)^{\frac{1}{2}}$
3. Taking the SVD of A^{-1}, we note from Theorem 10.2.1 that the largest singular value of A^{-1} is $1/\sigma_n$. (Note that when A is invertible, $\sigma_n \neq 0$). Then the result follows from 1.
4. Follows from the definition of $\text{Cond}_2(A)$ and using 1 and 3. ■

Remark: For an $m \times n$ matrix A, we define $Cond_2(A) = \dfrac{\sigma_{\max}}{\sigma_{\min}}$. When A is rank deficient, $\sigma_{\min} = 0$, and we say that $\text{Cond}(A)$ is *infinite*.

Example 10.6.1

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 4 & 5 \\ 6 & 7 & 7.999 \end{pmatrix}$$

$\sigma_1 = 14.5570, \qquad \sigma_2 = 1.0375, \qquad \sigma_3 = 0.0001$

1. $\|A\|_2 = \sigma_1 = 14.5570$
2. $\|A\|_F = \sqrt{\sigma_1^2 + \sigma_2^2 + \sigma_3^2} = 14.5940$
3. $\text{Cond}(A) = \sigma_1/\sigma_3 = 1.0993 \times 10^5$.

10.6.2 Orthonormal Bases and the Projections

In Chapter 5 we showed how to construct orthonormal bases for the range and the null space of a matrix A using QR decomposition. In this section we show how the singular vectors can be used to construct these bases.

Let $A = U\Sigma V^T$ be the singular value decomposition of an $m \times n$ matrix. Let $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$ be the positive singular values of A. Then, from

Theorem 10.2.1 (the SVD theorem), we have

$$\begin{aligned} Av_i &= \sigma_i u_i, \qquad i = 1, \ldots, r \\ Av_i &= 0, \qquad i = r+1, \ldots, n \end{aligned} \tag{10.6.1}$$

Similarly, taking the SVD of $A^T = V\Sigma^T U^T$, we have

$$\begin{aligned} A^T u_i &= \sigma_i v_i, \qquad i = 1, \ldots, r \\ A^T u_i &= 0, \qquad i = r+1, \ldots, m \end{aligned}$$

Thus,

$$\begin{aligned} R(A) &= \text{span}\{u_1, \ldots, u_r\} \\ N(A) &= \text{span}\{v_{r+1}, \ldots, v_n\} \\ R(A^T) &= \text{span}\{v_1, \ldots, v_r\} \\ N(A^T) &= \text{span}\{u_{r+1}, \ldots, u_m\} \end{aligned}$$

> The columns of U corresponding to the nonzero singular values of A form an orthonormal basis of $R(A)$; the columns of U corresponding to the zero singular values form an orthonormal basis of $N(A^T)$, and so on.

Orthogonal Projections

Once the orthonormal bases for the range $R(A)$ and the null space $N(A)$ of A are obtained, the orthogonal projections can be easily computed. Thus, if we partition U and V as

$$U = (U_1, U_2), \qquad V = (V_1, V_2)$$

where U_1 and V_1 consist of the first r columns of U and V, then the following can be computed.

> **Computing Projections Using the SVD**
>
> 1. Projection onto $R(A) = U_1 U_1^T$
> 2. Projection onto $N(A) = V_2 V_2^T$
> 3. Projection onto $N(A^T) = U_2 U_2^T$
> 4. Projection onto $R(A^T) = V_1 V_1^T$

Example 10.6.2

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{pmatrix}$$

$\sigma_1 = 14.5576, \qquad \sigma_2 = 1.0372, \qquad \sigma_3 = 0$

$$U = \begin{pmatrix} 0.2500 & 0.8371 & 0.4867 \\ 0.4852 & 0.3267 & -0.8111 \\ 0.8378 & -0.4379 & 0.3244 \end{pmatrix}$$

$$V = \begin{pmatrix} 0.4625 & -0.7870 & 0.4082 \\ 0.5706 & -0.0882 & -0.8165 \\ 0.6786 & 0.6106 & 0.4082 \end{pmatrix}$$

An orthonormal basis for the null-space of A is

$$V_2 = \begin{bmatrix} 0.4082 \\ -0.8165 \\ 0.4082 \end{bmatrix}$$

An orthonormal basis for the range of A is

$$U_1 = \begin{bmatrix} 0.2500 & 0.8371 \\ 0.4852 & 0.3267 \\ 0.8370 & -0.4379 \end{bmatrix}$$

(Now compute the four projections yourself.)

MATCOM Note: MATCOM program ORTHPROJ computes the preceding projections.

10.6.3 The Rank and the Rank-Deficiency of a Matrix

Finding the rank of an $m \times n$ matrix and, in particular, determining the nonsingularity of a square matrix are very important tasks that frequently arise in linear algebra and many important applications. The most obvious and the least expensive way of determining the rank of a matrix is, of course, to triangularize the matrix using Gaussian elimination and then to find the rank of the reduced upper triangular matrix. Unfortunately, however, this is not a very reliable approach in floating-point arithmetic. In practice, it is more important, as we will see, to determine if the given matrix is near a matrix of a certain rank and, in particular, to know if a matrix of full rank is near a rank-deficient matrix. The Gaussian elimination method, which uses elementary transformations, may transform a rank-deficient matrix rank into one having full rank, due to numerical round-off error. For example, consider an $n \times n$ upper triangular matrix A with 1's along the main diagonal and -1 everywhere else. We have seen in Chapter 6 that $Cond_\infty(A) = n2^{n-1}$. Thus, A is very close to a singular matrix when n is large. However, by looking at this matrix one does not get any indication of that.

Another approach, which is more reliable than Gaussian elimination, for determining the rank and rank-deficiency is the *QR factorization with column pivoting*. As we saw in Chapter 5, this method constructs a permutation matrix P and orthogonal matrix Q such that

$$Q^T AP = \begin{pmatrix} R_1 & R_2 \\ 0 & 0 \end{pmatrix} \tag{10.6.2}$$

R_1 is upper triangular and nonsingular, and the dimension r of R_1 is the rank of A.

This result above, however, does not determine reliably the nearness of A to a rank-deficient matrix. For details, see our discussions on rank-revealing QR factorization in Chapter 5.

Because the number of *nonzero* singular values determines the rank of a matrix, we can say that a *matrix A is arbitrarily near a matrix of full rank*: Just change each zero singular value by a small number ϵ. It is, therefore, more meaningful to know if a matrix is near a matrix of a certain rank, rather than knowing what the rank is. The singular value decomposition exactly answers this question.

The most reliable way to determine the rank and nearness to rank-deficiency is to use the SVD.

Suppose that A has rank r, that is, $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$ and $\sigma_{r+1} = \cdots = \sigma_n = 0$. Then the question is how far A is from a matrix of rank $k < r$. The following theorems can be used to answer the question. Theorem 10.6.2 is generally known as the Eckart–Young theorem (Eckart and Young, 1939).

THEOREM 10.6.2 Let $A = U\Sigma V^T$ be the SVD of A, and let rank (A) be $r > 0$. Let $k \leq r$. Define

$$A_k = U\Sigma_k V^T$$

where

$$\Sigma_k = \left(\begin{array}{ccc|c} \sigma_1 & & 0 & \\ & \ddots & & 0 \\ 0 & & \sigma_k & \\ \hline & & 0 & 0 \end{array}\right), (\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_k > 0).$$

Then (1) rank $(A_k) = k$. (2). Let B be any other matrix of rank k, then $\|B - A\|_F \geq \|A_k - A\|_F$.

Proof: The statement about the rank is obvious, because

$$\text{rank}(A_k) = \text{rank}(U\Sigma_k V^T) = \text{rank}(\Sigma_k) = k$$

To prove the second part, we show that if B is any other matrix of rank k, then

$$\|B - A\|_F^2 \geq \|A_k - A\|_F^2$$

To see this, let's denote the singular values of B by $\phi_1 \geq \phi_2 \geq \cdots \geq \phi_n$. Then, because B has rank k, $\phi_{k+1} = \phi_{k+2} = \cdots = \phi_n = 0$. By Theorem 10.5.2, we then have

$$\|B - A\|_F^2 \geq \sum_{i=1}^{n} |\phi_i - \sigma_i|^2 \geq \sigma_{k+1}^2 + \cdots + \sigma_n^2 = \|A_k - A\|_F^2 \qquad \blacksquare$$

We now present a similar result using 2-norm. The line of proof is a bit different here than that of Theorem 10.6.2.

THEOREM 10.6.3 Let A and A_k be the same as in Theorem 10.6.2. Then

1. A_k has rank k.
2. The distance of A_k from A (in 2-norm): $\|A - A_k\|_2 = \sigma_{k+1}$.
3. Out of all the matrices of rank k, A_k is closest to A; that is,

$$\min_{\text{rank}(B)=k} \|A - B\|_2 = \|A - A_k\|_2$$

Proof of 1: It is obvious that rank $(A_k) = k$; because rank $(A_k) = \text{rank}(U\Sigma_k V^T) = \text{rank}(\Sigma_k) = k$ (note that $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_k > 0$).

Proof of 2: Because $A - A_k = U(\Sigma - \Sigma_k)V^T$, we have $\|A - A_k\|_2 = \|U(\Sigma - \Sigma_k)V^T\|_2 = \|(\Sigma - \Sigma_k)\|_2 = \sigma_{k+1}$. Thus, the distance between A and A_k is σ_{k+1}.

Proof of 3: To prove (3), we show that if $B \in \mathbb{R}^{m\times n}$ is any other matrix of rank k, then $\|A - B\|_2 \geq \sigma_{k+1}$; that is A_k is closest to A among all other matrices of rank k. Since B has rank k, the null space of B, $N(B)$ has dimension $n - k$. Consider now the space $S = \text{span}\{v_1, \ldots, v_{k+1}\}$, where v_1 through v_{k+1} are the right singular vectors of A. Since $N(B)$ and S are both subspaces of $\mathbb{R}^n$ and the sum of their dimensions is greater than n, their intersection must be nonempty. Let z be a unit vector lying in this intersection. Then, since $z \in \text{span}\{v_1, \ldots, v_{k+1}\}$, there exist scalars (not all zero) such that $z = c_1v_1 + c_2v_2 + \cdots + c_{k+1}v_{k+1}$.

Furthermore, because $v_1, \ldots, v_{k+1}$ are orthonormal, we must have $|c_1|^2 + |c_2|^2 + \cdots + |c_{k+1}|^2 = 1$. Because z also belongs to $N(B)$, we have $Bz = 0$. So

$$(A - B)z = Az = \sum_{i=1}^{k+1} c_iAv_i = \sum_{i=1}^{k+1} \sigma_i c_i u_i$$

Because $u_1, \ldots, u_{k+1}$ are also orthonormal,

$$\|(A - B)z\|_2^2 = \sum_{i=1}^{k+1} |\sigma_i c_i|^2 \geq \sigma_{k+1}^2 \sum_{i=1}^{k+1} |c_i^2| = \sigma_{k+1}^2$$

Thus, $\|A - B\|_2 \geq \dfrac{\|(A - B)z\|_2}{\|z\|_2} \geq \sigma_{k+1}$ (because $\|z\|_2 = 1$). ■

COROLLARY 10.6.1 Let A be an $m \times n$ matrix of full rank and let $r = \min(m, n)$. Let $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$ be the singular values of A. Let C be another $m \times n$ matrix such that $\|C - A\|_2 < \sigma_r$. Then C has also full rank.

COROLLARY 10.6.2 The relative distance of a nonsingular matrix A to the nearest singular matrix C is $\dfrac{1}{\text{Cond}_2(A)}$; that is,

$$\frac{\|C - A\|_2}{\|A\|_2} = \frac{1}{\text{Cond}_2(A)}$$

Example 10.6.3 Let

$$A = \begin{pmatrix} 1 & 2 \\ 2 & 3 \\ 3 & 4 \end{pmatrix}, \qquad \sigma_1 = 6.5468, \qquad \sigma_2 = 0.3742$$

Let k be 1. Then

$$A_1 = U\Sigma_1V^T = \begin{pmatrix} 1.2608 & 1.8193 \\ 2.0534 & 2.9630 \\ 2.8460 & 4.1067 \end{pmatrix}$$

Out of all the matrices of rank 1, A_1 is the closest to A.

Distance of a Matrix from the Nearest Matrix of Lower Rank

The preceding result states that the smallest nonzero singular value gives the distance from A to the nearest matrix of lower rank. In particular, for a nonsingular $n \times n$ matrix A, σ_n gives the measures of the distance of A to the nearest singular matrix.

Thus, in order to know if a matrix A of rank r is close enough to a matrix of lower rank, look into the smallest nonzero singular value σ_r. If this value is very small, then the matrix is very close to a matrix of rank $r - 1$, because there exists a perturbation of size as small as $|\sigma_r|$ that will produce a matrix of rank $r - 1$. In fact, one such perturbation is $u_r\sigma_r v_r^T$.

Example 10.6.4

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0.0000004 \end{pmatrix}$$

$$\text{rank}(A) = 3, \qquad \sigma_3 = 0.0000004$$

$$U = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$V = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$A' = A - u_3\sigma_3v_3^T = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

$$\text{rank}(A') = 2$$

The required perturbation $u_3\sigma_3v_3^T$ to make A singular is very small:

$$10^{-6}\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0.4 \end{pmatrix}$$

Approximations by Orthogonal Matrices

Let A be an $n \times n$ matrix. In this section we consider the problem of finding an orthogonal matrix Q such that

$$\|A - Q\|_F$$

is minimized.

It can be shown that if

$$A = U\Sigma V^T$$

is the singular value decomposition of A, then

$$Q = UV^T$$

A generalization of this problem arises in factor analysis in statistics. Given $n \times n$ real matrices A and B, find an orthogonal matrix Q such that

$$\|A - BQ\|_F$$

is minimized.

The SVD again can be used to solve the problem. It can be shown (Green 1952) that if

$$B^T A = U\Sigma V^T$$

is the SVD of $B^T A$, then

$$Q = UV^T$$

For a further generalization of the problem, see Rao (1980).

10.6.4 An Outer-Product Expansion of A and Its Consequences

In many applications the matrix size is so large that the storage of A may require a major part of total available memory space.

The knowledge of nonzero singular values and the associated left and right singular vectors can help store the matrix with a substantial amount of savings in storage space.

Let A be $m \times n$ ($m \geq n$), and $\sigma_1, \ldots, \sigma_r$ be the nonzero singular values of A. Let the associated right and left singular vectors be $v_1, \ldots, v_r$, and $u_1, \ldots, u_r$, respectively. Then the following theorem shows that A can be written as the sum of r matrices, each of rank 1. Because a rank-one matrix of order $m \times n$ can be stored using only $(m + n)$ storage locations rather than mn locations needed to store an arbitrary matrix, this will result in a very substantial savings if r is substantially less than n.

THEOREM 10.6.4 Let A be an $m \times n$ matrix and let $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$ be the r nonzero singular values of A. Then

$$A = \sum_{j=1}^{r} \sigma_j u_j v_j^T$$

Proof: The proof follows immediately from the relation

$$A = U\Sigma V^T$$

noting that σ_{r+1} through σ_n are zero. ■

DEFINITION 10.6.1 The representation of $A = \sum_{j=1}^{r} \sigma_j u_j v_j^T$ is called the outer-product expansion of A.

Example 10.6.5

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{pmatrix}$$

$$\sigma_1 = 14.5576$$
$$\sigma_2 = 1.0372$$
$$\sigma_3 = 0, \qquad r = 2$$

$$u_1 = \begin{pmatrix} 0.2500 \\ 0.4852 \\ 0.8379 \end{pmatrix}, \qquad u_2 = \begin{pmatrix} 0.8371 \\ 0.3267 \\ -0.4389 \end{pmatrix}$$

$$v_1 = \begin{pmatrix} 0.4625 \\ 0.5706 \\ 0.6786 \end{pmatrix}, \qquad v_2 = \begin{pmatrix} -0.7870 \\ -0.0882 \\ 0.6106 \end{pmatrix}$$

$$\sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{pmatrix}$$

An Important Consequence

Theorem 10.6.4 has an important practical consequence. In practical applications, the matrix A generally has a large number of small singular values. Suppose there are $(n - k)$ small singular values of A that can be neglected. Then the matrix $A_k = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \cdots + \sigma_k u_k v_k^T$ is a very good approximation of A, and such an approximation can be adequate in applications. For example, in *digital image processing*, even when k is chosen much less than n, the digital image corresponding to A_k can be very close to the original image. For details see Andrews and Hunt (1977). If A is $n \times n$, the storage of A_k will require $(2nk + k)$ locations, compared with n^2 locations needed to store A, resulting in a large amount of savings.

10.6.5 Numerical Rank

As we just saw, in practical applications that need singular values, we have to know when to accept a computed singular value to be "zero." Of course, if it is of the order of "roundoff zeros" (machine epsilon), we can declare it to be zero.

However, if the data shares large relative error, it should also be taken into consideration. A practical criterion would be the following:

Accept a computed singular value to be zero if it is less than or equal to $10^{-t}\|A\|_\infty$, *where the entries of A are correct to t digits*

Having defined a tolerance $\delta = 10^{-t}\|A\|_\infty$ for a zero singular value, we can make the following convention (Golub and Van Loan 1989, 247):

A has "numerical rank" r if the computed singular values $\tilde{\sigma}_1, \tilde{\sigma}_2, \ldots, \tilde{\sigma}_n$ satisfy

$$\tilde{\sigma}_1 \geq \tilde{\sigma}_2 \geq \cdots \geq \tilde{\sigma}_r > \delta \geq \tilde{\sigma}_{r+1} \geq \cdots \geq \tilde{\sigma}_n \tag{10.6.4}$$

Thus, to determine the numerical rank of a matrix A, count the "large" singular values only. If this number is r, then A has numerical rank r.

Remark: Note that finding the numerical rank of a matrix will be "tricky" if there is no suitable gap between a set of singular values.

10.7 Computing the Variance-Covariance Matrix

In Chapter 7 we described a method to compute the variance-covariance matrix $(A^TA)^{-1}$ using the QR factorization of A. We note here this matrix can also be computed immediately once the singular values and the right singular vectors of A have been computed.

Computing $(A^TA)^{-1}$ Using the SVD

Let

$$A = U\Sigma V^T$$

be the SVD of A; then the entries of the matrix $C = (A^TA)^{-1} = (c_{ij})$ are given by

$$c_{ij} = \sum_{k=1}^{n} \frac{v_{ik}v_{jk}}{\sigma_k^2}$$

where $n = \text{rank}(A)$. (Note that $C = (A^TA)^{-1} = V(\Sigma^T\Sigma)^{-1}V^T$.)

Example 10.7.1

$$A = \begin{pmatrix} 1 & 2 \\ 2 & 3 \\ 3 & 4 \end{pmatrix}$$

$$c_{11} = \frac{v_{11}^2}{\sigma_1^2} + \frac{v_{12}^2}{\sigma_2^2} = 4.833$$

$$c_{12} = \frac{v_{11}v_{21}}{\sigma_1^2} + \frac{v_{12}v_{22}}{\sigma_2^2} = -3.3333$$

$$c_{22} = \frac{v_{21}^2}{\sigma_1^2} + \frac{v_{22}^2}{\sigma_2^2} = 2.3333$$

MATCOM Note: MATCOM program COVSVD implements the above method for computing the variance-covariance matrix.

10.8 The Singular Value Decomposition, the Least-Squares Problem, and the Pseudoinverse

In Chapter 7 we have seen that the full-rank linear least squares problem can be effectively solved using the QR decomposition, but pivoting will be needed to handle the rank-deficient case.

We have just seen that the singular values of A are needed to determine reliably the nearness to rank deficiency. We will now see that the SVD is also an effective tool to solve the least-squares problem, both in the full rank and rank-deficient cases.

10.8.1 The SVD and Least-Squares Problem

We briefly discussed how to solve a least-squares problem using the SVD in Chapter 7. Here we give a full treatment.

Consider the least-squares problem once more: Find x such that $\|r\|_2 = \|Ax - b\|_2$ is minimum.

Let $A = U\Sigma V^T$ be the SVD of A. Then we have

$$\begin{aligned} \|r\|_2 &= \|(U\Sigma V^T x - b)\|_2 \\ &= \|U(\Sigma V^T x - U^T b)\|_2 \\ &= \|\Sigma y - b'\|_2 \end{aligned}$$

where $V^Tx = y$ and $U^Tb = b'$. *Thus, the use of the SVD of A reduces the least-squares problem for a full matrix A to one with a diagonal matrix Σ:*

Find y such that

$$\|\Sigma y - b'\|_2$$

is minimum.

The reduced problem is trivial to solve. We have

$$\|\Sigma y - b'\|_2 = \sum_{i=1}^{k} |\sigma_i y_i - b_i'|^2 + \sum_{i=k+1}^{m} |b_i'|^2$$

where k is the number of nonzero singular values of A. Thus, the vector

$$y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

that minimizes $\|\Sigma y - b'\|_2$ is given by:

$$y_i = \begin{cases} \dfrac{b_i'}{\sigma_i}, & \text{if } \sigma_i \neq 0 \\ \text{arbitrary}, & \text{if } \sigma_i = 0 \end{cases}$$

(Note that when $k < n$, y_{k+1} through y_n do not appear in the preceding expression and, therefore, do not have any effect on the residual.) Of course, once y is computed, the solution can be recovered from $x = Vy$.

Because corresponding to each "zero" singular value σ_i, y_i can be set arbitrarily, *in the rank-deficient case, we will have infinitely many solutions to the least-squares problem.* There are instances where this rank deficiency is actually desirable because it provides a rich family of solutions that might be used for optimizing some other aspect of the original problem. In the full-rank case, the least-squares solution is, of course, unique.

The preceding discussion can be summarized in the following algorithm.

ALGORITHM 10.8.1 Least-Squares Solutions Using the SVD

Step 1: Find the SVD of A:

$$A = U\Sigma V^T$$

Step 2: Form $b' = U^T b = \begin{pmatrix} b_1' \\ b_2' \\ \vdots \\ b_m' \end{pmatrix}$

Step 3: Compute $y = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}$, choosing

$$y_i = \begin{cases} \dfrac{b_i'}{\sigma_i}, & \text{when } \sigma_i \neq 0 \\ \text{arbitrary}, & \text{when } \sigma_i = 0 \end{cases}$$

Step 4: Compute the family of least-squares solutions:

$$x = Vy$$

(Note that in the full-rank case, the family has just one member.)

Flop count. Using the Golub–Kahan–Reinsch method to be described later, it takes about $2mn^2 + 4n^3$ flops to solve the least-squares problem when A is $m \times n$ and $m \geq n$. (In deriving this flop count, note that the complete vector b' does not need to be computed; only the columns of U that correspond to the nonzero singular values are needed in computation.)

An Expression for the Minimum-Norm Least-Squares Solution

It is clear from Step 3 that in the rank-deficient case, the minimum 2-norm least-squares solution is the one that is obtained by setting $y_i = 0$, whenever $\sigma_i = 0$. Thus, from before, we have the following expression for the minimum 2-norm solution:

Minimum 2-Norm Least-Squares Solution Using the SVD

$$x = \sum_{i=1}^{k} \frac{u_i^T b}{\sigma_i} v_i \tag{10.8.1}$$

where k = numerical rank(A) and u_i and v_i are respectively the ith columns of U and V.

Example 10.8.1

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 1 & 2 & 3 \end{pmatrix}, \qquad b = \begin{pmatrix} 6 \\ 9 \\ 6 \end{pmatrix}$$

1. $\sigma_1 = 7.5358$
   $\sigma_2 = 0.4597$
   $\sigma_3 = 0$
   A is rank deficient.

$$U = \begin{pmatrix} 0.4956 & 0.5044 & 0.7071 \\ 0.7133 & -0.7008 & -0.0000 \\ 0.4956 & 0.5044 & -0.7071 \end{pmatrix}$$

$$V = \begin{pmatrix} 0.3208 & -0.8546 & 0.4082 \\ 0.5470 & -0.1847 & -0.8165 \\ 0.7732 & 0.4853 & 0.4082 \end{pmatrix}$$

2. $b' = U^T b = (12.3667, -0.2547, 0)^T$
3. $y = (1.6411, -0.5541, 0)$
4. The minimum 2-norm least-squares solution is $Vy = (1, 1, 1)^T$.

Computational Remarks: Let A have rank $r \leq \min(m, n)$. Then, computing least-squares solutions using the SVD of A via Algorithm 10.8.1, we need to compute only y_1 through y_r, which can be computed using only the first r columns of U. The whole U, therefore, does not need to be computed in such a case. Similarly, in computing the minimum-norm solution, only the first r columns of V are needed.

Furthermore, as we will see in Section 10.9.1, the matrices U and V are computed as the products of Householder and Givens matrices, so the vector b' can be computed by updating the product of b with a Householder matrix or a Givens matrix as soon as the latter is computed, thus completely avoiding the explicit formation of U.

These remarks should be taken into consideration in practical computations, because a substantial savings might result. Solving a least-squares problem using the SVD is much more expensive than using the QR decomposition or any other method discussed in Chapter 7, no matter what savings are made in computing in U and/or V (see Section 10.9.3).

Also note that, in practical computations, r should be taken as the numerical rank of A as determined by the criterion given in Section 10.6.5.

MATCOM Note: Algorithm 10.8.1 has been implemented in MATCOM program LSQRSVD. Also, MATCOM has a program called MINNMSVD to compute the minimum 2-norm solution using the SVD.

10.8.2 Solving a Linear System Using Singular Value Decomposition

The idea of using the SVD in the solution of the least-squares problem can be easily applied to determine if a linear system $Ax = b$ has a solution and, if so, how to compute it.

Thus, if

$$A = U\Sigma V^T$$

then

$$Ax = b$$

is equivalent to

$$\Sigma y = b'$$

where $y = V^T x$ and $b' = U^T b$.

So, the system $Ax = b$ is consistent if and only if the diagonal system $\Sigma y = b'$ is consistent (which is trivial to check), and a solution of $Ax = b$ can be computed by solving the diagonal system $\Sigma y = b'$ first and then recovering x from $x = Vy$. However, this approach will be much more expensive than the Gaussian elimination and QR decomposition methods. This cost is why the SVD is not used in practice, in general, to solve a linear system.

10.8.3 The SVD and the Pseudoinverse

In Chapter 7 we saw that when A is an $m \times n$ ($m \geq n$) matrix having full rank, the pseudoinverse of A is given by

$$A^\dagger = (A^T A)^{-1} A^T$$

A formal definition of the pseudoinverse of any matrix A (whether it has full rank or not) can be given as follows:

Four Properties of the Pseudoinverse

The pseudoinverse of an $m \times n$ matrix A is an $n \times m$ matrix X satisfying the following conditions:

1. $AXA = A$
2. $XAX = X$
3. $(AX)^T = AX$
4. $(XA)^T = XA$

The pseudoinverse of a matrix always exists and is unique. We now show that the SVD provides a nice expression for the pseudoinverse.

Let $A = U\Sigma V^T$ be the SVD of A; then it is easy to verify that the matrix

$$V\Sigma^\dagger U^T, \quad \text{where}$$
$$\Sigma^\dagger = \operatorname{diag}\left(\frac{1}{\sigma_1}, \frac{1}{\sigma_2}, \ldots, \frac{1}{\sigma_r}, 0, \ldots, 0\right) \in \mathbb{R}^{n \times m} \tag{10.8.2}$$

satisfies all the four conditions and is, therefore, the pseudoinverse of A. Note that this expression for the pseudoinverse coincides with A^{-1} when A is nonsingular, because

$$\begin{aligned} A^{-1} &= (A^T A)^{-1} A^T \\ &= (V\Sigma^T U^T U \Sigma V^T)^{-1} V \Sigma^T U^T \\ &= V\Sigma^{-1}(\Sigma^T)^{-1} V^T V \Sigma^T U^T \\ &= V\Sigma^{-1} U^T \end{aligned} \tag{10.8.3}$$

(Note that in this case, $\Sigma^\dagger = \Sigma^{-1}$.)

The process for computing the pseudoinverse $A^\dagger$ of A using the SVD of A can be summarized as follows.

ALGORITHM 10.8.2 Computing the Pseudoinverse Using the SVD

Step 1: Find the SVD of A:

$$A = U\Sigma V^T$$

Step 2: Compute

$$\Sigma^\dagger = \left(\begin{array}{cccc|c} \frac{1}{\sigma_1} & & & & \\ & \frac{1}{\sigma_2} & & & \\ & & \ddots & & 0 \\ & & & \frac{1}{\sigma_r} & \\ \hline & & & 0 & 0 \end{array}\right)$$

where $\sigma_1, \ldots, \sigma_r$ are the r nonzero singular values of A.

Step 3: Compute $A^\dagger = V\Sigma^\dagger U^T$

Example 10.8.2 The SVD of a matrix A is given by:

$$A = \begin{pmatrix} 0 & 0 & -1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} \frac{1}{3} & -\frac{6}{9} & -\frac{6}{9} \\ -\frac{6}{9} & \frac{1}{3} & -\frac{6}{9} \\ -\frac{6}{9} & -\frac{6}{9} & \frac{1}{3} \end{pmatrix}$$

Then,

$$\Sigma^\dagger = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

$$U^T = \begin{pmatrix} 0 & -1 & 0 \\ 0 & 0 & -1 \\ -1 & 0 & 0 \end{pmatrix}$$

$$V^T = V$$

Thus, the pseudoinverse $A^\dagger = V\Sigma^\dagger U^T$ will be

$$\begin{aligned} A^\dagger &= \begin{pmatrix} \frac{1}{3} & -\frac{6}{9} & -\frac{6}{9} \\ -\frac{6}{9} & \frac{1}{3} & -\frac{6}{9} \\ -\frac{6}{9} & -\frac{6}{9} & \frac{1}{3} \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & -1 & 0 \\ 0 & 0 & -1 \\ -1 & 0 & 0 \end{pmatrix} \\ &= \begin{pmatrix} 0 & -\frac{1}{3} & \frac{1}{3} \\ 0 & \frac{2}{3} & -\frac{1}{6} \\ 0 & \frac{2}{3} & \frac{1}{3} \end{pmatrix} \end{aligned}$$

The Pseudoinverse and the Least-Squares Problem

From (10.8.1) and (10.8.2), it follows that the minimum 2-norm solution of the linear least-squares problem obtained through the SVD in the previous section is

$$A^{\dagger}b$$

where $A^{\dagger}$ is the pseudoinverse of A.

Example 10.8.3

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 3 \\ 1 & 2 & 3 \end{pmatrix}, \qquad b = \begin{pmatrix} 6 \\ 8 \\ 6 \end{pmatrix}$$

$$A^{\dagger} = \begin{pmatrix} -0.9167 & 1.3333 & -0.9167 \\ -0.1667 & 0.3333 & -0.1667 \\ 0.5833 & -0.6667 & 0.5833 \end{pmatrix}$$

The minimum 2-norm least-squares solution is

$$A^{\dagger}b = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

10.9 Computing the Singular Value Decomposition

As mentioned before, the ability of the singular values and the singular vectors to compute effectively such values as the numerical rank, the condition number, the orthonormal bases of the range and the null spaces, and the distance of a matrix A from a matrix of lower rank is what makes the singular value decomposition so attractive in many practical applications. Therefore, it is important to know how the SVD can be computed in a numerically effective way. In this section we discuss this most important aspect of the SVD.

10.9.1 Computing the SVD from the Eigenvalue Decomposition of A^TA

We have seen (Theorem 10.3.1) that the singular values of a matrix A are just the nonnegative square roots of the eigenvalues of the symmetric matrix A^TA. It is thus natural to think of computing the singular values by finding the eigenvalues of the symmetric matrix A^TA. However, this is not a numerically effective approach because, as we saw in Chapter 7, some vital information may be lost due to roundoff error in the process of computing A^TA.

The following simple example illustrates the phenomenon:

Example 10.9.1

$$A = \begin{pmatrix} 1.0001 & 1.0000 \\ 1.0000 & 1.0001 \end{pmatrix} \tag{10.9.1}$$

The singular values of A are 2.00010 and 0.0001.

Now

$$A^TA = \begin{pmatrix} 2.0002 & 2.0002 \\ 2.0002 & 2.0002 \end{pmatrix}$$

The eigenvalues of A^TA are 0, 4.0004 (in 4-digit arithmetic).

10.9.2 The Golub–Kahan–Reinsch Algorithm

The following algorithm is, today, a standard algorithm for computing singular values and singular vectors.

The algorithm comes in two phases: Phase I is a finite procedure and Phase II is an iterative one.

Phase I. The $m \times n$ matrix $A(m \geq n)$ is transformed to an upper bidiagonal matrix by orthogonal equivalence:

$$U_0^T A\, V_0 = \begin{pmatrix} B \\ 0 \end{pmatrix} = \tilde{B} \tag{10.9.2}$$

where B is an $n \times n$ bidiagonal matrix given by

$$\begin{pmatrix} b_{11} & b_{12} & 0 & \dots & 0 \\ 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & \ddots & b_{n-1,n} \\ 0 & \dots & \dots & 0 & b_{nn} \end{pmatrix}$$

Phase II. The bidiagonal matrix B is further reduced by orthogonal equivalence to a diagonal matrix Σ using the implicit QR iteration algorithm:

$$U_1^T B V_1 = \Sigma = \operatorname{diag}(\sigma_1, \dots, \sigma_n) \tag{10.9.3}$$

The singular value decomposition of A is then given by

$$U^T A V = \begin{pmatrix} \Sigma \\ 0 \end{pmatrix}$$

where $U = U_0 \operatorname{diag}(U_1, I_{(m-n)})$ and $V = V_0 V_1$.

Remarks: The idea of bidiagonalization of A was first published in a celebrated paper by Golub and Kahan (1965). The idea of using implicit QR on the bidiagonal matrix is due to Golub (Golub 1968). A definitive description of the procedure, along with Algol codes, appeared in Golub and Reinsch (1970). A Fortran program implementing the algorithm appeared in Businger and Golub (1969). A variant of Phase I, which is more efficient in certain cases, was proposed by Chan (1982a,

1982b), and a variant of Phase II has been recently proposed by Demmel and Kahan (1990). Yet another new and faster algorithm has just been proposed by Fernando and Parlett (1994).

A Perturbation Result for Singular Values of a Bidiagonal Matrix

The following result shows that the singular values of a bidiagonal matrix are quite insensitive to small perturbations. For a proof of Theorem 10.9.1, see Demmel and Kahan (1990).

THEOREM 10.9.1 Let $B = (b_{ij})$ be an $n \times n$ bidiagonal matrix. Let $\Delta B = (\delta b_{ij})$ be also bidiagonal. Suppose that $b_{ii} + \delta b_{ii} = \alpha_{2i-1} b_{ii}$ and $b_{i,i+1} + \delta b_{i,i+1} = \alpha_{2i} b_{i,i+1}$, $\alpha_j \neq 0$. Let $\overline{\alpha} = \Pi_{i=1}^{2n-1} \max(|\alpha_i|, |\alpha_i^{-1}|)$. Let $\sigma_1 \geq \cdots \geq \sigma_n$ be the singular values of B and let $\sigma_1' \geq \cdots \geq \sigma_n'$ be the singular values of $B + \Delta B$. Then

$$\frac{\sigma_i}{\overline{\alpha}} \leq \sigma_i' \leq \overline{\alpha}\sigma_i, \qquad i = 1, \ldots, n \tag{10.9.4}$$

Reduction to Bidiagonal Form

The matrices U_0 and V_0 in Phase I are constructed as the product of at most $2n - 2$ Householder matrices as follows:

$$U_0 = U_{01} U_{02} \cdots U_{0n} \tag{10.9.5}$$

and

$$V_0 = V_{01} V_{02} \cdots V_{0,n-2} \tag{10.9.6}$$

Let's illustrate the construction of U_{01}, V_{01} and U_{02}, V_{02}, and their role in the bidiagonalization process with $m = 5$ and $n = 4$.

First, a Householder matrix U_{01} is constructed such that

$$A^{(1)} = U_{01} A = \begin{pmatrix} * & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \end{pmatrix}$$

Next, a Householder matrix V_{01} is constructed such that

$$A^{(2)} = A^{(1)} V_{01} = \begin{pmatrix} * & * & 0 & 0 \\ 0 & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \end{pmatrix} = \left(\begin{array}{c|ccc} * & * & 0 & 0 \\ \hline 0 & & & \\ 0 & & A' & \\ 0 & & & \\ 0 & & & \end{array}\right)$$

The process is now repeated with $A^{(2)}$; that is, Householder matrices U_{02} and V_{02} are constructed so that

$$U_{02} A^{(2)} V_{02} = \begin{pmatrix} * & * & 0 & 0 \\ 0 & * & * & 0 \\ 0 & 0 & * & * \\ 0 & 0 & * & * \\ 0 & 0 & * & * \end{pmatrix}$$

Of course, in this step we will work with the 4×3 matrix A' rather than the matrix $A^{(2)}$. Thus, orthogonal matrices U_{02}' and V_{02}' are first constructed such that

$$U_{02}' A' V_{02}' = \begin{pmatrix} * & * & 0 \\ 0 & * & * \\ 0 & * & * \\ 0 & * & * \end{pmatrix}$$

Then U_{02} and V_{02} are constructed from U_{02}' and V_{02}' in the usual way—that is, by embedding them in identity matrices of appropriate orders. The process is continued until the bidiagonal matrix B is obtained.

MATCOM Note: The preceding bidiagonalization procedure of Golub and Kahan (1965) has been implemented in MATCOM program BIDIAG.

Example 10.9.2

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{pmatrix}$$

Step 1:

$$U_{01} = \begin{pmatrix} -0.1474 & -0.4423 & -0.8847 \\ -0.4423 & 0.8295 & -0.3410 \\ -0.8847 & -0.3410 & 0.3180 \end{pmatrix}$$

$$A^{(1)} = U_{01} A = \begin{pmatrix} -6.7823 & -8.2567 & -9.7312 \\ 0 & 0.0461 & 0.0923 \\ 0 & -0.9077 & -1.8154 \end{pmatrix}$$

Step 2:

$$V_{01} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -0.6470 & -0.7625 \\ 0 & -0.7625 & 0.6470 \end{pmatrix}$$

$$A^{(2)} = A^{(1)} V_{01} = \begin{pmatrix} -6.7823 & 12.7620 & 0 \\ 0 & -0.1002 & 0.0245 \\ 0 & 1.9716 & -0.4824 \end{pmatrix}$$

Step 3:

$$U_{02} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -0.0508 & 0.9987 \\ 0 & 0.9987 & 0.0508 \end{pmatrix}$$

$$B = U_{02} A^{(2)} = U_{021} A^{(1)} V_{01} = U_{02} U_{01} A V_{01}$$

$$= \begin{pmatrix} -6.7823 & 12.7620 & 0 \\ 0 & 1.9741 & -0.4830 \\ 0 & 0 & 0 \end{pmatrix}$$

Note that from the preceding expression of B, it immediately follows that zero is a singular value of A. Also, only three Householder transformations were needed.

Finding the SVD of the Bidiagonal Matrix

The process is a variant of the QR iteration. Starting from the $n \times n$ bidiagonal matrix B obtained in Phase I, it successively constructs a sequence of bidiagonal matrices $\{B_i\}$ such that each B_i has possibly smaller off-diagonal entries than the previous one. The ith iteration is equivalent to applying the implicit symmetric QR, described in Chapter 8, with a Wilkinson shift to the symmetric tridiagonal matrix $B_i^T B_i$ but without, of course, forming the product $B_i^T B_i$ explicitly. The effective tridiagonal matrices are assumed to be unreduced (note that the implicit symmetric QR algorithm works with unreduced matrices); otherwise we would work with decoupled SVD problems. For example, if $b_{k,k+1} = 0$, then B can be written as the direct sum of two bidiagonal matrices B_1 and B_2, and $\sigma(B) = \sigma(B_1) \cup \sigma(B_2)$.

We assume, furthermore, that none of the diagonal entries of B is zero. If $b_{kk} = 0$ for some k, then it is possible to zero the corresponding superdiagonal entry $b_{k,k+1}$ by premultiplication with a sequence of suitable Givens rotations; thus, the problem can be decoupled again (Exercise 25). Therefore, a zero entry in diagonal or superdiagonal is a blessing; it results in a decoupling of the problem.

Just one iteration step of the method is described here. For a detailed description of the algorithm, see Golub and Reinsch (1970). To simplify the notation, let

$$B = \begin{pmatrix} \alpha_1 & \beta_2 & & 0 \\ & \ddots & \ddots & \\ & & \ddots & \beta_n \\ & 0 & & \alpha_n \end{pmatrix} \tag{10.9.7}$$

be an $n \times n$ bidiagonal matrix. Then the Wilkinson shift σ for the symmetric matrix $B^T B$ is the eigenvalue λ of 2×2 right-hand corner submatrix of $B^T B$:

$$\begin{pmatrix} \alpha_{n-1}^2 + \beta_{n-1}^2 & \alpha_{n-1}\beta_n \\ \beta_n \alpha_{n-1} & \alpha_n^2 + \beta_n^2 \end{pmatrix} \tag{10.9.8}$$

that is closer to $\alpha_n^2 + \beta_n^2$ (see Section 8.11.2).

1. Compute σ as shown in Section 8.11.2 and then compute a Givens rotation J_1' such that

$$J_1' \begin{pmatrix} \alpha_1^2 - \sigma \\ \alpha_1 \beta_2 \end{pmatrix} = \begin{pmatrix} * \\ 0 \end{pmatrix}$$

Form

$$J_1 = \begin{pmatrix} J_1' & 0 \\ 0 & I_{n-2} \end{pmatrix}$$

2. Apply J_1 to the right of B; that is, form

$$B \equiv BJ_1$$

This will fill in the (2, 1) position of B. That is, we will have

$$B \equiv BJ_1 = \begin{pmatrix} * & * & & & \\ + & * & * & & \\ & & \ddots & \ddots & \\ & & & \ddots & * \end{pmatrix} \tag{10.9.9}$$

where + indicates a filled-in value. (The filled-in value is at the (2, 1) position.)

The idea is now to chase the nonzero entry + down the subdiagonal to the end of the matrix by applying the Givens rotations in an appropriate order, as indicated by the following:

3. $B \equiv J_2 B = \begin{pmatrix} * & * & + & & \\ & * & \ddots & & \\ & & \ddots & \ddots & \\ & & & \ddots & * \\ & & & & * \end{pmatrix}$ (Fill-in at the (1, 3) position.)

$B \equiv BJ_3 = \begin{pmatrix} * & * & & & \\ & * & \ddots & & \\ & + & \ddots & \ddots & \\ & & & \ddots & * \\ & & & & * \end{pmatrix}$ (Fill-in at the (3, 2) position.)

4. $B \equiv J_4 B = \begin{pmatrix} * & * & & & \\ & * & * & + & \\ & & \ddots & \ddots & \\ & & & \ddots & * \\ & & & & * \end{pmatrix}$ (Fill-in at the (2, 4) position.)

$B \equiv BJ_5 = \begin{pmatrix} * & * & & & \\ & * & * & & \\ & & * & \ddots & \\ & & + & \ddots & * \\ & & & & * \end{pmatrix}$ (Fill-in at the (4, 3) position, and so on).

Note that the entries (2, 1), (3, 2), and so on are annihilated by a premultiplication, whereas the entries (1, 3), (2, 4), and so on are annihilated by a postmultiplication.

At the end of one iteration we will have a new bidiagonal matrix $\overline{B}$ orthogonally equivalent to the original bidiagonal matrix B:

$$\overline{B} = (J_{2n-2} \cdots J_4 J_2) B (J_1 J_3 \cdots J_{2n-3})$$

Example 10.9.3

$$B = \begin{pmatrix} 1 & 2 & 0 \\ 0 & 2 & 3 \\ 0 & 0 & 1 \end{pmatrix}$$

1. The Wilkinson shift: $\sigma = 15.0828$

$$J_1 = \begin{pmatrix} -0.9901 & 0.1406 & 0 \\ -0.1406 & -0.9901 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

2. $B \equiv BJ_1 = \begin{pmatrix} -1.2713 & -1.8395 & 0 \\ -0.2812 & -1.9801 & 3 \\ 0 & 0 & 1 \end{pmatrix}$

(Fill-in at the (2, 1) position.)

3. $J_2 = \begin{pmatrix} -0.9764 & -0.2160 & 0 \\ 0.2160 & -0.9764 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

$$B \equiv J_2 B = \begin{pmatrix} 1.3020 & 2.2238 & -0.6480 \\ 0 & 1.5361 & -2.9292 \\ 0 & 0 & 1 \end{pmatrix}$$

(Fill-in at the (1, 3) position.)

$$J_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0.9601 & 0.2797 \\ 0 & -0.2797 & 0.9601 \end{pmatrix}$$

$$B \equiv BJ_3 = \begin{pmatrix} 1.3020 & 2.3163 & 0 \\ 0 & 2.2942 & -2.3825 \\ 0 & -0.2797 & 0.9601 \end{pmatrix}$$

(Fill-in at the (3, 2) position.)

4. $J_4 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0.9926 & -0.1210 \\ 0 & 0.1210 & 0.9926 \end{pmatrix}$

$$B \equiv J_4 B = \begin{pmatrix} 1.3020 & 2.3163 & 0 \\ 0 & 2.3112 & -2.4812 \\ 0 & 0 & 0.6646 \end{pmatrix}$$

Deflation and Convergence

We just described one **iteration step** of the bidiagonal SVD algorithm. As mentioned before, by repeated applications of the algorithm, one constructs a sequence of upper bidiagonal matrices $\{B_k\}$ with diagonal entries $\alpha_1^{(k)}, \ldots, \alpha_n^{(k)}$ and the superdiagonal entries $\beta_2^{(k)}, \ldots, \beta_n^{(k)}$.

From the quadratic convergence of the implicit QR iteration algorithm with Wilkinson shift described in Algorithm 8.11.2, it follows that $\beta_n^{(k)}$ converges to zero quadratically (the convergence is more like cubic in practice).

Having set $\beta_n^{(k)}$ to zero (according to the criterion that follows), the matrix is deflated and the algorithm is repeated on one or more matrices of order $(n-1)$ or lower. The process is continued until B becomes diagonal.

Golub and Reinsch (1970) have remarked that "the numerical experiments indicate that the average number of complete QR iteration on the bidiagonal matrix is usually less than two per singular value."

Criteria for Neglecting Off-Diagonal and Diagonal Entries

1. Accept an off-diagonal β_i to be zero if

$$|\beta_i| \le \epsilon(|\alpha_i| + |\alpha_{i-1}|)$$

where ϵ is a small multiple of the machine precision μ.

2. Accept a diagonal entry α_i to be zero if

$$|\alpha_i| \le \epsilon(|\beta_i| + |\beta_{i+1}|)$$

where ϵ is a small multiple of the machine precision μ.

Flop count. The cost of the method is dominated by the cost of Phase I. The **estimated** total flop count is $2m^2n + 4mn^2 + \frac{9}{2}n^3 (m \ge n)$. This count includes the cost of U, Σ, and V. There are applications (for example, least squares) where all three matrices are not explicitly required. A nice table of different flop counts of the Golub–Kahan–Reinsch SVD algorithm and the Chan SVD (described in the next section) for different requirements of U, Σ, and V appears in Golub and Van Loan (1989, p.239). If only the singular values are needed, then the estimated flop count is

$$2mn^2 - 2\frac{n^3}{3}$$

Roundoff property. It can be shown (Bjorck, 1994b, Golub and Van Loan, 1989) that the computed singular values, produced by the Golub–Kahan–Reinsch algorithm, are the exact singular values of $A + E$, where

$$\|E\|_2 \le c(m, n)\mu\|A\|_2$$

Here $c(m, n)$ is a constant depending on m and n. The algorithm is thus backward stable.

Accuracy of the Computed Singular Values

Furthermore, let $\overline{\sigma}_i$ be a computed singular value; then

$$|\sigma_i - \overline{\sigma}_i| \le c(m, n)\|A\|_2 = c(m, n)\mu\sigma_{\max}$$

The result says that the computed singular values cannot differ from the true singular values by an amount larger than $\delta = c(m, n)\mu\sigma_{\max}$.

10.9.3 The Chan SVD

T. Chan (1982a) observed that the Golub–Kahan–Reinsch algorithm for computing the SVD described in the last section can be improved in the case $m \gg n$ if the matrix A is first factored into QR and then the bidiagonalization is performed on the smaller upper triangular matrix resulting from the decomposition. The improvement naturally comes from the fact that the work required to bidiagonalize the triangular matrix R is much less than that required originally to bidiagonalize the matrix A. Of course,

once the SVD of R is obtained, one can easily retrieve the SVD of A. Thus the **Chan SVD** can be described as follows:

1. Find the QR factorization of A:

$$Q^T A = \begin{pmatrix} R_1 \\ 0 \end{pmatrix} \tag{10.9.10}$$

2. Find the SVD of R using the Golub–Kahan–Reinsch algorithm:

$$R_1 = X\Sigma Y^T \tag{10.9.11}$$

Then the singular values of A are just the singular values of R_1. The singular vector matrices U and V are given by

$$U = Q\text{diag}(X, I_{m-n}), \qquad V = Y \tag{10.9.12}$$

Flop count. The Chan SVD requires about $2m^2n + 11n^3$ flops to compute Σ, U, and V, compared with $2m^2n + 4mn^2 + \frac{9}{2}n^3$ flops required by the Golub–Kahan–Reinsch SVD algorithm. Clearly, there will be savings with the Chan SVD when $m \gg n$.

Remarks: The Chan SVD plays an important role in several applications, such as in control theory, where typically $m \gg n$. The idea of triangularizing A before performing bidiagonalization for the SVD of A was mentioned earlier in Lawson and Hanson (1974, 119) in the context of least-squares problems.

Example 10.9.4

$$A = \begin{pmatrix} 1 & 2 \\ 2 & 3 \\ 4 & 5 \end{pmatrix}$$

1. The QR factorization of A is

$$Q = \begin{pmatrix} -0.2182 & -0.8165 & -0.5345 \\ -0.4364 & -0.4082 & 0.8018 \\ -0.8729 & 0.4082 & -0.2673 \end{pmatrix}$$

$$R_1 = \begin{pmatrix} -4.5826 & -6.1101 \\ 0 & -0.8165 \end{pmatrix}$$

2. The SVD of R_1 is

$$R_1 = X\Sigma Y^T$$

$$X = \begin{pmatrix} -0.9963 & 0.0856 \\ -0.0856 & -0.9963 \end{pmatrix}$$

$$Y = \begin{pmatrix} 0.5956 & -0.8033 \\ 0.8033 & 0.5956 \end{pmatrix}$$

$$\Sigma = \begin{pmatrix} 7.6656 & 0 \\ 0 & 0.4881 \end{pmatrix}$$

3. The singular value decomposition of $A = U\Sigma V^T$ is as follows. The singular values of A are 7.6656 and 0.4881.

$$U = \begin{pmatrix} 0.2873 & 0.7948 & -0.5345 \\ 0.4698 & 0.3694 & 0.8018 \\ 0.8347 & -0.4814 & -0.2673 \end{pmatrix}$$

$$V = Y$$

Flop Count for the Least-Squares Problem Using the SVD

The number of flops required to solve the linear least-squares problem ($m \geq n$) using the SVD computed by the Golub–Kahan–Reinsch algorithm is about $2mn^2 + 4n^3$. If the Chan SVD is used, the flop count is about $mn^2 + \frac{11}{2}n^3$.

Recall that the flop count for the normal equations approach for the least-squares problem is $mn^2/2 + n^3/6$ and that using QR factorization (with Householder matrices) is $mn^2 - n^3/3$. Again a nice table of different least-squares methods in regards to their efficiency appears in Golub and Van Loan (1989, p.248). See also Chan (1982a). In the above count, it is assumed that an average 2 QR iterations per singular value are needed.

10.9.4 The Demmel–Kahan Zero-Shift QR Algorithm

In a recent paper, Demmel and Kahan (1990) presented a new algorithm that computes all the singular values of a bidiagonal matrix with as much accuracy as permitted by data. They call this algorithm *QR iteration with a zero shift*. The reason for this name is that the new algorithm corresponds to the Golub–Kahan–Reinsch algorithm when the shift is zero. The new algorithm is based on an observation that when the shift is zero, no cancellation due to subtraction occurs; therefore, the very small singular values may be found (almost) as accurately as the data permits.

Indeed, the effect of the zero shift is remarkable. This is demonstrated in the following. Let B be the same as in (10.9.7). Let J_1' be the Givens rotation such that

$$(J_1')\begin{pmatrix} \alpha_1^2 \\ \alpha_1\beta_2 \end{pmatrix} = \begin{pmatrix} * \\ 0 \end{pmatrix}$$

Compute

$$J_1 = \begin{pmatrix} J_1' & 0 \\ 0 & I_{n-2} \end{pmatrix}$$

Then

$$BJ_1 = \begin{pmatrix} * & 0 & & & \\ + & * & * & & \\ & & \ddots & \ddots & \\ & & & \ddots & * \\ & & & & * \end{pmatrix}$$

where a fill-in at the (2, 1) position occurs as before, but the (1, 2) entry now is zero instead of being nonzero. This zero will propagate through the rest of the algorithm and is the key to the effectiveness of this algorithm.

Let us now apply J_2 to the left of BJ_1 to zero the nonzero entry at the (2, 1) position. Then

$$B \equiv J_2BJ_1 = \begin{pmatrix} * & * & + & \\ & * & \ddots & \\ 0 & & \ddots & * \\ & & & * \end{pmatrix}$$

There is now a fill-in at the (1, 3) position, as in the Golub–Kahan–Reinsch algorithm. Now it is easy to verify that the rank of the 2×2 submatrix of the matrix J_2BJ_1:

$$\begin{pmatrix} b_{12} & b_{13} \\ b_{22} & b_{23} \end{pmatrix}$$

is 1, it then follows that when J_3 is applied to J_2BJ_1 to the right to zero-out the (1, 3) entry, it will zero out the (2, 3) entry as well; that is, we will have

$$B \equiv J_2BJ_1J_3 = \begin{pmatrix} * & * & 0 & & & \\ & * & 0 & & & \\ & + & \ddots & * & & \\ & & & \ddots & \ddots & \\ & & & & \ddots & * \\ 0 & & & & & * \end{pmatrix}$$

Thus, we now have one extra zero on the superdiagonal, compared to the same stage of the Golub–Kahan–Reinsch algorithm. This phenomenon continues. Indeed, the rotation J_5 zeros-out the (3, 4) entry as well as the (2, 4) entry, and so on.

Remark: Here we have merely demonstrated the effect of a zero shift on the Golub–Kahan–Reinsch algorithm. For details of the algorithm we refer to the paper by Demmel and Kahan (1990).

Example 10.9.5

1.

$$B = \begin{pmatrix} 1 & 2 & 0 \\ 0 & 2 & 3 \\ 0 & 0 & 1 \end{pmatrix}$$

$$J_1 = \begin{pmatrix} 0.4472 & -0.8944 & 0 \\ 0.8944 & 0.4472 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

2. $$B \equiv BJ_1 = \begin{pmatrix} 2.2361 & 0 & 0 \\ 1.7889 & 0.8944 & 3 \\ 0 & 0 & 1 \end{pmatrix}$$

(Note that there is a fill-in at the (2, 1) position, but the (1, 2) entry is zero.)

3.

$$J_2 = \begin{pmatrix} 0.7809 & 0.6244 & 0 \\ -0.6247 & 0.7809 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$B \equiv J_2BJ_1 = \begin{pmatrix} 2.8636 & 0.5588 & 1.8741 \\ 0 & 0.6984 & 2.3426 \\ 0 & 0 & 1 \end{pmatrix}$$

Note that the rank of the 2×2 matrix is 1.

$$\begin{pmatrix} 0.5588 & 1.8741 \\ 0.6984 & 2.3476 \end{pmatrix}$$

$$J_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0.2857 & -0.9583 \\ 0 & 0.9583 & 0.2857 \end{pmatrix}$$

$$B \equiv J_2BJ_1J_3 = \begin{pmatrix} 2.8636 & 1.9557 & 0 \\ 0 & 2.4445 & 0 \\ 0 & 0.9583 & 0.2857 \end{pmatrix}$$

(Note that both the (1, 3) and (2, 3) entries are zero.)

4.

$$J_4 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0.9310 & 0.3650 \\ 0 & -0.3650 & 0.9310 \end{pmatrix}$$

$$B \equiv J_4J_2BJ_1J_3 = \begin{pmatrix} 2.8636 & 1.9557 & 0 \\ 0 & 2.6256 & 0.1043 \\ 0 & 0 & 0.2660 \end{pmatrix}$$

Note that the singular values of the last matrix B are 3.8990, 1.9306, and 0.2657, same as those of the original B.

Rate of convergence. The convergence of the last off-diagonal entry to zero is linear with constant factor $\sigma_{n-1}^2/\sigma_n^2$. If there is a cluster of m small singular values isolated from the remaining ones, then the convergence will be linear with constant factor $\sigma_{n-m+1}^2/\sigma_{n-m}^2$.

Accuracy of the computed eigenvalues. Let $\overline{\sigma}_i$, $i = 1, \ldots, n$ be the singular values of the computed bidiagonal matrix $\overline{B}$ obtained by applying one implicit zero shift QR iteration to B. Let $\overline{\sigma}_1 \geq \overline{\sigma}_2 \geq \cdots \geq \overline{\sigma}_n$. It has been shown by Demmel and Kahan (1990) that, if

$$w \approx 69n^2\mu < 1$$

then

$$|\sigma_i - \overline{\sigma}_i| \leq \frac{w}{1-w}\sigma_i, \qquad i = 1, \ldots, n$$

Moreover, if after k iterations, the singular values of the then bidiagonal matrix B_k are $\sigma_{k1} \geq \cdots \geq \sigma_{kn}$, then when $w \approx 69n^2\mu < 1$, we have

$$|\sigma_i - \sigma_{ki}| \leq \left(\frac{1}{(1-w)^k} - 1\right) \cdot \sigma_i \approx 69kn^2\mu \cdot \sigma_i$$

The preceding result basically states that "the relative error in the computed singular values can grow only with the squares of the dimension." This is a rather pessimistic result. The authors have given another roundoff result that states that with the approach of the convergences of the algorithm, "errors do not accumulate at all and the error in the computed α_i and β_i is bounded by $c'\mu$, c' another modest constant."

An error bound on the computed singular values is of the form

$$\frac{c'n\mu}{(1 - c'n\mu)}$$

where c' is another modest constant. For details and the proofs of these results, the reader is referred to the paper by Demmel and Kahan (1990).

10.9.5 The Fernando-Parlett dqds Algorithm

Recently, Fernando and Parlett (1994) have proposed an algorithm called dqds (differential quotient difference with shift) that computes all the singular values with maximal relative accuracy. Unfortunately, we cannot describe this important algorithm here, as the machinery needed to explain the algorithm has not been developed in this book. For description of the algorithm, a comparison of the algorithm over the other SVD algorithms, and references of the original quotient difference algorithm of H. Rutishauser, refer to Fernando and Parlett (1994).

10.9.6 The Bisection Method

To conclude the section, we note that the bisection method for the symmetric eigenvalue problem discussed in Chapter 8 can be adapted to compute the singular values of a matrix. The idea was first stated in Golub and Kahan (1965).

Note that the singular values of

$$B = \begin{pmatrix} b_{11} & b_{12} & & 0 \\ & \ddots & \ddots & \\ & & \ddots & b_{n-1,n} \\ 0 & & & b_{nn} \end{pmatrix}$$

are positive eigenvalues of

$$\tilde{T} = \begin{pmatrix} 0 & B \\ B^T & 0 \end{pmatrix}$$

It is easy to see that there exists a permutation matrix P such that

$$T = P^T\tilde{T}P = \begin{pmatrix} 0 & b_{11} & & & & & \\ b_{11} & \ddots & b_{12} & & & & \\ & b_{12} & \ddots & b_{22} & & & \\ & & b_{22} & \ddots & b_{23} & & \\ & & & b_{23} & \ddots & \ddots & \\ & & & & \ddots & \ddots & b_{n-1,n} \\ & & & & & b_{n-1,n} & \ddots & b_{nn} \\ & & & & & & b_{nn} & 0 \end{pmatrix}$$

which is a symmetric tridiagonal matrix with 0 diagonal entries. Thus the bisection method described earlier in Chapter 8 can now be used to find the positive eigenvalues of T—that is, the singular values of B. The advantage of the bisection method is its simplicity, accuracy, and inherent parallelism (see a forthcoming paper by Li, Rhee, and Zeng). However, it is inefficient in sequential computation compared to QR-based algorithms.

10.10 Generalized SVD

The SVD Theorem (Theorem 10.2.1) can be generalized for a pair of matrices A and B, and this generalized SVD is useful in certain applications such as constrained least-squares problems.

The generalization was first performed by Van Loan (1976). We merely state the theorem here without proof. For a proof, see Van Loan (1976), and Paige and Saunders (1981). For computational algorithms see Stewart (1983), Paige (1986a), and Bai and Demmel (1993).

THEOREM 10.10.1 Generalized Singular Value Decomposition Theorem Let A and B be, respectively, matrices of order $m \times n$ and $p \times n$ ($m \geq n$). Then there exist orthogonal matrices U and V, a nonsingular matrix W, and diagonal matrices C and D of order $m \times n$ and $p \times n$, respectively, such that

$$U^TAW = C = \text{diag}(c_1, \ldots, c_n), \qquad c_i \geq 0$$
$$V^TBW = D = \text{diag}(d_1, \ldots, d_q), \qquad d_i \geq 0$$

where $q = \min(p, n)$; $d_1 \geq \cdots \geq d_r > d_{r+1} = \cdots = d_q = 0$, $r = \text{rank}(B)$. The elements

$$\left\{\frac{c_1}{d_1}, \frac{c_2}{d_2}, \ldots, \frac{c_r}{d_r}\right\}$$

are called the generalized singular values of A and B. The generalized singular values corresponding to zero d_is are infinite.

10.11 Review and Summary

The following are the most important results discussed in this chapter.

1. *Existence and uniqueness of the SVD:* The singular value decomposition (SVD) of a matrix A always exists (Theorem 10.2.1).

$$A = U\Sigma V^T$$

The singular values (the diagonal entries of Σ) are unique, but U and V are not unique.

2. *Relationship of the singular values and singular vectors to the eigenvalues:* The singular values of A are the nonnegative square roots of the eigenvalues of A^TA (or of AA^T). (Theorem 10.3.1; see also Theorem 10.3.2.)
3. *Sensitivity of the singular values:* The singular values are insensitive to small perturbations (Theorem 10.5.1).
4. *Applications of the SVD:* The singular values and the singular vectors of a matrix A are useful and are the most reliable tools for determining the (numerical) rank and the rank-deficiency of A; finding the orthonormal bases for range and the null space of A; finding the distance of A from another matrix of lower rank (in particular the nearness to singularity of a nonsingular matrix); solving both full-rank and the rank-deficient least-squares problems; and finding the pseudoinverse of A (Sections 10.6–10.8).

   These remarkable abilities and the fact that the singular values are insensitive to small perturbations have made the SVD an indispensable tool in a wide variety of application areas, such as control and systems theory, signal processing, statistical applications, biomedical engineering, and image processing.
5. *Computing the SVD:* The most widely used approach for computing the SVD of A is the Golub–Kahan–Reinsch algorithm. This algorithm comes into two phases. In Phase I, the matrix A is reduced to a bidiagonal matrix by orthogonal equivalence, and in Phase II, the bidiagonal matrix is further reduced to a diagonal matrix by orthogonal similarity using implicit QR iteration with the Wilkinson shift. A modification of this method, known as the zero-shift QR iteration, or the QR iteration with a zero shift, was proposed by Demmel and Kahan in 1990. The Demmel–Kahan method computes all the singular values of a bidiagonal matrix to high relative accuracy. Yet another method has just been proposed by Fernando and Parlett (Section 10.9.5).

EXERCISES*

Sections 10.2 and 10.3

1. (a) Let A be $n \times n$ and let U and V be orthogonal. Then from the definition of singular values, prove that the singular values of A and U^TAV are the same.
   (b) How are the singular vectors of A related to those of U^TAV?

*Use MATLAB wherever needed and appropriate.

   (c) If A is symmetric positive definite, prove that the eigenvalues of A are the same as its singular values.
2. (*An Economic Version of the SVD*) Let A be $m \times n$ ($m \geq n$). Let $\text{rank}(A) = r \leq n$. Then prove that there exist matrices $\hat{U}_{m\times r}$ and $\hat{V}_{n\times r}$ with orthonormal columns and a nonsingular diagonal matrix $S_{r\times r}$ such that

$$A = \hat{U}S(\hat{V})^T$$

Why is this version called the economic version?

3. Let σ be a singular value of A with multiplicity ℓ; that is, $\sigma_i = \sigma_{i+1} = \cdots = \sigma_{i+\ell-1}$. Let $U\Sigma V^T$ be a singular value decomposition of A. Then show how to construct $\tilde{U}$ and $\tilde{V}$ from U and V such that $\tilde{U}\Sigma(\tilde{V})^T$ is also an *SVD* of A.
4. (a) Derive Theorem 10.3.1 from Theorem 10.3.2.
   (b) Given

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$$

Find the singular values σ_1 and σ_2 of A by computing the eigenvalues of A^TA. Then find the orthogonal matrix P such that

$$P^TSP = \text{diag}(\sigma_1, \sigma_2, -\sigma_1, -\sigma_2, 0)$$

where $S = \begin{pmatrix} 0_{3\times 3} & A \\ A^T & 0_{2\times 2} \end{pmatrix}$.

5. Using the constructive proof of Theorem 10.2.1, find the SVD of the following matrices:

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}, \qquad A = (1 \quad 2 \quad 3)$$

$$A = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \qquad A = \text{diag}(1, 0, 2, 0, -5)$$

$$A = \begin{pmatrix} 1 & 1 \\ \epsilon & 0 \\ 0 & \epsilon \end{pmatrix}, \qquad \text{where } \epsilon \text{ is small}$$

Sections 10.5–10.8

6. (a) Find the rank, $\|\ \|_2$, $\|\ \|_F$, $\text{Cond}(A)$, and the orthonormal bases for the null space and the range of each of the matrices in Exercise 5. Find the orthogonal projections onto the range and the null space of A and of their orthogonal complements.
   (b) Using the *SVD* of A, prove that for an $m \times n$ matrix A,
      (i) $\text{rank}(A^TA) = \text{rank}(AA^T) = \text{rank}(A) = \text{rank}(A^T)$.
      (ii) A^TA and AA^T have the same nonzero eigenvalues.
      (iii) If the eigenvectors u_1 and u_2 of A^TA are orthogonal, then Au_1 and Au_2 are orthogonal.

7. (a) Let A be an $m \times n$ matrix with singular values $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n > 0$. Let $v_1, \ldots, v_n$ be the right singular vectors of A. Then prove that $\|Ax\|_2/\|x\|_2$ is maximized if $x = v_1$ and minimized if $x = v_n$.
   (b) Let U be an orthogonal matrix. Then prove that
$$\|AU\|_2 = \|A\|_2$$
and
$$\|AU\|_F = \|A\|_F$$
   (c) Let A be an $m \times n$ matrix. Using the SVD of A, prove that
   (i) $\text{Cond}_2(A^TA) = (\text{Cond}_2(A))^2$
   (ii) $\|A^TA\|_2 = \|A\|_2^2$
   (iii) $\text{Cond}_2(A) = \text{Cond}_2(U^TAV)$
   where U and V are orthogonal.
   (d) Let $\text{rank}(A_{m\times n}) = r$, and let $B_{m\times r}$ be a matrix obtained by deleting $n - r$ columns from A. Then prove that $\text{Cond}_2(B) \leq \text{Cond}_2(A)$.
8. Prove that if A is an $m \times n$ matrix with rank r and if B is another $m \times n$ matrix satisfying $\|A - B\|_2 < \sigma_r$, then B has at least rank r.
9. (a) Give an alternate proof of Theorem 10.6.3.
   (b) Given
$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$$
   find a rank-one matrix B closest to A in 2-norm. What is $\|B - A\|_2$?
10. Let A and B be $n \times n$ real matrices. Let Q be an orthogonal matrix such that $\|A - BQ\|_F \leq \|A - BX\|_F$ for any orthogonal matrix X. Then prove that $Q = UV^T$, where $B^TA = U\Sigma V^T$ (Green, 1952).
11. Given
$$A = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 5 & 6 & 7 \end{pmatrix}$$
and using the result of Exercise 10, show that out of all the orthogonal matrices X, the matrix
$$Q = \begin{pmatrix} -0.2310 & -0.3905 & 0.8912 \\ -0.4824 & 0.8414 & 0.2436 \\ 0.8449 & 0.3736 & 0.3827 \end{pmatrix}$$
is such that $\|A - Q\|_F \leq \|A - X\|_F$. Find $\|A - Q\|_F$ by computing the singular values of $A - Q$.
12. (a) Let
$$A = \begin{pmatrix} 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{pmatrix}$$
   Find the outer product expansion of A using the SVD of A.
   (b) Compute $(A^TA)^{-1}$ using the SVD of A.

Section 10.8

13. Let
$$D = \begin{pmatrix} \sigma_1 & & & & & 0 \\ & \ddots & & & & \\ & & \sigma_r & & & \\ & & & 0 & & \\ & & & & \ddots & \\ 0 & & & & & 0 \end{pmatrix}, \qquad \sigma_i > 0,\ i = 1, \ldots, r$$
Then show that
$$D^\dagger = \begin{pmatrix} \frac{1}{\sigma_1} & & & & & 0 \\ & \ddots & & & & \\ & & \frac{1}{\sigma_r} & & & \\ & & & 0 & & \\ & & & & \ddots & \\ 0 & & & & & 0 \end{pmatrix}$$
14. Verify that the matrix $A^\dagger = V\Sigma^\dagger U^T$, where $\Sigma^\dagger$ is given by (10.8.2), is the pseudoinverse of A. (Check all four conditions for the definition of the pseudoinverse.)
15. For any nonzero matrix A, show that
   (a) $AA^\dagger v = v$ for any vector v in $R(A)$.
   (b) $A^\dagger x = 0$ for any x in $N(A^T)$.
   (c) $(A^T)^\dagger = (A^\dagger)^T$
   (d) $(A^\dagger)^\dagger = A$
16. Let A be an $m \times n$ matrix. Show that
   (a) If A has full column rank, then
$$A^\dagger = (A^TA)^{-1}A^T$$
   (b) If A has full row rank, then
$$A^\dagger = A^T(AA^T)^{-1}$$
17. From the SVD of A, compute the singular value decompositions of the projection matrices: $P_1 = A^\dagger A$, $P_2 = I - A^\dagger A$, $P_3 = AA^\dagger$, and $P_4 = I - AA^\dagger$. Also verify that each of these is a projection matrix.
18. Let
$$A = \begin{pmatrix} 1 & 2 \\ 2 & 4 \\ 3 & 6 \end{pmatrix}, \qquad b = \begin{pmatrix} 3 \\ 6 \\ 9 \end{pmatrix}$$
Find the minimum 2-norm solution x to the least-squares problem
$$\min_x \|Ax - b\|_2$$
What is $\|x\|_2$?

19. Given

$$u = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}, \qquad v = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

find $A^\dagger$ where

$$A = uv^T$$

20. Let

$$A = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 2 & 3 & 4 & 5 \end{pmatrix}, \qquad b = \begin{pmatrix} 3 \\ 3 \\ 14 \end{pmatrix}$$

(a) Find a least-squares solution x: $\min_x \|Ax - b\|_2$. What is $\|x\|_2$?
(b) Find the minimum 2-norm solution x and $\|x\|_2$.

Section 10.9

21. Let B be an upper bidiagonal matrix having a multiple singular value. Then prove that B must have a zero either on its diagonal or superdiagonal.
22. Consider the family of bidiagonal matrices

$$B(\eta) = \begin{pmatrix} 1-\eta & \beta(1+\eta) & & \\ & \ddots & \ddots & \\ & & \ddots & \ddots \\ & & & \beta(1+\eta) \\ & & & 1-\eta \end{pmatrix}$$

$\beta \gg 1$. It can be shown (Demmel and Kahan 1990) that the smallest singular value of $B(\eta)$ is approximately $\beta^{1-n}(1-(2n-1)\eta)$. Taking $\beta = 10^6$, and using $n = 20$ and $\eta = 0$, verify this result.
23. Let

$$A = \begin{pmatrix} 1 & 0.9999 \\ 2 & 1.9999 \\ 3 & 2.9999 \end{pmatrix}$$

(a) Find the SVD of A using the Golub–Kahan–Reinsch algorithm.
(b) Find the SVD of A using the Demmel–Kahan algorithm.
(c) Compare the results of (a) and (b).
24. Show that an approximate flop count for the Golub–Kahan–Reinsch and the Chan SVD are, respectively, $2m^2n + 4mn^2 + \frac{9}{2}n^3$ and $2m^2n + 11n^3$ for an $m \times n$ matrix A. Also compute the flop count for the Demmel–Kahan algorithm.
25. Let B be an $n \times n$ upper bidiagonal matrix with $\alpha_1, \ldots, \alpha_n$ on the diagonal and $\beta_2, \ldots, \beta_n$ on the superdiagonal. Prove that if $\alpha_k = 0$, with $\alpha_j \neq 0$ and $\beta_j \neq 0$, $j = k+1, \ldots, n$, then premultiplying B by $(n-k)$ Givens rotations will decouple B.

MATLAB PROGRAMS AND PROBLEMS

Note: You will need **housqr**, **parpiv**, and **varcovar** from MATCOM.

1. (The purpose of this exercise is to study the sensitivities (insensitivities) of the singular values of a matrix). Using MATLAB commands **svd** and **norm**, verify the inequalities in Theorems 10.5.1 and 10.5.2, respectively.

Test Data

1. $$A = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 0.99 & 1 & 1 \\ 0 & 0 & 0.99 & 1 \\ 0 & 0 & 0 & 0.99 \end{pmatrix}$$

2. $A =$ the Wilkinson bidiagonal matrix of order 20.

In each case, construct a suitable E so that $(A + E)$ differs from A in the $(n, 1)$st element only by an $\epsilon = 10^{-5}$. (Note that the eigenvalues of both matrices are ill conditioned.)

2. Generate randomly a 15×5 matrix using the MATLAB command $A =$ **rand**$(15, 5)$. Find $s =$ **svd**(A). Compute $\|A\|_2$ and $\|A\|_F$, the condition number of A with respect to 2-norm using the entries of s, and then compare your results with those obtained by MATLAB commands **norm(a)**, **norm(A,'fro')**, and **cond(A)**, respectively.
3. (a) Compute the orthonormal bases for the range and the null space of a matrix A as follows:
   (i) Use $[U, S, V] =$ **svd** (A) from MATLAB (SVD Approach: Section 10.6.2).
   (ii) Use **housqr** or **housqrp** from Chapter 5, as appropriate (QR Approach: Section 5.6).
   (iii) Use **orth**(A) and **null**(A) from MATLAB.
   (b) Compare the results of (i), (ii), and (iii) and the flop count for each algorithm.

Test Data

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

2. $A =$ a randomly generated matrix of order 10.

4. Compute the rank of each of the following matrices using (i) MATLAB command **rank** (that uses the singular values of A) and (ii) the programs **housqrp** (House-

holder QR factorization with pivoting) and **parpiv** from Chapter 5; compare the results.

Test Data and the Experiment

1. (Kahan matrix)

$$A = \text{diag}(1, s, \cdots, s^{n-1}) \begin{pmatrix} 1 & -c & -c & \cdots & -c \\ 0 & 1 & -c & \cdots & -c \\ \vdots & & \ddots & & \vdots \\ \vdots & & & \ddots & -c \\ 0 & 0 & \cdots & 0 & 1 \end{pmatrix}$$

where $c^2 + s^2 = 1$, $c, s > 0$, $c = 0.2$, $s = 0.6$, and $n = 100$.

2. A 15×10 matrix A is created as follows: $A = xy^T$, where

$$x = \text{round}(10 * \text{rand}(15, 1))$$
$$y = \text{round}(10 * \text{rand}(10, 1))$$

5. (a) Generate randomly a matrix A of order 6 by using the MATLAB command **rand** (6). Now run the following MATLAB command:

$$[U, S, V] = \textbf{svd}(A)$$

Set $S(4, 4) = 0$; compute $B = U * S * V'$. What is rank (B)?

(b) Construct a matrix C of rank 6 as follows:

$$C = \textbf{qr}(\textbf{rand}(6))$$

Verify that $||C - A||_F^2 \geq ||B - A||_F^2$ using the MATLAB command **norm(A,'fro')** for computing the Frobenius norm.

(c) What is the distance of B from A?

(d) Find a matrix D of rank 2 that is closest to A.

(This exercise is based on Theorems 10.6.2 and 10.6.3.)

6. Let

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 0.0100 & 1 & 1 \\ 0 & 0 & 0.0100 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Find the distance of A from the nearest singular matrix. Find a perturbation that will make A singular. Compare the size of this perturbation with $|\sigma_r|$.

7. Let $A = U\Sigma V^T$ be the singular value decomposition (SVD) of a randomly generated 15×10 matrix $A = \textbf{rand}(15, 10)$ obtained by using the MATLAB command $[U, S, V] = \textbf{svd}(A)$. Set $S(8, 8) = S(9, 9) = S(10, 10) = 0$ all equal to zero. Compute $B = U * S * V'$.

Find the best approximation of the matrix B in the form

$$B \approx \sum^{r} x_i y_i^T$$

such that $||B - \sum_{i=1}^{r} x_i y_i^T||_2 =$ minimum, where x_i and y_i are vectors and r is the rank of B.

8. For matrices A and B in Exercise 7, find a unitary matrix Q such that $||A - BQ||_F$ is minimized. (*Hint*: $Q = VU^T$, where $A^TB = U\Sigma V^T$). (Use the MATLAB command **svd** to solve this problem.)

9. (a) Write a MATLAB program called **covsvd** to compute $(A^TA)^{-1}$ using the singular value decomposition. Use it to find $(A^TA)^{-1}$ for the 8×8 Hilbert matrix and compare your results and flop count with those obtained by running **varcovar** from MATCOM.

(b) Using **covsvd**, find the pseudoinverse of A and compare your result with that obtained by running MATLAB command **pinv**. (A is same as in part (a).)

10. Let A be a 10×10 Hilbert matrix and b be a vector generated such that all entries of the vector x of $Ax = b$ are equal to 1. Solve $Ax = b$ using the SVD of A, and compare the accuracy, flop count, and elapsed time with those obtained by **linsyspp** from Chapter 6.

11. Let $A = \text{rand}(10, 3)$, and

$$X = \textbf{pinv}(A)$$

Verify that X satisfies all the four conditions of the pseudoinverse using MATLAB: $AXA = A$, $XAX = X$, $(AX)^T = AX$, $(XA)^T = XA$.

12. Write a MATLAB program called **chansvd** to implement the improved SVD algorithm of Chan given in Section 10.9.3, using MATLAB commands **qr** and **svd**.

$$[U, S, V] = \textbf{chansvd}(A)$$

Run your program with a randomly generated 30×5 matrix $A = \textbf{rand}(30, 5)$ and compare the flop count and elapsed time with those obtained by using **svd**(A).

13. Write a MATLAB program called **bidiag** to bidiagonalize a matrix A (Section 10.9.2):

$$[B] = \textbf{bidiag}(A)$$

where B is a bidiagonal matrix. Test your program using $A = \textbf{rand}(15, 10)$.

14. (The purpose of this exercise is to compare the three related approaches for finding the small singular values of a bidiagonal matrix.) Write a MATLAB program to implement the Demmel–Kahan algorithm for computing the singular values of a bidiagonal matrix:

$$[s] = \textbf{dksvd}(A)$$

where s is the vector containing the singular values of A.

Experiment: Set $A = \textbf{rand}(50, 20)$. Compute $[U, S, V] = \textbf{svd}(A)$. Set $S(10, 10) = S(9, 9) = S(8, 8) = 10^{-5}$. Compute $B = U * S * V'$. Run the program **bidiag** on B:

$$C = \textbf{bidiag}(B)$$

Compute the singular values of C using (i) **svd**, (ii) **dksvd**, and (iii) **chansvd** and compare the results with respect to accuracy (especially for the smallest singular

Note: Some of the MATLAB programs you have been asked to write are already in MATCOM, but it is a good idea to write your own programs.

Suggestions for Further Reading

Our proof of the SVD theorem (Theorem 10.2.1) is rather traditional. For a different and elegant proof of this classical, important theorem, see Pan and Sigmon (1994). See also Horn and Johnson (1991) in this context.

As mentioned before, applications of the SVD are varied. There are books in each area of applications where the SVD plays an important role. For applications of the SVD to classical control problem, see an earlier survey of Klema and Laub (1980).

The SVD also plays an important role in modern control theory, especially in H-infinity and robust control theory. Curious readers are referred to a growing number of interesting papers in these areas. For applications of the SVD to robust pole assignment problem, see Kautsky, Nichols, and Van Dooren (1985).

For some applications of the SVD to system identification and signal-processing problems, see an interesting survey paper, " Variety of Applications of Singular Value Decomposition" by Joos Vandewalle and Bart De Moor (1988).

Two important books in the area of SVD and signal processing are: (1) *SVD and Signal Processing, Algorithms, Applications, and Architecture*, edited by Ed. F. Deprettere (1988), (2) *SVD and Signal Processing II, Algorithms, Analysis, and Applications*, edited by R. Vaccaro (1991).

For applications to image processing, see the book *Fundamentals of Digital Image Processing* by A. K. Jain, (1989) and *Digital Image Processing* by H. C. Andrews and B. R. Hunt (1977).

Many matrices arising in signal processing and control, and systems theory applications are structured, such as Hankel or Toeplitz. Finding efficient and reliable algorithms for different types of computations that can exploit the structures of these matrices is a challenging problem to researchers. Much work has been done, and it is presently an active area of research. For some interesting papers in this context, see the book *Linear Algebra in Signals, Systems, and Control*, edited by B. N. Datta et al. (1988). See also the recent paper of Van Dooren (1991).

For some statistical applications, see the excellent survey papers by Golub (1969) and Hammarling (1985).

Discussions on mathematical applications of the SVD, such as finding numerical rank, nearness to singularity, and orthonormal bases for the range and null space, are contained in all numerical linear algebra books. The books by Golub and Van Loan (1989) and Watkins (1991), in particular, have treated these aspects very well. For discussions on singular values sensitivity, see Stewart (1979a) and (1991).

For computational aspects of singular values and singular vectors, the original paper by Golub and Kahan (1965) and the recent papers by Demmel and Kahan (1990) and Fernando and Parlett (1994) are valuable. The LAPACK routine **SGESVD** computes all or part of the singular values and singular vectors of a general nonsymmetric matrix. The reduction to bidiagonal form is done by the subroutine **SGEBRD**, while the SVD of the bidiagonal matrix is computed by the subroutine **SBDSQR**.

The implicit QR algorithm of Section 10.9.2 for bidiagonal matrices has recently been extended by Chandrasekaran and Ipsen (1994d) to find the SVD of triangular matrices.

A natural generalization of the Jacobi method for computing the eigenvalues of a symmetric matrix is the classical Kogbetliantz's algorithm for computing the singular value decomposition. For more recent papers, see Charlier and Van Dooren (1987), Charlier, Vanbegin, and Van Dooren (1988), Paige (1986b), and Bai (1988). For a comprehensive list, see Golub and Van Loan (1989, 458).

The generalized SVD is a straightforward generalization of the SVD. For aspects of other variants of the SVD and different other generalizations, see De Moor and Zha (1991) and De Moor and Van Dooren (1992).

For the SVD of the product of two matrices, see Heath, Laub, Paige, and Ward (1986) and De Moor (1992).

For the relationship between the generalized SVD and the generalized eigenvalue problem, see Kågström (1985).

CHAPTER ELEVEN

A Taste of Roundoff Error Analysis

Here we give the readers a taste of roundoff error analysis in matrix computations by giving backward analyses of some basic computations such as solutions of triangular systems, LU factorization using Gaussian elimination, and solution of a linear system.

Let's recall that by backward error analysis we mean an analysis that shows that the computed solution by the algorithm is an exact solution of a perturbed problem. When the perturbed problem is close to the original problem, we say that the algorithm is *backward stable*.

11.1 Basic Laws of Floating-Point Arithmetic

We first remind the readers of the basic laws of floating-point arithmetic that will be used in the sequel. These laws were obtained in Chapter 2. Let $|\delta| \leq \mu$, where μ is the machine precision. Then

$$\mathrm{fl}(x \pm y) = (x \pm y)(1 + \delta) \tag{11.1.1}$$

$$\mathrm{fl}(xy) = xy(1 + \delta) \tag{11.1.2}$$

$$\text{If } y \neq 0, \quad \text{then } \mathrm{fl}\left(\frac{x}{y}\right) = \left(\frac{x}{y}\right)(1 + \delta) \tag{11.1.3}$$

Occasionally, we will also use

$$\mathrm{fl}(x * y) = \frac{x * y}{1 + \delta} \tag{11.1.4}$$

where * denotes any of the arithmetic operations $+, -, \times$, or $\div$.

11.2 Backward Error Analysis for Forward Elimination and Back Substitutions

Case 1. Lower Triangular System

Consider solving the lower triangular system

$$Ly = b \tag{11.2.1}$$

where

$$L = (l_{ij}), \qquad b = (b_1, \ldots, b_n)^T \tag{11.2.2}$$

and

$$y = (y_1, \ldots, y_n)^T$$

using the forward-elimination scheme.

We will use $\hat{s}$ to denote a computed quantity of s.

Step 1:

$$\hat{y}_1 = \mathrm{fl}\left(\frac{b_1}{l_{11}}\right) = \frac{b_1}{l_{11}(1 + \delta_1)} \qquad \text{(using 11.1.4)},$$

where

$$|\delta_1| \leq \mu.$$

This gives

$$l_{11}(1 + \delta_1)\hat{y}_1 = b_1$$

that is,

$$\hat{l}_{11}\hat{y}_1 = b_1$$

where

$$\hat{l}_{11} = l_{11}(1 + \delta_1)$$

This shows that $\hat{y}_1$ is the exact solution of an equation whose coefficient is a number close to l_{11}.

Step 2: Similarly,

$$\hat{y}_2 = \mathrm{fl}\left(\frac{b_2 - l_{21}\hat{y}_1}{l_{22}}\right) = \mathrm{fl}\left(\frac{b_2 - \mathrm{fl}(l_{21}\hat{y}_1)}{l_{22}}\right) \tag{11.2.3}$$

$$= \frac{(b_2 - l_{21}\hat{y}_1(1 + \delta_{11}))(1 + \delta_{22})}{l_{22}(1 + \delta_2)} \tag{11.2.4}$$

(using both (11.1.2) and (11.1.4)), where $|\delta_{11}|$, $|\delta_{21}|$, and $|\delta_2|$ are all less than or equal to μ. Equation (11.2.4) can be rewritten as

$$l_{21}(1 + \delta_{11})(1 + \delta_{22})\hat{y}_1 + l_{22}(1 + \delta_2)\hat{y}_2 = b_2(1 + \delta_{22})$$

that is,

$$l_{21}(1 + \epsilon_{21})\hat{y}_1 + l_{22}(1 + \epsilon_{22})\hat{y}_2 = b_2$$

where

$$\epsilon_{21} = \delta_{11}, \qquad \epsilon_{22} = \left(\frac{\delta_2 - \delta_{22}}{1 + \delta_{22}}\right)$$

(neglecting $\delta_{11}\delta_{22}$, which is small). Thus, we can say that $\hat{y}_1$ and $\hat{y}_2$ satisfy

$$\hat{l}_{21}\hat{y}_1 + \hat{l}_{22}\hat{y}_2 = b_2 \tag{11.2.5}$$

where

$$\hat{l}_{21} = l_{21}(1 + \epsilon_{21}) \quad \text{and} \quad \hat{l}_{22} = l_{22}(1 + \epsilon_{22})$$

Step k: The preceding can easily be generalized, and we can say that at the kth step, the unknowns y_1 through y_k satisfy

$$\hat{l}_{k1}\hat{y}_1 + \hat{l}_{k2}\hat{y}_2 + \cdots + \hat{l}_{kk}\hat{y}_k = b_k \tag{11.2.6}$$

where $\hat{l}_{kj} = l_{kj}(1 + \epsilon_{kj})$, $j = 1, \ldots, k$.

The process can be continued until $k = n$.

Thus, we see that the computed $\hat{y}_1$ through $\hat{y}_n$ satisfy the following perturbed triangular system:

$$\begin{aligned} \hat{l}_{11}\hat{y}_1 &= b_1 \\ \hat{l}_{21}\hat{y}_1 + \hat{l}_{22}\hat{y}_2 &= b_2 \\ &\vdots \\ \hat{l}_{n1}\hat{y}_1 + \hat{l}_{n2}\hat{y}_2 + \cdots + \hat{l}_{nn}\hat{y}_n &= b_n \end{aligned}$$

where $\hat{l}_{kj} = l_{kj}(1 + \epsilon_{kj})$, $k = 1, \ldots, n$, $j = 1, \ldots, k$. Note that $\epsilon_{11} = \delta_1$.

These equations can be written in matrix form

$$\hat{L}\hat{y} = (L + \Delta L)\hat{y} = b \tag{11.2.7}$$

where ΔL is a lower triangular matrix whose (i, j)th element $(\Delta L)_{ij} = l_{ij}\epsilon_{ij}$.

Knowing the bounds for ϵ_{ij}, the bounds for Δ_{ij} can be easily computed. For example, if n is small enough so that $n\mu < \frac{1}{10}$, then $|\epsilon_{kj}| \le 1.06(k - j + 2)\mu$ (see Chapter 2, Section 2.5). Then

$$|(\Delta L)_{ij}| \le 1.06(i - j + 2)\mu|l_{ij}| \tag{11.2.8}$$

The preceding can be summarized in the following theorem.

THEOREM 11.2.1 The computed solution $\hat{y}$ to the $n \times n$ lower triangular system $Ly = b$ obtained by forward elimination satisfies a perturbed triangular system:

$$(L + \Delta L)\hat{y} = b$$

where the entries of ΔL satisfy (11.2.8), assuming that $n\mu < \frac{1}{10}$.

Case 2. Upper Triangular System

The roundoff error analysis of solving an upper triangular system

$$Ux = c$$

using back substitution is similar to case 1. We can state the following.

THEOREM 11.2.2 Let U be an $n \times n$ upper triangular matrix and c be a vector. Then the computed solution $\hat{x}$ to the system

$$Ux = c$$

using back substitution satisfies

$$(U + \Delta U)\hat{x} = c \tag{11.2.9}$$

where

$$|(\Delta U)_{ij}| \le 1.06(j - i + 2)\mu|u_{ij}| \tag{11.2.10}$$

assuming $n\mu < \frac{1}{10}$.

11.3 Backward Error Analysis for Triangularization by Gaussian Elimination

The treatment here follows very closely the one given in Ortega (1990) and in Forsythe and Moler (1967).

Recall that the process of triangularization using Gaussian elimination consists of $n - 1$ steps. At step k, the matrix $A^{(k)}$, which is triangular in the first k columns, is constructed; that is

$$A^{(k)} = \begin{pmatrix} a_{11}^{(k)} & \cdots & & \cdots & a_{1n}^{(k)} \\ & \ddots & & & \vdots \\ & & a_{kk}^{(k)} & \cdots & a_{kn}^{(k)} \\ & & \vdots & \ddots & \vdots \\ & & a_{nk}^{(k)} & \cdots & a_{nn}^{(k)} \end{pmatrix} \tag{11.3.1}$$

The final matrix, $A^{(n-1)}$, is triangular. We shall assume that the quantities $a_{ij}^{(k)}$ are the computed numbers.

First, let's analyze the computations of the entries of $A^{(1)}$ from A in the first step. Let the computed multipliers be

$$-\hat{m}_{i1}, \qquad i = 2, 3, \ldots, n.$$

Then

$$\hat{m}_{i1} = \text{fl}\left(\frac{a_{i1}}{a_{11}}\right) = \frac{a_{i1}}{a_{11}}(1 + \delta_{i1})$$

where

$$|\delta_{i1}| \le \mu$$

Thus, the error $e_{i1}^{(0)}$ in setting $a_{i1}^{(1)}$ to zero is given by

$$\begin{aligned} e_{i1}^{(0)} &= a_{i1}^{(1)} - a_{i1} + \hat{m}_{i1}a_{11} \\ &= 0 - a_{i1} + \left(\frac{a_{i1}}{a_{11}}\right)(1 + \delta_{i1})a_{11} \\ &= \delta_{i1}a_{i1} \end{aligned} \tag{11.3.2}$$

Let us now find the errors in computing the other elements $a_{ij}^{(1)}$ of $A^{(1)}$. The computed elements $a_{ij}^{(1)}$, $i, j = 2, \ldots, n$ are given by

$$\begin{aligned} a_{ij}^{(1)} &= \mathrm{fl}(a_{ij} - \mathrm{fl}(\hat{m}_{i1}a_{ij})) \\ &= (a_{ij} - \mathrm{fl}(\hat{m}_{i1}a_{1j}))(1 + \alpha_{ij}^{(1)}) \\ &= [a_{ij} - \hat{m}_{i1}a_{1j}(1 + \beta_{ij}^{(1)})](1 + \alpha_{ij}^{(1)}), \qquad i, j = 2, \ldots, n \end{aligned} \tag{11.3.3}$$

where

$$|\alpha_{ij}^{(1)}| \le \mu, \qquad |\beta_{ij}^{(1)}| \le \mu$$

Equation (11.3.3) can be rewritten as

$$a_{ij}^{(1)} = (a_{ij} - \hat{m}_{i1}a_{1j}) + e_{ij}^{(0)}, \qquad i, j = 2, \ldots, n \tag{11.3.4}$$

where

$$e_{ij}^{(0)} = \frac{\alpha_{ij}^{(1)}a_{ij}^{(1)}}{1 + \alpha_{ij}^{(1)}} - \hat{m}_{i1}a_{1j}\beta_{ij}^{(1)}, \quad i, j = 2, \ldots, n \tag{11.3.5}$$

Noting that the first row of $A^{(1)}$ is the same as the first row of A, we have from (11.3.4) and (11.3.5)

$$\boxed{A^{(1)} = A - L^{(0)}A + E^{(0)} \qquad (11.3.6)}$$

where

$$L^{(0)} = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ \hat{m}_{21} & 0 & \cdots & 0 \\ \vdots & & \ddots & \\ \hat{m}_{n1} & 0 & \cdots & 0 \end{pmatrix}, \qquad E^{(0)} = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ e_{21}^{(0)} & & \cdots & e_{2n}^{(0)} \\ \vdots & & & \\ e_{n1}^{(0)} & & \cdots & e_{nn}^{(0)} \end{pmatrix}$$

Analogously, at the end of step 2, we will have

$$\boxed{A^{(2)} = A^{(1)} - L^{(1)}A^{(1)} + E^{(1)} \qquad (11.3.7)}$$

where $L^{(1)}$ and $E^{(1)}$ are similarly defined.

Substituting (11.3.6) in (11.3.7), we have

$$\boxed{\begin{aligned} A^{(2)} &= A^{(1)} - L^{(1)}A^{(1)} + E^{(1)} \\ &= A - L^{(0)}A + E^{(0)} - L^{(1)}A^{(1)} + E^{(1)} \end{aligned}}$$

Continuing in this way, we can write

$$\begin{aligned} A^{(n-1)} &+ L^{(0)}A + L^{(1)}A^{(1)} + \cdots + L^{(n-2)}A^{(n-2)} \\ &= A + E^{(0)} + E^{(1)} + \cdots + E^{(n-2)} \end{aligned} \tag{11.3.8}$$

Because

$$L^{(k-1)} = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ & 0 & & \\ \vdots & \hat{m}_{k+1,k} & \ddots & \\ \vdots & \vdots & & \vdots \\ 0 & \hat{m}_{n,k} & \cdots & 0 \end{pmatrix}$$

we have

$$L^{(k)}A^{(k)} = L^{(k)}A^{(n-1)}, \qquad k = 0, 1, 2, \ldots, n-2 \tag{11.3.9}$$

Thus from (11.3.8) and (11.3.9), we obtain

$$\boxed{\begin{aligned} A^{(n-1)} &+ L^{(0)}A^{(n-1)} + L^{(1)}A^{(n-1)} + \cdots + L^{(n-2)}A^{(n-1)} \\ &= A + E^{(0)} + E^{(1)} + \cdots + E^{(n-2)} \end{aligned} \qquad (11.3.10)}$$

or

$$\boxed{\begin{aligned} &A + E^{(0)} + E^{(1)} + \cdots + E^{(n-2)} \\ &\quad = (I + L^{(0)} + L^{(1)} + \cdots + L^{(n-2)})A^{(n-1)} \end{aligned} \qquad (11.3.11)}$$

Noting now that

$$\begin{aligned} I + L^{(0)} + L^{(1)} + \cdots + L^{(n-2)} &= \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ \hat{m}_{21} & 1 & 0 & \cdots & 0 \\ \hat{m}_{31} & \hat{m}_{32} & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ \hat{m}_{n1} & \hat{m}_{n2} & \cdots & \hat{m}_{n,n-1} & 1 \end{pmatrix} \\ &= \hat{L} \text{ (computed } L) \end{aligned} \tag{11.3.12}$$

and $A^{(n-1)} = \hat{U}$ (computed U) and denoting $E^{(0)} + E^{(1)} + \cdots + E^{(n-2)}$ by E, we have, from (11.3.11) and (11.3.12),

$$\boxed{A + E = \hat{L}\hat{U} \qquad (11.3.13)}$$

where the matrices $E^{(0)}, \ldots, E^{(n-2)}$ are given by

$$E^{(k-1)} = \begin{pmatrix} 0 & 0 & 0 & \cdots & \cdots & 0 \\ 0 & 0 & 0 & \cdots & \cdots & 0 \\ \vdots & \vdots & \vdots & e_{k+1,k}^{(k-1)} & \cdots & e_{k+1,n}^{(k-1)} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & e_{n,k}^{(k-1)} & \cdots & e_{n,n}^{(k-1)} \end{pmatrix}, k = 1, 2, \ldots, n-1 \tag{11.3.14}$$

$$e_{ik}^{(k-1)} = a_{ik}^{(k-1)} \delta_{ik} \qquad i = k+1, \ldots, n \tag{11.3.15}$$

$$e_{ij}^{(k-1)} = \frac{\alpha_{ij}^{(k)}}{1 + \alpha_{ij}^{(k)}} a_{ij}^{(k)} - \hat{m}_{ik} a_{kj}^{(k-1)} \beta_{ij}^{(k)}, \qquad i, j = k+1, \ldots, n \tag{11.3.16}$$

and

$$|\delta_{ik}| \le \mu, \qquad |\alpha_{ij}^{(k)}| \le \mu \tag{11.3.17}$$

and

$$|\beta_{ij}^{(k)}| \le \mu \tag{11.3.18}$$

We formalize this discussion in the following theorem:

THEOREM 11.3.1 The computed upper and lower triangular matrices $\hat{L}$ and $\hat{U}$ produced by Gaussian elimination satisfy

$$A + E = \hat{L}\hat{U}$$

where $\hat{U} = A^{(n-1)}$ and $\hat{L}$ is lower triangular matrix of computed multipliers:

$$\hat{L} = \begin{pmatrix} 1 & 0 & \cdots & \cdots & 0 \\ \hat{m}_{21} & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \hat{m}_{n1} & \hat{m}_{n2} & \cdots & \hat{m}_{n,n-1} & 1 \end{pmatrix}$$

Example 11.3.1 Using two-digit arithmetic in the computations of L and U, find the error matrix E such that $A + E = \hat{L}\hat{U}$.

$$A = \begin{pmatrix} 0.21 & 0.35 & 0.11 \\ 0.11 & 0.81 & 0.22 \\ 0.33 & 0.22 & 0.39 \end{pmatrix}$$

Step 1:

$$a_{22}^{(1)} = 0.81 - 0.52 \times 0.35 = 0.63$$

$$\hat{m}_{21} = -\frac{0.11}{0.21} = -0.52 \quad a_{23}^{(1)} = 0.22 - 0.52 \times 0.11 = 0.16$$

$$\hat{m}_{31} = \frac{0.33}{0.21} = 1.57 \quad a_{32}^{(1)} = 0.22 - 1.57 \times 0.35 = -0.33$$

$$a_{33}^{(1)} = 0.39 - 1.57 \times 0.11 = 0.22$$

$$A^{(1)} = \begin{pmatrix} 0.21 & 0.35 & 0.11 \\ 0 & 0.63 & 0.16 \\ 0 & -0.33 & 0.22 \end{pmatrix}$$

$$e_{21}^{(0)} = 0 - [0.11 - 0.52 \times 0.21] = -0.0008$$

$$e_{22}^{(0)} = 0.63 - [0.81 - 0.52 \times 0.35] = 0.0020$$

$$e_{23}^{(0)} = 0.10 - [0.22 - 0.52 \times 0.11] = -0.0020$$

$$e_{31}^{(0)} = 0 - [0.33 - 1.57 \times 0.21] = -0.0003$$

$$e_{32}^{(0)} = -0.33 - [0.22 - 1.57 \times 0.35] = -0.0005$$

$$e_{33}^{(0)} = 0.22 - [0.39 - 1.57 \times 0.11] = 0.0027$$

$$E^{(0)} = \begin{pmatrix} 0 & 0 & 0 \\ -0.0008 & 0.0020 & -0.0028 \\ -0.0003 & -0.0005 & 0.0027 \end{pmatrix}$$

Step 2:

$$m_{32} = -\frac{0.33}{0.63} = -0.52, \qquad a_{33}^{(2)} = 0.22 + 0.52 \times 0.16 = 0.30$$

$$A^{(2)} = \begin{pmatrix} 0.21 & 0.35 & 0.11 \\ 0 & 0.63 & 0.16 \\ 0 & 0 & 0.30 \end{pmatrix} = \hat{U}$$

$$e_{32}^{(1)} = 0 - [-0.33 + 0.52 \times 0.63] = 0.0024$$

$$e_{33}^{(1)} = 0.30 - [0.22 + 0.52 \times 0.16] = -0.0032$$

$$E^{(1)} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0.0024 & -0.0032 \end{pmatrix}$$

Thus,

$$E = E^{(0)} + E^{(1)} = \begin{pmatrix} 0 & 0 & 0 \\ -0.0008 & 0.0020 & -0.0028 \\ -0.003 & 0.0019 & -0.0005 \end{pmatrix}$$

Because

$$\hat{L} = \begin{pmatrix} 1 & 0 & 0 \\ 0.52 & 1 & 0 \\ 1.57 & -0.52 & 1 \end{pmatrix}$$

we can easily verify that $\hat{L}\hat{U} = A + E$.

Bounds for the Elements of E

We now assess how large the entries of the error matrix E can be. For this purpose we assume that pivoting has been used in Gaussian elimination so that $|\hat{m}_{ik}| \le 1$. Recall from Chapter 3 and Chapter 5 that the growth factor ρ is defined by

$$\rho = \frac{\max_{i,j,k} |a_{ij}^{(k)}|}{\max_{i,j} |a_{ij}|}$$

Let $a = \max_{i,j} |a_{ij}|$. Then from (11.3.15)–(11.3.18) we have

$$|e_{ik}^{(k-1)}| \le a\rho\mu, \qquad k = 1, 2, \ldots, n-1,\ i = k+1, \ldots, n$$

and, for $i, j = k+1, \ldots, n, (k = 1, 2, \ldots, n-1)$

$$|e_{ij}^{(k-1)}| \le \frac{\mu}{1-\mu}|a_{ij}^{(k)}| + \mu|a_{ij}^{(k-1)}| \le \frac{2}{1-\mu}a\rho\mu \qquad \text{(because } |\hat{m}_{ik}| \le 1)$$

Denote $\mu/(1-\mu)$ by η. Then

$$\begin{aligned}
|E| &= |E^{(0)} + \cdots + E^{(n-2)}| \\
&\le |E^{(0)}| + \cdots + |E^{(n-2)}| \\
&\le a\rho\eta \left[\begin{pmatrix} 0 & 0 & \cdots & 0 \\ 1 & 2 & \cdots & 2 \\ \vdots & \vdots & & \vdots \\ 1 & 2 & \cdots & 2 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 2 & \cdots & 2 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 1 & 2 & \cdots & 2 \end{pmatrix} \right. \\
&\quad \left. + \cdots + \begin{pmatrix} 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 1 & 2 \end{pmatrix} \right] \\
&\le a\rho\eta \begin{pmatrix} 0 & 0 & \cdot & \cdots & 0 \\ 1 & 2 & 2 & \cdots & 2 \\ 1 & 3 & 4 & \cdots & 4 \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & 3 & 5 & \cdots & 2n-2 \end{pmatrix} \qquad (11.3.19)
\end{aligned}$$

Remark: The inequalities (11.3.18) hold elementwise. We can immediately obtain a bound in terms of norms. Thus,

$$\begin{aligned}
\|E\|_\infty &\le a\rho\eta(1 + 3 + \cdots + (2n-2)) \\
&\le a\rho n^2\eta \qquad (11.3.20)
\end{aligned}$$

11.4 Backward Error Analysis for Solving $Ax = b$

We are now ready to give a backward roundoff error analysis for solving $Ax = b$ using triangularization by Gaussian elimination, followed by forward elimination and back substitution.

First, from Theorem 11.3.1, we know that triangularization of A using Gaussian elimination yields $\hat{L}$ and $\hat{U}$ such that $A + E = \hat{L}\hat{U}$.

These $\hat{L}$ and $\hat{U}$ will then be used to solve

$$\hat{L}y = b$$
$$\hat{U}x = y$$

From Theorem 11.2.1 and Theorem 11.2.2, we know that computed solution $\hat{y}$ and $\hat{x}$ of these two triangular systems satisfy

$$(\hat{L} + \Delta L)\hat{y} = b \quad \text{and} \quad (\hat{U} + \Delta U)\hat{x} = \hat{y}$$

From these equations, we have

$$(U + \Delta U)\hat{x} = (L + \Delta L)^{-1}b$$

or

$$(\hat{L} + \Delta L)(\hat{U} + \Delta U)\hat{x} = b$$

or

$$(A + F)\hat{x} = b \qquad (11.4.1)$$

where

$$F = E + (\Delta L)\hat{U} + \hat{L}(\Delta U) + (\Delta L)(\Delta U) \qquad (11.4.2)$$

(Note that $A + E = \hat{L}\hat{U}$.)

Bounds for F

Taking $\|\cdot\|_\infty$ in (11.4.2), we have

$$\|F\|_\infty \le \|E\|_\infty + \|\Delta L\|_\infty \|\hat{U}\|_\infty + \|\hat{L}\|_\infty \|\Delta U\|_\infty + \|\Delta L\|_\infty \|\Delta U\|_\infty$$

We now obtain expressions for $\|\Delta L\|_\infty$, $\|\Delta U\|_\infty$, $\|\hat{L}\|_\infty$ and $\|\hat{U}\|_\infty$. Because

$$\hat{L} = \begin{pmatrix} 1 & & & \\ \hat{m}_{21} & \ddots & 0 & \\ \vdots & & \ddots & \\ \hat{m}_{n1} & \cdots & \hat{m}_{n,n-1} & 1 \end{pmatrix}$$

from (11.2.8), we obtain

$$|\Delta L| \le 1.06\mu \begin{pmatrix} 2 & & 0 \\ 3|\hat{m}_{21}| & 2 & \\ \vdots & & \ddots \\ (n+1)|\hat{m}_{n1}| & \cdots & 3|\hat{m}_{n,n-1}| \quad 2 \end{pmatrix} \qquad (11.4.3)$$

Assuming partial pivoting—that is, $|\hat{m}_{ik}| \le 1$, $k = 1, 2, \ldots, n-1$, $i = k+1, \ldots, n$—we have

$$\|\hat{L}\|_\infty \le n \qquad (11.4.4)$$

$$\|\Delta L\|_\infty \le \frac{n(n+3)}{2}(1.06\mu) \qquad (11.4.5)$$

Similarly,

$$\|\hat{U}\|_\infty \le na\rho \tag{11.4.6}$$

(Note that $U = A^{(n-1)}$.)

and from (11.2.10), we have

$$\|\Delta U\|_\infty \le \frac{n(n+3)}{2} 1.06 a\rho\mu \tag{11.4.7}$$

(Note that $\max |u_{ij}| \le a\rho$.)

Also recall that

$$\|E\|_\infty \le n^2 a\rho \frac{\mu}{1-\mu} \tag{11.4.8}$$

Assume that $\frac{(n+3)^2}{4} \cdot (1.06)^2\mu < 1$, then

$$\|\Delta L\|_\infty \cdot \|\Delta U\|_\infty \le n^2 \rho a \mu \tag{11.4.9}$$

Using (11.4.4)–(11.4.8) in (11.4.2), we then have

$$\begin{aligned}\|F\|_\infty &\le \|E\|_\infty + \|\Delta L\|_\infty \|\hat{U}\|_\infty + \|\hat{L}\|_\infty \|\Delta U\|_\infty \\ &\quad + \|\Delta L\|_\infty \|\Delta U\|_\infty \\ &\le n^2 a\rho \frac{\mu}{1-\mu} + 1.06 n^2 (n+3) a\rho\mu + n^2 \rho a \mu \end{aligned} \tag{11.4.10}$$

Because $\frac{1}{1-\mu} < 1.06$ and $a \le \|A\|_\infty$, from (11.4.10) we can write

$$\|F\|_\infty \le 1.06(n^3 + 5n^2)\rho \|A\|_\infty \mu \tag{11.4.11}$$

Neglecting the terms involving $n^2\mu$, the discussion leads to the following theorem.

THEOREM 11.4.1 The computed solution $\hat{x}$ to the linear system

$$Ax = b$$

using Gaussian elimination with partial pivoting satisfies a perturbed system

$$(A + F)\hat{x} = b$$

where F is defined by (11.4.2) and $\|F\|_\infty \le cn^3\rho\|A\|_\infty\mu$, where c is a constant of order unity.

Remarks:

1. The preceding bound for F is grossly overestimated. In practice, this bound for F is very rarely attained. Wilkinson (1965) states that in practice, $\|F\|_\infty$ is always less than or equal to $n\mu\|A\|_\infty$.
2. Making use of (11.2.8), (11.2.10), and (11.3.14)–(11.3.19), we can also obtain an elementwise bound for F (Exercise 4).

11.5 Review and Summary

In this chapter, we have presented backward error analyses for

1. Lower and upper triangular systems using forward elimination and back substitution (Theorems 11.2.1 and 11.2.2).
2. LU factorization using Gaussian elimination with partial pivoting (Theorem 11.3.1).
3. Linear systems problem $Ax = b$ using Gaussian elimination with partial pivoting followed by forward elimination and back substitution (Theorem 11.4.1).

Bounds for the error matrix E in each case have been derived (11.2.8), (11.2.10), (11.3.19), (11.4.11).

We have merely attempted here to give the reader a taste of roundoff error analysis, as the title of the chapter suggests. The results of this chapter are already known to the reader. They have been stated earlier in the book without proof. We have tried to give formal proofs here.

To repeat, these results say that the forward-elimination and back-substitution methods for triangular systems are backward stable, whereas the stability of the Gaussian elimination process for LU factorization—and, therefore, for the linear system problem $Ax = b$ using the process—depend upon the size of the growth factor.

EXERCISES

1. Using $\beta = 10$, and $t = 2$, compute the LU decomposition using Gaussian elimination (without pivoting) for the following matrices, and find the error matrix E in each case such that

$$A + E = LU$$

(a) $A = \begin{pmatrix} 3 & 4 \\ 5 & 6 \end{pmatrix}$

(b) $\begin{pmatrix} 0.25 & 0.79 \\ 0.01 & 0.12 \end{pmatrix}$

(c) $\begin{pmatrix} 10 & 9 \\ 8 & 5 \end{pmatrix}$

(d) $\begin{pmatrix} 4 & 1 \\ 1 & 2 \end{pmatrix}$

(e) $\begin{pmatrix} 0.01 & 0.05 \\ 0.03 & 0.01 \end{pmatrix}$

2. Suppose now that partial pivoting has been used in computing the LU factorization of each of the matrices of Exercise 1. Find the error matrix E in each case, and compare the bounds of the entries in E predicted by (11.3.19) with the actual errors.
3. Consider the problems of solving linear systems

$$Ax = b$$

using Gaussian elimination with partial pivoting with each of the matrices from Exercise 1 and taking

$$b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

in each case. Find F in each case such that the computed solution x satisfies

$$(A + F)x = b$$

Compare the bounds predicated by (11.4.11) with actual errors.

4. Making use of (11.2.8), (11.2.10), and (11.3.13)–(11.3.19), find an elementwise bound for F in Theorem 11.4.1:

$$(A + F)\hat{x} = b$$

Also find a bound for the residual.

5. From Theorems 11.2.1 and 11.2.2, what can you say about the backward stability of the processes of forward elimination and back substitution for lower and upper triangular systems?
6. From (11.3.20), conclude that the backward stability of Gaussian elimination is essentially determined by the size of the growth factor ρ.
7. Consider the problem of evaluating the polynomial

$$p(\alpha) = a_n\alpha^n + a_{n-1}\alpha^{n-1} + \cdots + a_0$$

by synthetic division:

$$p_n = a_n$$
$$p_{i-1} = \mathrm{fl}(\alpha p_i + a_{i-1}), \qquad i = n, n-1, \ldots, 1$$

Then $p_0 = p(\alpha)$. Show that

$$p_0 = a_n(1 + \delta_n)\alpha^n + a_{n-1}(1 + \delta_{n-1})\alpha^{n-1} + \cdots + a_0(1 + \delta_0)$$

Find a bound for each δ_i, $i = 0, 1, \ldots, n$. What can you say about the backward stability of the algorithm from your bounds?

8. Show that the Cholesky algorithm (Algorithm 6.4.4) computes a lower triangular matrix $\hat{L}$ such that $A + E = \hat{L}(\hat{L})^T$. Find a bound for the elements of the matrix E. Hence prove that the Cholesky algorithm is backward stable.

Suggestions for Further Reading

The most authoritative book for learning details of backward error analyses of algorithms for matrix computations is Wilkinson's *Algebraic Eigenvalue Problem* (1965).

Wilkinson's other book in this area, *Rounding Errors in Algebraic Process* (1963), is also a good source of knowledge.

The books *Numerical Analysis—A Second Course*, by James Ortega (1990), and *Computer Solutions of Linear Algebraic Systems*, by Forsythe and Moler (1967), also have in-depth treatment of the material of this chapter. A complete book devoted to error analysis is presently under preparation by N.J. Higham (forthcoming).

APPENDIX A

A Brief Introduction to MATLAB

A.1 Some Basic Information on MATLAB

A.1.1 What is MATLAB?

MATLAB stands for MATrix LABoratory. It is an interactive software package for solving problems arising in scientific and engineering computations. MATLAB contains programs for all fundamental matrix computations, such as solutions of linear systems, various matrix factorizations, solutions of least squares problems, eigenvalues and eigenvector computations, and singular value and singular vector computation. The most current version contains programs for many other types of computations, including 2-D and 3-D graphic capabilities.

A.1.2 Entering and Exiting MATLAB

On most systems the command **matlab** will let you enter into MATLAB. Give the command **exit** to quit MATLAB.

A.1.3 Two Important Commands: help and demo

The two most important commands are **help** and **demo**. Typing **help** you will get the listing of all the MATLAB functions and other MATLAB capabilities. Typing **help** followed by a MATLAB function name from the list will give you more specific information about that particular function.

Example A.1.1

```
>> help norm

 NORM Matrix or vector norm.
  For matrices..
   NORM(X) is the largest singular value of X, max(svd(X)).
   NORM(X,2) is the same as NORM(X).
   NORM(X,1) is the 1-norm of X, the largest column sum,
                  = max(sum(abs((X)))).
```

```
   NORM(X,inf) is the infinity norm of X, the largest row sum,
                  = max(sum(abs((X')))).
   NORM(X,'inf') is same as NORM(X,inf).
   NORM(X,'fro') is the F-norm, sqrt(sum(diag(X'*X))).
   NORM(X,P) is available for matrix X only if P is 1, 2, inf
                  or 'fro'.

  For vectors..
   NORM(V,P) = sum(abs(V)^P)^(1/P).
   NORM(V) = norm(V,2).
   NORM(V,inf) = max(abs(V)).
   NORM(V,-inf) = min(abs(V)).

In MATLAB 4.0, if X has complex components, z, then
abs(z) = sqrt(z*conj(z)), not abs(real(z)) + abs(imag(z)),
which was used in some earlier versions of MATLAB.

See also COND, RCOND, CONDEST, NORMEST.
```

Demo teaches you how to use the MATLAB functions, such as how to enter matrix values into a matrix, how to find its transpose, how to find the rank of a matrix, and so on.

A.1.4 Most Frequently Used MATLAB Operations and Functions

Some basic matrix operations

+	Plus
–	Minus
*	Matrix multiplication
.*	Array multiplication
^	power
\	Backslash or left division
/	Slash or right division
&	Logical AND
|	Logical OR
~	Logical NOT

Matrix Functions: Numerical Linear Algebra

Matrix analysis

cond	Matrix condition number
norm	Matrix or vector norm
rcond	LINPACK reciprocal condition estimator
rank	Number of linearly independent rows or columns
det	Determinant
trace	Sum of diagonal elements
null	Null space
orth	Orthogonalization
rref	Reduced row echelon form

Linear equations

\ and /	Linear equation solution; use "help slash"
chol	Cholesky factorization
lu	Factors from Gaussian elimination
inv	Matrix inverse
qr	Orthogonal-triangular decomposition
qrdelete	Delete a column from the QR factorization
qrinsert	Insert a column in the QR factorization
pinv	Pseudoinverse
lscov	Least squares in the presence of known covariance

Eigenvalues and singular values

eig	Eigenvalues and eigenvectors
poly	Characteristic polynomial
hess	Hessenberg form
svd	Singular-value decomposition
qz	Generalized eigenvalues
rsf2csf	Real block diagonal form to complex diagonal form
cdf2rdf	Complex diagonal form to real block diagonal form
schur	Schur decomposition
balance	Diagonal scaling to improve eigenvalue accuracy

Using **help**, you can get information on any of the above routines. Here is an example.

Example A.1.2

```
>> help lu

  LU   Factors from Gaussian elimination.
   [L,U] = LU(X) stores an upper triangular matrix in U and a
   "psychologically lower triangular matrix," i.e., a product
   of lower triangular and permutation matrices, in L, so
   that X = L*U.

   [L,U,P] = LU(X) returns lower triangular matrix L, upper
   triangular matrix U, and permutation matrix P so that
   P*X = L*U.

   By itself, LU(X) returns the output from LINPACK'S ZGEFA
   routine.
```

A.1.5 Numerical Examples

To enter a matrix and a vector in MATLAB:

```
>> A=[1 3 5; 2 4 6; 1 3 9]
```

```
A =

     1     3     5
     2     4     6
     1     3     9

>> b=[1 1 1]'

b =

     1
     1
     1
```

To solve the linear system $Ax = b$:

```
>> x=A\b

x =

   -0.5000
    0.5000
         0
```

To compute the inverse of A:

```
>> inv(A)

ans =

   -2.2500    1.5000    0.2500
    1.5000   -0.5000   -0.5000
   -0.2500         0    0.2500
```

To compute the eigenvalues of A:

```
>> eig(A)

ans =

   -0.3246
   12.3246
    2.0000
```

To reduce A to an upper Hessenberg matrix:

```
>> hess(A)

ans =

    1.0000   -4.9193    3.1305
   -2.2361    8.6000   -6.2000
         0   -3.2000    4.4000
```


To find the rank of A:

```
>> rank(A)

ans =

     3
```

To compute the 2-norm of A:

```
>> norm(A)

ans =

   13.3538
```

To compute the condition number of A (with respect to 2-norm):

```
>>cond(A)

ans =

    42.1539
```

To compute the QR factorization of A using Householder matrices:

```
>> [q,r] = qr(A) will give the qr factorization of A

q =

   -0.4082    0.5774   -0.7071
   -0.8165   -0.5774   -0.0000
   -0.4082    0.5774    0.7071

r =

   -2.4495   -5.7155  -10.6145
         0    1.1547    4.6188
         0         0    2.8284
```

To compute the lu factorization of A using partial pivoting:

```
>> [l,u] = lu(A)

l =

    0.5000    1.0000         0
    1.0000         0         0
    0.5000    1.0000    1.0000

u =

     2     4     6
     0     1     2
     0     0     4
```

A.2 Writing Your Own Programs Using MATLAB Commands

It is easy to write your own programs using MATLAB commands. We first list some most common uses and then give examples.

> We urge you to obtain the *MATLAB Primer* by Professor Kermit Sigmon of the University of Florida, which gives a more complete introduction to efficient use of MATLAB. It is available via mail from:
> CRC Press, Inc.
> 2000 Corporate Blvd., N.W.
> Boca Raton, FL 33431
> USA
>
> and via telephone from:
> 1-800-272-7737 continental USA, except Florida
> 1-407-994-0555 Florida and outside continental USA

A.2.1 Some Relational Operators in MATLAB

`<`	Less than
`>`	Greater than
`<=`	Less than or equal to
`>=`	Greater than or equal to
`==`	Equal

A.2.2 Some Matrix-Building Functions

eye	Identity matrix
zeros	Matrix of zeros
rand	Randomly generated matrix with entries between zero and one
max	Max entry
diag	Create or extract a diagonal
triu	Upper triangular part of a matrix
tril	Lower triangular part of a matrix
hilb	Hilbert matrix
toeplitz	Toeplitz matrix
size	Gives the dimension of a matrix
abs	Absolute value of the elements of a matrix or a vector

Example A.2.1

1. rand(5, 3) will create a 5×3 randomly generated matrix.
2. If x is a vector, diag(x) is the diagonal matrix with entries of x on the diagonal; diag(A) is the vector consisting of the diagonal of A.
3. hilb(5) will create the 5×5 Hilbert matrix.
4. max(A(:, 2)) will give the maximum value of the second column of A; max(max(A)) will give the maximum entry of the whole matrix A.

A.2.3 Colon Notation

Colon notation (:) is used to conveniently indicate a whole row or column.

Example A.2.2

A(:, 2) denotes the entire 2nd column of A.
A(1 : 3, :) denotes the first 3 rows of A.
A(:, [2, 5, 7]) denotes the second, fifth, and seventh columns of A.

The statement A(:, [2, 5, 7]) = B(:, 1:3) will replace the columns 2, 5, 7 of A with the first three columns of B.

A.2.4 for, while, if Commands

The **for**, **while**, and **if** commands are most useful in writing MATLAB programs for matrix algorithms. The commands are illustrated in the following examples.

A.2.5 Computing Flop Count and Elapsed Time of an Algorithm

To compute the flop count of an algorithm set **flops(0)** immediately before executing the algorithm and type **flops** immediately after completion.

Example A.2.3

```
flops(0)
x  = a\b
flops
```

gives the total flops to solve a linear system with a given matrix a and a vector b.

The function **cputime** returns the CPU time in seconds that has been used by the MATLAB process since the MATLAB process started. For example,

```
t = cputime;your_operation;cputime - t
```

returns the cputime to run your operation. Because the PC version of MATLAB does not have a **cputime** function, MATCOM contains a program cputime.m. If the version of MATLAB that you use contains the **cputime** function, then you can delete the cputime.m program from MATCOM.

A.2.6 Saving a MATLAB Program

The command **save** ⟨*filename*⟩ will store all variables in the filename 'filename.mat'.

The command **load** ⟨*filename*⟩ will restore all the variables from the file named 'filename.mat'.

A.2.7 Getting a Hard Copy

The use of **diary**⟨*filename*⟩ will cause a copy of all subsequent terminal input and most of the resulting output to be written on the named file; **diary off** suspends it.

A.2.8 Examples of Some Simple MATLAB Programs

Example A.2.4 The following code will make the elements below the diagonal of the 4×4 matrix $A = a(i, j)$ zero.

```
a = rand(4,4)
for i = 1:4
  for j = 1:4
      if i > j
         a(i,j) = 0;
      end;
  end;
end;
```

Example A.2.5 The following code will create a matrix A such that the (i, j)th entry of the matrix $A = a(i, j)$ is $(i + j)$.

```
a = zeros(4,4)
for i = 1:4
  for j = 1:4
      a(i,j) = i+j;
  end;
end;
```

Example A.2.6 The following MATLAB program computes the matrix–matrix product of two upper triangular matrices. This program will be called **matmat.m**.

```
% Matrix-Matrix product with upper triangular matrices
% input U and V two upper triangular matrices of order n
% output C = U * V
% function  C = matmat(U,V)
  function  C = matmat(U,V)
        [n,m] = size(U)
         C = zeros(n,n)
         for i = 1:n
          for j = i:n
            for k = i:j
                C(i,j) = C(i,j) + U(i,k) * V(k,j);
            end;
          end;
         end;

  end;
```

Example A.2.7 (MATLAB Implementation of Algorithm 3.1.1)

```
% Computing  the two norm of a vector x
% input x
% output  nrm, the two norm of the vector x
% function  nrm = twonorm(x)
  function  nrm = twonorm(x)
        [n,m] = size(x);
        r = max(abs(x));
```



```
y=x/r;
s = 0;
for i=1:n
 s = s + y(i)^2;
end;
nrm = r * s^0.5;
end;
```

A.2.9 Executing an M-File

The files of the form ⟨*filename*⟩.m are called M-files. These files can be ordinary ASCII files and can be created using an editor. Once created, these files can be run inside MATLAB. There are two types of M-files: **Scripts** and **Function** files.

For a script file all variables are global, whereas for a function file all variables are local. When a script file is run in MATLAB, commands are executed immediately, without waiting for the inputs to be inserted. A function file has the word "function" in its first line.

```
function [ output1, output2, ...., outputk] = program
name(input1, input2, ...., inputr).
```

Such a file can be executed in MATLAB just by specifying the input and output arguments.

For example, to run the M-file matmat.m, supply the input matrices U and V and then type C = matmat(U, V).

A.2.10 Use of *diary* Command and Printing the Output

You may want a listing of your MATLAB program step by step as it is executed. The **diary** command can be used to create a listing.

Example A.2.8

```
>>diary B:diary8      <----- sets the diary on
>> C = matmat(U,V)    <----- executes your program
>> diary off          <----- sets the diary off
```

The above command will store all the commands that are executed by the program matmat.m in the file diary8. Only that data printed on the screen is stored in diary8. This file can then be printed. In case you do not want the output to be printed on the screen, place a semicolon at the end of the line.

To write a comment line, put the % sign in the first column. Examples of some M files are given in Appendix B.

APPENDIX B

MATCOM and Selected MATLAB Programs from MATCOM

B.1 What is MATCOM?

MATCOM is a MATLAB-based interactive software package containing the implementation of all the major algorithms of Chapter 3 through Chapter 10 of this text.

For each problem considered in this book, there is more than one (in some cases several) algorithm.

By using the programs in MATCOM, students will be able to compare different alogrithms for the same problem with respect to accuracy, flop count, and elapsed time, for instance. Students will be able to verify the statements about goodness and badness of different algorithms. In particular, they will be able to distinguish between a good and a bad algorithm. MATCOM was written by Conrad Fernandes, one of Professor Datta's graduate students.

B.2 Using MATCOM

Section B3 gives a chapter-by-chapter listing of the programs in MATCOM.

Example B.2.1 Suppose that you want to use the program HOUSZERO from MATCOM. To use this program in MATLAB, you need to know the input and output variables. To find out these variables, type

```
>> help houszero

HOUSZERO Householder zeroing in a vector.
```

[u, sigma] = houszero(x) creates a vector u defining a Householder matrix H such that $Hx = (\sigma, 0, \ldots, 0)^T$. The program implements Algorithm 5.4.1.

```
input   vector x
output u and sigma
function [u,sigma] = houszero(x)
```

APPENDIX B

MATCOM and Selected MATLAB Programs from MATCOM

B.1 What is MATCOM?

MATCOM is a MATLAB-based interactive software package containing the implementation of all the major algorithms of Chapter 3 through Chapter 10 of this text.

For each problem considered in this book, there is more than one (in some cases several) algorithm.

By using the programs in MATCOM, students will be able to compare different alogrithms for the same problem with respect to accuracy, flop count, and elapsed time, for instance. Students will be able to verify the statements about goodness and badness of different algorithms. In particular, they will be able to distinguish between a good and a bad algorithm. MATCOM was written by Conrad Fernandes, one of Professor Datta's graduate students.

B.2 Using MATCOM

Section B3 gives a chapter-by-chapter listing of the programs in MATCOM.

Example B.2.1 Suppose that you want to use the program HOUSZERO from MATCOM. To use this program in MATLAB, you need to know the input and output variables. To find out these variables, type

```
>> help houszero

HOUSZERO Householder zeroing in a vector.
```

[u, sigma] = houszero(x) creates a vector u defining a Householder matrix H such that $Hx = (\sigma, 0, \ldots, 0)^T$. The program implements Algorithm 5.4.1.

```
input   vector x
output u and sigma
function [u,sigma] = houszero(x)
```

The input is a vector x and the outputs are a vector u and a scalar sigma.
To execute the program you have to do the following

1. Create the input vector x.
2. Then type

```
[u,sigma] = houszero(x)
```

As output you will get the vector u defining the Householder matrix H such that Hx is a multiple of e_1.

```
>> x = rand(4,1)

x =

    0.2190
    0.0470
    0.6789
    0.6793

>> [u,sigma] = houszero(x)

u =

    1.7740
    0.0693
    0.9994
    1.0000

sigma = -0.9861
```

B.3 Chapter-by-Chapter Listing of MATCOM Programs

Chapter 3

Title	Program Name	Algorithm or Section
Back substitution	BACKSUB	Algorithm 3.1.3 (p.69)

Chapter 4

Title	Program Name	Algorithm or Section
Premultiplication by a Householder matrix	PHOUSMUL	Algorithm 4.2.1 (p.97)
Postmultiplication by a Householder matrix	HOUSMULP	
The Inverse of an upper triangular matrix	INVUPTR	Algorithm 4.2.2 (p.99)
Basic Gaussian elimination	GAUSS	Algorithm 4.2.3 (p.100)

Chapter 5

Title	Program Name	Algorithm or Section
Triangularization using Gaussian elimination without pivoting	LUGSEL	Algorithm 5.2.1 (p.118)
Triangularization using Gaussian elimination with partial pivoting	PARPIV	Algorithm 5.2.2 (p. 126)
Triangularization using Gaussian elimination with complete pivoting	COMPIV	Algorithm 5.2.3 (p.129)
Householder zeroing in a vector	HOUSZERO	Algorithm 5.4.1 (p.135)
Householder QR factorization	HOUSQR	Algorithm 5.4.3 (p.142)
Householder QR factorization for a nonsquare matrix	HOUSQRN	Section 5.4.2 (p.145)
Householder Hessenberg reduction	HOUSHESS	Algorithm 5.4.4 (p.150)
Givens zeroing in a vector	GIVZERO and GIVROT	Algorithm 5.5.1 (p.154)
Premultiplication by a Givens matrix	PGIVMUL	Algorithm 5.5.2 (p.155)
Givens specific zeroing in a vector	GIVSZERO	Algorithm 5.5.3 (p.158)
QR factorization using Givens rotations	GIVQR	Algorithm 5.5.4 (p.160)
Givens Hessenberg reduction	GIVHESS	Algorithm 5.5.5 (p.164)

Chapter 6

Title	Program Name	Algorithm or Section
Forward elimination	FORELM	Algorithm 6.4.1 (p.208)
Linear system solution without explicit factorization	GAUSSWF	Algorithm 6.4.2 (p.213)
Cholesky algorithm	CHOLES	Algorithm 6.4.4 (p.223)
Sherman–Morrison formula	SHERMOR	Section 6.5.2 (p.239)
Inverse by LU factorization without pivoting	INLU	

Inverse by partial pivoting	INPARPIV	
Inverse by complete pivoting	INCOMPIV	
Hager's norm-1 condition number estimator	HAGCOND1	Algorithm 6.7.1 (p.258)
Iterative refinement	ITERREF	Algorithm 6.9.1 (p.262)
The Jacobi method	JACOBI	Algorithm 6.10.1 (p.268)
The Gauss–Seidel method	GAUSED	Algorithm 6.10.2 (p.270)
The Successive overrelaxation method	SUCOV	Algorithm 6.10.3 (p.278)
The classical conjugate gradient algorithm	CONGRAD	Algorithm 6.10.4 (p.284)
Incomplete Cholesky factorization	ICHOLES	Algorithm 6.10.6 (p.287)
No-fill incomplete LDL^T	NICHOL	Algorithm 6.10.7 (p.287)

Chapter 7

Title	Program Name	Algorithm or Section
Least-squares solution using normal equations	LSFRNME	Algorithm 7.8.1 (p.330)
The Golub–Householder method for the full-rank least-squares problem	LSFRQRH	Algorithm 7.8.2 (p.336)
Classical Gram–Schmidt for QR factorization	CLGRSCH	Algorithm 7.8.3 (p.338)
Modified Gram–Schmidt for QR factorization	MDGRSCH	Algorithm 7.8.4 (p.339)
Least-squares solution by MGS	LSFRMGS	Algorithm 7.8.5 (p.342)
Least-squares solution for the rank-deficient problem using Householder QR	LSRDQRH	Algorithm 7.8.6 (p.344)
Minimum-norm solution for the underdetermined system using normal equations	MNUDNME	Algorithm 7.9.1 (p.350)
Minimum-norm solution for the underdetermined system using Householder QR	MNUDQRH	Algorithm 7.9.2 (p.352)

Linear systems analog least-squares iterative refinement	LSITRN1	Algorithm 7.10.1 (p.353)
Iterative refinement for least-squares solution	LSITRN2	Algorithm 7.10.2 (p.354)
Variance-covariance matrix	VARCOVAR	Algorithm 7.11.1 (p.359)

Chapter 8

Title	Program Name	Algorithm or Section
Power method	POWER	Algorithm 8.5.1 (p.408)
Inverse iteration	INVITR	Algorithm 8.5.2 (p.411)
Rayleigh quotient iteration	RAYQOT	Algorithm 8.5.3 (p.416)
Reciprocal of the condition numbers of eigenvalues	SENSEIG	Section 8.7.2 (p.427)
The basic QR iteration	QRITRB	Section 8.9.1 (p.435)
The Hessenberg QR iteration	QRITRH	Section 8.9.2 (p.437)
The explicit single-shift QR iteration	QRITRSSE	Section 8.9.4 (p.439)
The explicit double-shift QR iteration	QRITRDSE	Section 8.9.5 (p.441)
One iteration step of the implicit double-shift QR	QRITRDSI	Algorithm 8.9.1 (p.446)
Symmetric Lanczos algorithm	LANSYM	Algorithm 8.12.2 (p.470)

Chapter 9

Title	Program Name	Algorithm or Section
Hessenberg triangular reduction	HESSTRI	Section 9.3.1 (p.498)
Inverse iteration for eigenvectors of the pencil (A, B)	INVITRGN	Algorithm 9.4.1 (p.505)
Cholesky QR algorithm for the symmetric definite pencil	CHOLQR	Algorithm 9.5.1 (p.508)
Schur QR algorithm	SCHURQR	Algorithm 9.5.2 (p.508)
Simultaneous diagonalization of a symmetric definite pencil	SIMDIAG	Algorithm 9.5.3 (p.510)

Generalized Rayleigh quotient iteration	GENRAYQT	Algorithm 9.5.4 (p.511)
Sturm sequence method	GENSTURM	Section 9.9.1 (p.534)
Quadratic eigenvalue problem via standard reduction	QUADEIG1	Section 9.8 (p.531)
Quadratic eigenvalue problem via reduction to a generalized eigenvalue problem	QUADEIG2	Section 9.8 (p.532)
Lanczos algorithm for the symmetric definite pencil	LANSYMGN	Algorithm 9.9.2 (p.536)

Chapter 10

Title	Program Name	Algorithm or Section
Orthogonal projections using the SVD	ORTHPROJ	Section 10.6.2 (p.563)
Variance-covariance matrix using the SVD	COVSVD	Section 10.7 (p.570)
Minimum-norm least-squares solution using the SVD	MINNMSVD	Algorithm 10.8.1 (p.572)
Least-squares solutions using the SVD	LSQRSVD	Algorithm 10.8.1 (p.572)
Reduction to bidiagonal form	BIDIAG	Algorithm 10.9.2 (p.578)

No Chapter

Title	Program Name
The absolute maximum of a vector	ABSMAX
Interchange two vectors	INTER
Computing the CPU time	CPUTIME

B.4 Some Selected MATCOM Programs

```
  function y = backsub(T,b);
%BACKSUB Back substitution
%y = backsub(T,b) computes the solution y of a nonsingular
%upper triangular system Ty = b using back substitution
%process. This program implements Algorithm 3.1.3 of the book.
%input  : Matrix T and vector b
%output : vector y
```



```
[m,n] = size(T);
if m~=n
disp('matrix T  is not square');
return
end;
y = zeros(n,1);
for i = n:-1:1
  sum = 0;
  for j = i+1:n
  sum = sum + T(i,j)*y(j);
  end;
y(i) = (b(i) - sum ) / T(i,i);
end;

end;
```

```
  function t = invuptr(t);
%INVUPTR Inverse of a nonsingular upper triangular matrix
%T = invuptr(T) computes the inverse of a nonsingular upper
%triangular matrix T.  T is overwritten by its inverse.
%This program implements Algorithm 3.1.4 of the book.
%input  : Matrix t
%output : Matrix t

        [m,n] = size(t);
        if m~=n
        disp('matrix T  is not square')
        return
        end;
        for k = n:-1:1
          t(k,k) = 1/t(k,k);
          for i = k-1 :-1 :1
           sum = 0;
           for j = i+1:k
            sum = sum + t(i,j)*t(j,k);
           end;
           t(i,k) = -sum/t(i,i);
          end;
        end;

        end;
```

```
  function a = phousmul(a,u)
%PHOUSMUL Pre multiplication by a Householder matrix
%A = phousmul(A,u) computes the pre multiplication
%of a matrix A by a Householder matrix generated
%by the vector u.
%On output A contains the product HA.
%This program implements Algorithm 4.2.1 of the book.
%input   : Matrix a and vector u
%output  : Matrix a
```

```
        [m1,n] = size(a);

        beta = 2/(u'*u);
        for j = 1 : n
          alpha = 0;
          for i = 1 : m1
            alpha =  alpha + u(i) * a(i,j);
          end;
          alpha =  beta * alpha;
          for i = 1:m1
            a(i,j) = a(i,j) - alpha * u(i);
          end;
         end;

        end;

  function [l,u] = lugsel(a)
%LUGSEL LU factorization using Gaussian elimination without
%pivoting [L,U] = lugsel(A) produces the LU factorization of
%a matrix A using Gaussian elimination without pivoting :
%A = LU.  L is unit lower triangular and U is upper
%triangular.  This program implements Algorithm 5.2.1 of
%the book.
%input  : Matrix A
%output : Matrices L and U

        [m1,n] = size(a);
        for k = 1 : n-1
           for i = k+1:n
             a(i,k) =  a(i,k)/ a(k,k);
           end;
           for i = k+1:n
              for j = k+1 :n
                a(i,j)  = a(i,j)  - a(i,k) * a(k,j);
              end;
           end;
        end;
        u = triu(a);
        l =  tril(a,-1);
        for i = 1:n
        l(i,i) = 1;
        end;

        end;

  function [x,sigma] = houszero(x)
%HOUSZERO Householder zeroing in a vector.
%[x,sigma] = houszero(x) creates a vector u defining
% a Householder matrix H such that  Hx = [ sigma, 0,...,0 ]
%This program implements Algorithm 5.4.1 of the book.
%input  :  vector x
%output :  vector x and scalar sigma
```



```
        [m1,n] = size(x);
         m = max(abs(x));
         x = x/m;
         s = sign(x(1));
         if s == 0
          s = 1;
         end;
         sigma = s * norm(x,2);
         x = x + sigma * eye(m1,1);
         u = x;
         sigma = -m * sigma;
        end;

  function x = gausswf(A,b);
%GAUSSWF  Linear system solution without explicit
%factorization x = gausswf(A,b) computes the solution x
%of the system Ax = b using Gaussian elimination with
%partial pivoting.  The factorization MA = U is never
%explicitly formed.  The program calls the MATCOM program
%BACKSUB and INTER.  The program implements Algorithm
%6.4.2 of the book.
%input  : Matrix A and vector b
%output : vector x

        tv1 = cputime
        flops(0)
        [m,n] = size(A);
        if m~=n
        disp('matrix A  is not square')
        return
        end;
        s1 = b;
        U = zeros(n,n);
        for k = 1:n-1
        M1 = eye(n,n);
          d = k;
          s = A(k,k);

          for i = k+1:n
              if abs(A(i,k)) > abs(s)
                 s = A(i,k);
                 d = i;
              end;
          end;
          if (s == 0)
            disp('the algorithm has encountered a zero
                  pivot')
            x =[];
            return
          end;
         p(k) = d;
```

```
        if d ~= k
         [A(k,k:n),A(d,k:n)] = inter(A(k,k:n),A(d,k:n));
         [b(k),b(d)] = inter(b(k),b(d));
        end;
         'inter'
        for i = k+1:n
        M1(i,k) = -A(i,k)/A(k,k);
        A(i,k) = M1(i,k);
        end;
        for j = k+1:n
         for i = k+1:n
          A(i,j) = A(i,j) +M1(i,k) * A(k,j);
         end;
        end;
         for i = k+1:n
          b(i)   = b(i) + M1(i,k) * b(k);
         end;
      end;
      U = triu(A)
      x = backsub(U,b)

      tv2 = cputime - tv1;
      disp('the cpu time in seconds taken for this
            algorithm')
      tv2
      disp('the flop count for this algorithm is')
      flps = flops

      end;

  function x = lsfrqrh(A,b);
%LSFRQRH Least squares by the Golub-Householder method
%x = lsfrqrh(A,b) computes the least squares solution x
%to the full rank overdetermined system Ax = b, by the
%Golub-Householder method.  This program implements Algorithm
%7.8.2 of the book.  The program calls MATCOM programs backsub
%and phousmul.
%input  : Matrix A and vector b
%output : vector x
      [m1,n] = size(A);
      y = zeros(n,1);
      s= min(n,m1-1);

      for k = 1 : s
        [u,sigma] = houszero(A(k:m1,k));
        A(k:m1,k:n) = phousmul(A(k:m1,k:n),u);
        b(k:m1) = phousmul(b(k:m1),u);
      end;
      R1 =  A;
      c = b;
      ran = rank(r1);
      R = R1(1:ran,:);
```

APPENDIX C

A List of Commonly Used LAPACK Routines

In Chapter 4, we mentioned that the software package LAPACK implements the block versions of all major numerical linear algebra algorithms. These routines were designed mainly for high-performance computers. However, LAPACK now exists for most computer systems, including work stations. We therefore list a few commonly used LAPACK routines, for easy reference.

Routine	Description
SBDSQR	Computes the singular value decomposition (SVD) of a real bidiagonal matrix, using the bidiagonal algorithm.
SGEBAL	Balances a general matrix to improve the accuracy of computed eigenvalues.
SGEBRD	Reduces a general rectangular matrix to real bidiagonal form by an orthogonal/unitary transformation.
SGECON	Estimates the reciprocal of the condition number of a general matrix, in either the 1-norm or the infinity-norm using the LU factorization computed by SGETRF.
SGEEQU	Computes row and column scalings to equilibrate a general rectangular matrix and reduce its condition number.
SGEES	Computes the eigenvalues and Schur factorization of a general matrix.
SGEEV	Computes the eigenvalues and left and right eigenvectors of a general matrix.
SGEHRD	Reduces a general matrix to upper Hessenberg form by an orthogonal transformation.
SGELS	Computes the least-squares solution to an overdetermined system of linear equations, $AX = B$ or $A^H X = B$, or the minimum-norm solution of an underdetermined system, where A is a general rectangular matrix of full rank, using QR or LQ factorization of A.
SGELSS	Computes the minimum-norm least-squares solution to an over- or underdetermined system of linear equations $AX = B$ using the singular-value decomposition of A.

Routine	Description
SGEQPF	Computes a QR factorization with column pivoting of a general rectangular matrix.
SGEQRF	Computes a QR factorization of a general rectangular matrix.
SGERQF	Computes an RQ factorization of a general rectangular matrix.
SGESV	Solves a general system of linear equations $AX = B$.
SGESVD	Computes the singular value decomposition (SVD) of a general rectangular matrix.
SGETRF	Computes an LU factorization of a general matrix, using partial pivoting with row interchanges.
SGETRI	Computes the inverse of a general matrix, using the LU factorization computed by SGETRF.
SGTSV	Solves a general tridiagonal system of linear equations $AX = B$.
SHSEQR	Computes the eigenvalues and Schur factorization of an upper Hessenberg matrix, using the multishift QR algorithm.
SORGQR	Generates all or part of the orthogonal matrix Q from a QR factorization determined by SGEQRF.
SORMQR	Multiplies a general matrix by the orthogonal/unitary matrix from a QR factorization determined by SGEQRF.
SPOSV	Solves a symmetric/Hermitian positive definite system of linear equations.
SPOTRF	Computes the Cholsky factorization of a symmetric/Hermitian positive definite matrix.
SPOTRS	Solves a symmetric/Hermitian positive definite system of linear equations $AX = B$, using the Cholesky factorization computed by SPOTRF.
SPTSV	Solves a symmetric/Hermitian positive definite tridiagonal system of linear equations $AX = B$, and provides an estimate of the condition number and error bounds on the solution.
SPTTRF	Computes the LDL^H factorization of a symmetric/Hermitian positive definite tridiagonal matrix.
SSYGV	Computes all the eigenvalues and eigenvectors of a generalized symmetric/Hermitian definite generalized eigenproblem, $Ax = \lambda Bx$, $ABx = \lambda x$, or $BAx = \lambda x$.
SSYTRD	Reduces a symmetric/Hermitian matrix to real symmetric tridiagonal form by an orthogonal/unitary similarity transformation.
SSYTRF	Computes the factorization of a real symmetric/Hermitian indefinite matrix, using the diagonal pivoting method.
STRCON	Estimates the reciprocal of the condition number of a triangular matrix, in either the 1-norm or the infinity-norm.
STRSEN	Reorders the Schur factorization of a matrix in order to find an orthonormal basis of a right invariant subspace corresponding to selected eigenvalues, and returns reciprocal condition numbers (sensitivities) of the average cluster of eigenvalues and of the

Key Terms in Numerical Linear Algebra

accumulation of inner product in double precision: Computation of the inner product of two vectors using double precision, except for the final step when the final result is rounded to single precision.

algorithm: An ordered set of operations, logical and/or arithmetic, which when applied to a computational problem defined by a given set of data, called the input data, produces a solution to the problem.

back substitution: The process of solving an upper triangular system $Tx = b$. The entries of the column vector x are obtained one by one, starting from the bottom and working up through the elements of the vector.

backward stability: An algorithm is backward stable if the computed solution obtained by the algorithm is the exact solution of a nearby problem.

balancing: A process applied to A, before the start of the QR iteration algorithm, so that the entries of the matrix A become somewhat uniform.

BLAS: Basic Linear Algebra Subroutines. There are three levels: Level 1 subroutines are for vector-vector operations, Level 2 for matrix-vector operations, and Level 3 for matrix-matrix operations.

catastrophic cancellation: A phenomenon that occurs when two numbers that are nearly equal are subtracted. Many significant digits are lost.

characteristic polynomial: For an $n \times n$ matrix A, the polynomial $p(\lambda) = det(A - \lambda I)$, of degree n in λ.

Cholesky factorization: The factorization of a symmetric positive definite matrix A into HH^T, where H is a lower triangular matrix with positive diagonal entries.

companion matrix: An unreduced upper Hessenberg matrix with ones along the subdiagonal and possible nonzero entries on the last column and zeros everywhere else is an upper companion matrix. The coefficients of the characteristic polynomial of the upper companion matrix are given by the entries of the last row. The transpose of an upper companion matrix is a lower companion

complete pivoting: The form of pivoting where the search for the pivot entry at step k is made among the entries of the submatrix below the row $(k-1)$ during the Gaussian elimination process.

condition number: The number indicating the sensitivity of a problem. The condition number of the linear system problem $Ax = b$ is: $Cond(A) = \|A\|\|A^{-1}\|$. It is also called the condition number of the matrix A.

condition number of eigenvalue: If λ is an eigenvalue of a nondefective matrix, then the number $\frac{1}{|y_i^T x_i|}$, where y_i and x_i are the left and right unit length eigenvectors of the matrix, respectively, is the condition number of λ.

deflation: A technique used in eigenvalue computations: once an eigenvalue (or a pair of eigenvalues) is computed, an appropriate row and column (or pair of rows and columns) are deleted and the computations proceed with the remaining submatrix for the remaining eigenvalues.

dense matrix: A matrix with most entries nonzero. The zero entries are too few to devise any specialized algorithm.

direct method: A method that produces the solution of a problem in a finite number of steps. The Gaussian elimination, QR factorization, and Cholesky factorization methods are examples of direct methods for the linear system problem.

double-shift QR iteration: The shifted QR iteration where two single-shifts are used successively.

efficiency of an algorithm: An algorithm involving matrices of order n is efficient if it does not require more than order n^3 floating-point operations for implementation.

eigenvalue and eigenvector: A scalar λ is an eigenvalue of A if there exists a nonzero vector x such that $Ax = \lambda x$. x is called the eigenvector corresponding to λ.

elementary lower triangular matrix: A matrix of the form $I + m_k e_k^T$, where $m_k = (0, \ldots, 0, m_{k+1,k}, \ldots, m_{n,k})^T$. It is an identity matrix, except possibly for a few nonzero entries below the diagonal of a single column.

floating-point number: A floating-point number x has the form $x = \pm\gamma\beta^e$, where e is the exponent, γ is the significand, and β is the base of the number system. A floating-point number x is denoted by $fl(x)$.

flop: Floating-point operation [a coupled multiplication (division) and addition (subtraction)].

forward elimination: The process of solving a lower triangular system $Ly = b$. The entries of the unknown vector x are obtained one by one, starting from the top and working down through the vector's elements.

forward stability: An algorithm is forward stable if the computed solution $\hat{x}$ by the algorithm is close to the exact solution x in some sense.

Frobenius norm: $\|A\|_F = \left[\sum_{j=1}^{n}\sum_{i=1}^{m}|a_{ij}|^2\right]^{1/2}$

Gaussian elimination: An elimination process used to solve a linear system. It is named after the celebrated mathematician Karl Friedrich Gauss.

Gauss–Seidel method: An iterative method for solving $Ax = b$, named after two mathematicians, Gauss and Seidel. The ith component of the $(k+1)$th iteration vector $x_i^{(k+1)}$ is computed using a combination of information from the kth step and that which is available at the current $(k+1)$th step:

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\right)$$

generalized eigenvalue problem: The eigenvalue problem $Ax = \lambda Bx$ involving two matrices A and B.

Geršgorin disks: The disks in the complex plane $\{z : |z - a_{ii}| < r_i\}$, where $r_i = \sum_{j=1, j\neq i}^{n} |a_{ij}|$ is associated with $A = (a_{ij})$. Each eigenvalue of A lies in at least one of the disks.

Givens matrix: A matrix $J(i, j, \theta)$ is a Givens matrix if it is an identity matrix except for the four entries $J_{ii} = c, J_{ij} = s, J_{ji} = -s, J_{jj} = c$, where $c = cos\theta, s = sin\theta$. A Givens matrix is orthogonal.

GMRES method: The Generalized Minimal Residual Method. A Krylov subspace method to solve a large and sparse linear system.

Gram–Schmidt process: A process to generate an orthonormal basis of a subspace, starting from a given basis. If applied to the columns of a matrix A, it yields a QR factorization of A.

growth factor: The ratio of the largest element (in magnitude) of the matrices $A, A^{(1)}, \ldots, A^{(n-1)}$ (obtained during the Gaussian elimination process) to the largest element (in magnitude) of A. The growth factor is an indicator of stability or instability of the Gaussian elimination process.

guard digit: An extra digit in the lower end of the arithmetic register whose purpose is to catch the low-order digit that would otherwise be pushed out of existence when the decimal points are aligned.

Hilbert matrix: The matrix $H = (h_{ij})$, with $h_{ij} = \frac{1}{i+j-1}$, named after the celebrated mathematician David Hilbert. The higher order Hilbert matrices are extremely ill-conditioned.

Householder matrix: A matrix of the form $I - 2\frac{uu^T}{u^Tu}$, where u is a vector. A Householder matrix is symmetric and orthogonal. It is also known as a *projector*.

ill-conditioned problem: A problem is ill-conditioned if a small change in the input data causes a significant change in the solution.

implicit QR iteration: A version of the QR iteration algorithm where one constructs the QR iteration step $A_{s+1} = R_sQ_s + \mu I$ implicitly without forming the matrix $A_s - \mu I$.

incomplete Cholesky factorization: The Cholesky factorization of a sparse symmetric positive definite matrix where calculations are made only with the nonzero entries and nonzero entries in the Cholesky factor are allowed only in those positions that have a nonzero in A.

inertia of a matrix: The inertia of an $n \times n$ matrix A is the triplet $(\pi(A), \nu(A), \delta(A))$; where $\pi(A)$, $\nu(A)$, and $\delta(A)$ are, respectively, the number of eigenvalues of A with positive, negative, and zero real parts.

inverse iteration: An iterative process for finding an eigenvector given an approximation to the eigenvalue for which the eigenvector is to be computed.

inverse power method: The same as *inverse iteration*.

iterative method: A method that produces the solution of a problem after a few number of iterations, starting from an initial approximation of the solution. Usually, the approximations become closer and closer to the solution as the iteration proceeds. The Jacobi, Gauss–Seidel, and successive over relaxation methods are examples of iterative methods for solving linear systems problems. The power method, the inverse power method, and the QR method are examples of iterative methods for solving eigenproblems.

iterative refinement: A procedure to iteratively improve a computed solution of a problem.

Jacobi method: An iterative method for solving the system $Ax = b$, named after the mathematician Carl Gustav Jacobi. The ith component of the $(k+1)$th iteration vector $x_i^{(k+1)}$ is computed using only information from kth iteration step:

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1, i\neq j}^{n} a_{ij}x_j^{(k)}\right)$$

Krylov subspace: The subspace spanned by the sequence of vectors $\{x, Ax, \ldots, A^{m-1}x\}$, m is the dimension of the subspace. The sequence $\{A^i x\}$ is called the Krylov sequence.

Krylov subspace method: A method based on generating an orthonormal basis of a Krylov subspace. The Lanczos, conjugate gradient, GMRES, and the Arnoldi methods are examples of such methods.

LU factorization: A factorization of A into LU, where L is lower triangular and U is upper triangular. Elementary lower triangular matrices are used to achieve an LU factorization.

LAPACK: A mathematical software package for linear algebra computations. The package was developed mainly for portable high-performance computing.

least-squares solution: A solution x to the linear system $Ax = b$ having the property that $\|Ax - b\|_2$ is minimized.

machine precision: The machine precision μ is the smallest positive floating-point number in a computer such that $fl(1+\mu) > 1$.

MATCOM: A MATLAB-based software package implementing all the major algorithms of the book *Numerical Linear Algebra and Applications*.

MATLAB: MATLAB stands for MATrix LABoratory. It is an interactive computing system designed for easy computations of various matrix based scientific and engineering problems.

minimum-norm solution: Among all the least-squares solutions to the linear system $Ax = b$, the one having the minimum norm is the minimum-norm solution.

norm of a matrix: Let A be an $m \times n$ matrix. Then the norm of A, $\|A\|$, is a scalar such that $\|A\| \geq 0$, $\|\alpha A\| = |\alpha|\|A\|$, and $\|A + B\| \leq \|A\| + \|B\|$, where α is a scalar.

norm of a vector: Let x be a vector. Then the norm of x, $\|x\|$, is a scalar associated with x such that $\|x\| \geq 0$, $\|\alpha x\| = |\alpha|\|x\|$, and $\|x + y\| \leq \|x\| + \|y\|$, where α is a scalar.

normal equations: For the system $Ax = b$, the normal equations are the set of

orthogonal projection: P is the orthogonal projection onto a subspace S of $\mathbb{R}^n$ if $\text{range}(P) = S$, $P^2 = P$, and $P^T = P$.

overdetermined system: A linear system having more equations than unknowns.

overflow: A situation that results when an exponent of a computed quantity becomes too large to fit in a given computer.

partial pivoting: The form of pivoting where the search for pivot entry at step k is made in the kth column below the row $(k-1)$ during the Gaussian elimination process.

perturbation analysis: Analysis pertaining to the effect on the solution of a problem on the perturbation of the data of a problem.

power method: An iterative method for computing the dominant eigenvalue and the corresponding eigenvector of a matrix A. The method is so called because it is based on implicitly computing the powers of A.

preconditioning: A process to improve the condition number of a matrix.

pseudoinverse: The matrix $A^\dagger$ defined by $A^\dagger = V\Sigma^\dagger U^T$, where $A = U\Sigma V^T$ is the SVD of A, and $\Sigma^\dagger = \text{diag}(\frac{1}{\sigma_1}, \ldots, \frac{1}{\sigma_r}, 0, \ldots, 0)$, and r is the rank of A.

pseudocode: Form of codes for describing algorithms which can be translated easily into computer codes.

QR iteration: An iterative process for finding the eigenvalues of a matrix A, based on repeated QR factorizations of matrices orthogonally similar to A.

QR factorization: A factorization of A into QR, where Q is orthogonal and R is upper triangular. Householder matrices, Givens matrices, and the Gram–Schmidt process are used to achieve a QR factorization.

QR factorization with column pivoting: The factorization of A in the form:

$$Q^T AP = \begin{pmatrix} R_{11} & R_{12} \\ 0 & 0 \end{pmatrix}$$

where Q is orthogonal, P is a permutation matrix, and R_{11} is upper triangular and nonsingular.

quadratic eigenvalue problem: The eigenvalue problem of the form $(\lambda^2 A + \lambda B + C)x = 0$.

quasi-minimal residual method: A Krylov subspace method to solve a large and sparse linear system. It is popularly known as the QMR method.

QZ algorithm: An algorithm for finding the eigenvalues of the pencil $A - \lambda B$. The orthogonal matrices Q and Z are constructed so that $Q^T AZ$ is an upper quasi-triangular matrix T, and $Q^T BZ = S$ is upper triangular. The eigenvalues are then extracted from the eigenvalues of T and S.

rank-revealing QR: A QR factorization of the form:

$$Q^T AP = \begin{pmatrix} R_{11} & R_{12} \\ 0 & 0 \end{pmatrix}$$

that reveals the rank of A in exact arithmetic. Rank of $A = rank(R_{11})$.

Rayleigh quotient: The quotient $\frac{x^T Ax}{x^T x}$ is called the Rayleigh quotient of the vector x. If x is an approximation of an eigenvector, then the Rayleigh quotient is an approximation of the eigenvalue corresponding to which the approximate eigenvector is x.

Real Schur form: A real quasi-triangular matrix whose diagonal entries are 1×1 or 2×2 matrices. Every real matrix A can be transformed to real Schur form by orthogonal similarity.

relative error: If $\hat{x}$ is an approximation to x, then the relative error is $\frac{\|\hat{x}-x\|}{\|x\|}$.

rounding error: The error made in rounding a computed quantity.

scaling (row): A process by which a diagonal matrix D is constructed so that the rows of $D^{-1}A$ have more or less equal infinity norms.

single-shift QR iteration: The shifted QR iteration where a single shift is used but the shift varies from iteration to iteration.

singular value decomposition: A decomposition of the matrix A in the form $A \doteq U\Sigma V^T$, where U and V are orthogonal and Σ is a "diagonal" matrix.

sparse matrix: A matrix with a large number of zero entries. Sparsity is an asset with a large problem. A sparse matrix may conveniently be stored in a computer and specialized algorithms can be devised.

special matrices: An $n \times n$ matrix $A = (a_{ij})$ is:

diagonal if $a_{ij} = 0, i \neq j$.
upper triangular if $a_{ij} = 0, i > j$.
lower triangular if $a_{ij} = 0, i < j$.
upper Hessenberg if $a_{ij} = 0$ for $i > j + 1$.
lower Hessenberg if $a_{ij} = 0$ for $j > i + 1$.
tridiagonal if it is both lower and upper Hessenberg.
a permutation matrix if there is exactly one nonzero entry in each row and column that is a 1 and the rest are zero.
unreduced upper Hessenberg if A is upper Hessenberg and $a_{i,i-1} \neq 0, i = 2, \ldots, n$.
unreduced lower Hessenberg if A is lower Hessenberg and $a_{i,i+1} \neq 0, i = 1, \ldots, n-1$.
(row) diagonally dominant if $|a_{ii}| > \sum_{j \neq i} |a_{ij}|$, for all i.
symmetric (Hermitian) if $A^T = A (A^* = A)$.
positive definite if A is symmetric (Hermitian) and $x^T Ax > 0$ $(x^* Ax > 0)$, for every nonzero vector x.
defective matrix if it has fewer than n eigenvectors.
convergent matrix if $A^k \to 0$ as $k \to \infty$.

spectral norm: $\|A\|_2 = \sqrt{\text{maximum eigenvalue of } A^T A}$.

spectral radius: The spectral radius is $\max |\lambda_i|, i = 1, \ldots, n$; where $\lambda_1, \ldots, \lambda_n$ are the eigenvalues of A.

spectrum: The set of all the eigenvalues of a matrix.

stopping criterion: A criterion that terminates an iterative method.

successive overrelaxation method: A modified Gauss–Seidel method. The ith component of the $(k+1)$th approximation is given by

$$x_i^{(k+1)} = \frac{\omega}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right) + (1 - \omega) x_i^{(k)}$$

ω is called the relaxation factor.

symmetric definite generalized eigenvalue problem: The eigenvalue problem $Ax = \lambda Bx$, where A and B are symmetric matrices and B is positive definite.

underdetermined system: A linear system having more unknowns than equations.

underflow: A situation that results when the exponent of a computed quantity becomes too small to fit in a given computer.

variance-covariance matrix: The matrix $(A^T A)^{-1}$, where A has full rank.

well-conditioned problem: A problem whose solution is not sensitive to small perturbations of the input data.

Wilkinson bidiagonal matrix: The bidiagonal matrix whose entries along the diagonal are $1, 2, \ldots, 20$; and those along the superdiagonal (subdiagonal) are $20, 19, \ldots, 1$. Certain eigenvalues of this matrix are extremely sensitive.

Wilkinson shift: A special shift, named after James H. Wilkinson, used in the symmetric QR iteration algorithm. If $\alpha_1, \ldots, \alpha_n$ and $\beta_1, \ldots, \beta_{n-1}$ are, respectively, the diagonal and off-diagonal entries of a symmetric tridiagonal matrix, then the Wilkinson shift is given by $\mu = \alpha_n + r - \text{sign}(r)\sqrt{r^2 + \beta_{n-1}^2}$, where $r = \frac{\alpha_{n-1} - \alpha_n}{2}$.

Partial Solutions and Answers to Some Selected Problems

Chapter 1

8. (a) Let $A = (a_1, \ldots, a_n)$ and $x = (x_1, \ldots, x_n)^T$, $null(A) = 0$ if and only if $Ax = 0$ implies $x = 0$. But $Ax = a_1x_1 + \cdots + a_nx_n$. Thus $null(A) = 0$ if and only if $a_1x_1 + \cdots + a_nx_n = 0$ implies $x = 0$, i.e., the columns of A are linearly independent.

14. (a) First we show that $R(I - P_S) = S^\perp$. Let $y = x - P_Sx \in R(I - P_S)$, for some $x \in R^n$. Then $\forall z \in S, z^Ty = z^Tx - z^TP_Sx = z^Tx - z^Tx = 0$. Thus $R(I - P_S) \subseteq S^\perp$. Conversely, $\forall y \in S^\perp$, $P_Sy = 0$. Thus $y = y - P_Sy = (I - P_S)y \in R(I - P_S)$, i.e., $S^\perp \subseteq R(I - P_S)$. Also, $(I - P_S)^T = I - P_S$, $(I - P_S)^2 = I - P_S$.

(b) (i) It is obvious that $R(A(A^TA)^{-1}A^T) \subseteq R(A)$. Let $x \in R(A)$, then $\exists y$, such that $x = Ay$. So $A(A^TA)^{-1}A^T(Ay) = Ay = x$. Thus $x \in R(A(A^TA)^{-1}A^T)$. Hence $R(A(A^TA)^{-1}A^T) = R(A)$. Also, it is easy to see that $A(A^TA)^{-1}A^T$ are idempotent and symmetric.

23. (d) Let R be upper triangular. Then since R is orthogonal, $R^T = R^{-1}$. Noting that R^T is lower triangular and R^{-1} is upper triangular, it is immediate from $R^T = R^{-1}$ that the off-diagonal entries of R are all zero.

30. Since A has full rank, $Ax = 0$ if and only if $x = 0$. Thus for every nonzero x, $x^T(A^TA)x = (Ax)^T(Ax) = y^Ty > 0$ (since $Ax = y \neq 0$).

34. Since A is symmetric, $A^TA = A^2$. Thus, the eigenvalues of A^TA are $\lambda_1^2, \ldots, \lambda_n^2$, where $\lambda_1, \ldots, \lambda_n$ are the eigenvalues of A. Because the singular values of A are the nonnegative square roots of the eigenvalues of A^TA, the singular values of A are then $|\lambda_1|, \ldots, |\lambda_n|$.

38. (a) Let $a = \frac{x}{\|x\|_2}$, $b = \frac{y}{\|y\|_2}$. Then $\|a\|_2 = 1$, $\|b\|_2 = 1$. Now,

$$|a^Tb| \leq \frac{1}{2}(a_1^2 + b_1^2 + \cdots + a_n^2 + b_n^2)$$
$$= \frac{1}{2}(\|a\|^2 + \|b\|^2) = 1$$

Thus $\frac{|x^Ty|}{\|x\|_2\|y\|_2} \leq 1$, i.e. $|x^Ty| \leq \|x\|_2\|y\|_2$.

(b) The inequality holds for either $x = 0$ or $y = 0$. If $x \neq 0$, $y \neq 0$, then $x^T x$, $y^T y \neq 0$. Thus

$$\|xy^T\|_2 = \sqrt{\text{maximum eigenvalue of } (xy^T)^T(xy^T)}$$
$$= \sqrt{x^T x}\sqrt{\text{maximum eigenvalue of } (yy^T)} = \|x\|_2\|y\|_2$$

39. $\|x + y\|_2^2 = (x + y)^T(x + y) = x^T x + 2x^T y + y^T y$
$= x^T x + y^T y \qquad (\text{since } x^T y = 0)$
$= \|x\|_2^2 + \|y\|_2^2$

45. (a) $\|QAP\|_F^2 = \text{trace}(P^T A^T Q^T QAP)$
$= \text{trace}(P^T A^T AP) = \text{trace}(A^T A) = \|A\|_F^2$

(b) $\|QAP\|_2 = \sqrt{\text{maximum eigenvalue of } (P^T A^T Q^T QAP)}$
$= \sqrt{\text{maximum eigenvalue of } (A^T A)} = \|A\|_2$

(Note that $P^T = P^{-1}$, and the eigenvalues remain invariant under similarity transformations.)

49. Let $A = U\Sigma V^T$. Then $\|A\|_2 = \|\Sigma\|_2$. Let $\Sigma = \text{diag}(\sigma_1, \ldots, \sigma_n)$, with $\sigma_1 \geq \cdots \geq \sigma_n$. Then

$$\|\Sigma\|_2 = \max_{x \neq 0} \frac{\|\Sigma x\|_2}{\|x\|_2} = \max_{x \neq 0} \frac{(\sum_{i=1}^n (\sigma_i x_i)^2)^{1/2}}{(\sum_{i=1}^n x_i^2)^{1/2}} \leq \sigma_1$$

But choose $x = e_1 = (1, 0, \ldots, 0)^T$. Then $\frac{\|\Sigma x\|_2}{\|x\|_2} = \sigma_1$. Thus $\|A\|_2 = \|\Sigma\|_2 = \sigma_1$.

51. Let $B = (b_1, \ldots, b_r)$, where $b_1, \ldots, b_r$ are columns of B. Then

$$\|AB\|_F^2 = \|A(b_1, \ldots, b_r)\|_F^2$$
$$= b_1^T A^T Ab_1 + \cdots + b_r^T A^T Ab_r$$
$$= \|Ab_1\|_2^2 + \cdots + \|Ab_r\|_2^2$$
$$\leq \|A\|_F^2(\|b_1\|_2^2 + \cdots + \|b_r\|_2^2) \quad \text{(because of the compatibility of } F\text{-norm, } \|Ax\|_2 \leq \|A\|_F\|x\|_2)$$
$$= \|A\|_F^2\|B\|_F^2$$

Similarly we can prove $\|AB\|_F \leq \|A\|_F\|B\|_2$.

56. Let

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

and $\|\cdot\| = \|\cdot\|_2$, which is subordinate. Then $A^k = 0$, for $k \geq 2$. But $\|A\|_2 = 1$, thus the norm-test fails.

Chapter 2

3. $x = .9999 \times 10^{14}$, $y = .9815 \times 10^{11}$, $z = .9825 \times 10^{11}$, $t = 4$. $x(y + z) = .1899 \times 10^{26}$, $xy + yz = .1799 \times 10^{26}$

5. (a) $S_{1000} = \sum_{k=1}^{1000} \frac{1}{k}$ where each term $\frac{1}{k}$ is rounded to four significant digits. When added in ascending order the sum is 7.486, the same as the answer in four-digit arithmetic. In descending order it is 7.449. Verify this yourself.

14. For $t = 4$,

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

for $t = 9$,

$$fl(A^T A) = \begin{pmatrix} 1 + 10^{-8} & 1 \\ 1 & 1 + 10^{-8} \end{pmatrix}$$

15. (a) $e^x - x - 1 = \dfrac{1}{1 + y + \frac{y^2}{2!} + \cdots} + y - 1$ where $x = -y$, $y > 0$.

(b) $\sqrt{x^4 + 1} - x^2 = \dfrac{1}{\sqrt{x^4 + 1} + x^2}$

(c) $\dfrac{1}{x} - \dfrac{1}{x + 1} = \dfrac{1}{x(x + 1)}$

18. (e) $m = max\begin{pmatrix} |a| \\ |b| \end{pmatrix}$

$$y_i = \frac{a}{m}, \quad y_2 = \frac{b}{m}$$
$$e = m\sqrt{y_1^2 + y_2^2}$$

20. (a) $-b$ is about the same size as $\sqrt{b^2 - 4ac}$. A catastrophic cancellation will occur. Take

$$x_1 = \frac{-b - sign(b)\sqrt{b^2 - 4ac}}{2a} = 10^6$$
$$x_2 = \frac{c}{ax_1} = 10^{-6}$$

Chapter 3

1. (a) $fl(xy) = xy(1 + \delta) = xy'$. Take $x' = x$, $y' = y(1 + \delta)$. Since $|\delta| \leq \mu$, both x' and y' are close to x and y, respectively. Therefore the computation is backward stable.

(b) $fl(x^T y) = x_1 y_1(1 + \gamma) + x_2 y_2(1 + \gamma) + \cdots + x_n y_n(1 + \gamma)$. Let

$$y' = \begin{pmatrix} y_1(1 + \gamma) \\ y_2(1 + \gamma) \\ \vdots \\ y_n(1 + \gamma) \end{pmatrix}$$

take $x' = x$, $y' = y(1 + \delta)$ and use the above argument.

4. $Cond(cA) = \|cA\|\|(cA)^{-1}\| = c\|A\|\dfrac{\|(A)^{-1}\|}{c} = \|A\|\|(A)^{-1}\| = Cond(A)$

$$1 \leq \|I\| = \|AA^{-1}\| \leq \|A\|\|A^{-1}\| = Cond(A)$$

7. The growth factor for each of the matrices is 1.
10. (a) Change the coefficient 3 to 2.99, the computed roots are $1.6327 \pm 1.0493i$ and -0.2655.

Chapter 4

1. (a) Multiplication of two lower triangular matrices.

Input: Two lower triangular matrices A and B
Output: Matrix C, $C = A * B$
For $i = 1, \ldots, n$ do
  For $j = 1, \ldots, i$ do
    For $k = j, \ldots, i$ do
      $c_{ij} = c_{ij} + a_{ik} * b_{kj}$

(c) Multiplication of two tridiagonal matrices.

Input: Two tridiagonal matrices A and B
Output: Matrix C, $C = A * B$
For $i = 1, \ldots, n$ do
  For $j = max(1, i-2), \ldots, min(i+2, n)$ do
    For $k = max(i-1, 1), \ldots, min(i+1, n)$ do
      $c_{ij} = c_{ij} + a_{ik} * b_{kj}$

(d) Multiplication of an arbitrary matrix by an upper Hessenberg matrix

Input: Matrices A and B
Output: $C = A * B$
For $i = 1, \ldots, n$ do
  For $j = 1, \ldots, n$ do
    For $k = 1, \ldots, min(j+1, n)$ do
      $c_{ij} = c_{ij} + a_{ik} * b_{kj}$

3. *Compute:* $[I + xy^T] * A$

Input: $m \times n$ matrix A and vectors x and y
For $j = 1, \ldots, n$ do
  $\alpha = 0$
    For $i = 1, \ldots, m$ do
      $\alpha = \alpha + y_i * a_{ij}$
        For $i = 1, \ldots, m$ do
          $a_{ij} = a_{ij} + \alpha * x_i$

5. Addition of two $n \times n$ symmetric matrices A and B.

Input: Symmetric matrices A and B
Output: $C = A + B$. The algorithm overwrites B with C
For $i = 1, \ldots, n$ do
  For $j = i, \ldots, n$ do
    $b_{ij} = a_{ij} + b_{ij}$
      If $i \neq j$
        $b(j, i) = b(i, j)$

Chapter 5

1. (b) $(I + m_k e_k^T)(I - m_k e_k^T) = I - m_k e_k^T + m_k e_k^T - m_k e_k^T m_k e_k^T = I$ (since the last term is the zero matrix).
2. (a) $E = \begin{pmatrix} 1 & 0 \\ 10^{-5} & 1 \end{pmatrix}$

(b) $U = \begin{pmatrix} 10^{-5} & 1 \\ 0 & -0.999 \times 10^5 \end{pmatrix}$

$L = \begin{pmatrix} 1 & 0 \\ 10^5 & 1 \end{pmatrix}$

13. For $k = 1, 2, \ldots, r$ do

$$a_{ik} \equiv m_{ik} = -\frac{a_{ik}}{a_{kk}}, (i = k+1, \ldots, m)$$

$$a_{ij} \equiv a_{ij} + m_{ik} a_{kj}$$
$$= a_{ij} - a_{ik} a_{kj} (i = k+1, \ldots, m, j = k+1, \ldots, m)$$

E's will be formed out of the multipliers m_{ik} as shown in the book.

14. Write down a set of inequalities that will guarantee that A is column diagonally dominant.
15. Take L and U, respectively, as unit lower triangular and upper triangular matrices. Equate the entries on both sides of $A = LU$.
20. Only one entry needs to be eliminated at each step. Flop count for each step is $O(n)$.
27. $Q^T = Q_{n-1} Q_{n-2} \ldots Q_2 Q_1$. Each Q_i is diagonal except in the entries $(i+1, i)$ and $(i, i+1)$. Thus the product Q is Hessenberg.
29. The following algorithm computes AJ and overwrites A with AJ, given i and k.

For $j = 1, \ldots, n$ do
  $\alpha = a_{ji}$
  $\beta = a_{jk}$
  $a_{ji} = c\alpha - s\beta$
  $a_{jk} = s\alpha + c\beta$.

flops: 4 flops for each j.
Total flops $= 4n$.

35. (a) $A^{(k)} = P_k A P_k^T$. $P_k A$ requires $2(n-k)^2$ flops. $(P_k A) P_k^T$ requires $2n(n-k)$ flops. Follow now the "Hint" given in the problem.

(d) If A is symmetric, the reduced form is symmetric tridiagonal and therefore, only about half the entries need to be computed.

36. (a) Take

$$X = (x_1, x_2, \ldots, x_n), x_1 = (1, 0, \ldots, 0)^T$$

then from $HX = XC$, it follows that $x_i = H^{i-1} e_1$, $i = 1, 2, \ldots, n$. The matrix H is an upper triangular matrix with $h_{21}, h_{21} h_{32}, \ldots, h_{21} \ldots h_{n,n-1}$ as the diagonal entries.

(b) If the subdiagonal entries of H are very small, X will be ill-conditioned.

37. (a) First, find a Householder matrix P_1 such that $P_1 b$ is a multiple of e_1. Form $H_1 = P_1 A P_1^T$. Transform H_1 to an upper Hessenberg matrix using Householder transformations, updating the new vector b every time a Householder transformation is applied in the process.

Chapter 6

1. Let x, y, z, be the required cubic yards. Then $0.6x + 0.4y + 0.2z = 5000$; $0.2x + 0.4y + 0.3z = 5500$; $0.2x + 0.2y + 0.5y = 6000$.

3. $$\begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & -1 \\ 0 & 10 & -10 & 0 & -15 & -5 \\ 5 & -10 & 0 & -20 & 0 & 0 \end{pmatrix} \begin{pmatrix} i_{12} \\ i_{52} \\ i_{32} \\ i_{65} \\ i_{54} \\ i_{43} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 150 \end{pmatrix}$$

4. $node(1,1): -4T_{11} + T_{21} + T_{12} = -150$; $node(2,1): T_{11} - 4T_{21} + T_{31} + T_{22} = -50$; $node(3,1): T_{21} - 4T_{31} + T_{32} = -125$; $node(1,2): T_{11} - 4T_{12} + T_{22} + T_{13} = -100$; $node(2,2): -4T_{22} + T_{32} + T_{23} + T_{12} + T_{21} = 0$; $node(3,2): -4T_{32} + T_{33} + T_{22} + T_{31} = -75$; $node(1,3): -4T_{13} + T_{23} + T_{12} = -100$; $node(2,3): -4T_{23} + T_{33} + T_{13} + T_{22} = 0$; $node(3,3): -4T_{33} + T_{23} + T_{32} = -75$.

12. (a) The Cholesky factor $H = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0.0316 & 0 \\ 1 & 0.0316 & 0.9995 \end{pmatrix}$

    (c) The growth factor $\rho = \dfrac{max(2,1,1)}{2} = 1$

14. (a) The leading principal minors obtained from the U matrix of the LU factorization of A using Gaussian elimination without pivoting are 4, 15, 56, 192. A is, therefore, positive definite. Obtain the Cholesky factorization of A using MATLAB command **chol**(A) and see that all the diagonal entries of the Cholesky factor are positive.

    (b) $x = (1 \quad 1 \quad 1 \quad 1)^T$

16. (a) For $k = 1$,

$$\sum_{\substack{i=2 \\ i \neq j}}^{n} |a_{ij}^{(1)}| = \sum \left| a_{ij} - \frac{a_{1j}a_{i1}}{a_{11}} \right| \leq \sum |a_{ij}| + \sum \left| \frac{a_{1j}a_{i1}}{a_{11}} \right|$$

$$< (|a_{jj}| - |a_{1j}|) + \frac{|a_{1j}|}{|a_{11}|}(|a_{11}| - |a_{j1}|) \qquad \text{(since } A \text{ is column-diagonally dominant)}$$

$$= \left| a_{jj} - \frac{a_{j1}a_{1j}}{a_{11}} \right| = |a_{jj}^{(1)}|$$

19. (a) (i) Let $H = (h_{ij})$ be $n \times n$ upper Hessenberg. For each value of k from 1 to $n-1$ perform the following steps:

    1. Interchange h_{kj} and $h_{k+1,j}$ if
    $$|h_{kk}| < |h_{k+1,k}|, \; j = k, \ldots, n.$$
    2. Compute $m = \dfrac{-h_{k+1,k}}{h_{k,k}}$.
    3. Update $h_{k+1,j} \equiv h_{k+1,j} + mh_{kj}$, $j = k+1, \ldots, n$.

28. (a) $Cond(A^{-1}) = \|A^{-1}\|\|A\| = Cond(A)$

    (b) $1 = \|I\| = \|A^{-1}A\| \leq \|A^{-1}\|\|A\| = Cond(A)$ (Assuming that $\|I\| = 1$). $Cond_2(A^TA) = \|A^TA\|_2\|(A^TA)^{-1}\|_2$. Now $\|A^TA\|_2 = \|A\|_2^2 = \sigma_1^2$, $\|(A^TA)^{-1}\|_2 = \dfrac{1}{\sigma_n^2}$, where σ_1 and σ_n are the largest and smallest singular values of A. Therefore, $Cond_2(A^TA) = \dfrac{\sigma_1^2}{\sigma_n^2} = (Cond_2(A))^2$. (See also Section 10.6.1).

29. (a) $Cond_2(O) = \|O\|_2\|(O)^{-1}\|_2 = \|O\|_2\|O^T\|_2 = 1 \times 1 = 1$.

    (b) $A = \alpha O$; $Cond_2(A) = Cond_2(O) = 1$. If $Cond_2(A)$ is 1, then $\|(A)^{-1}\|_2$ has to be reciprocal of $\|A\|_2$.

30. Since U is upper triangular, $\|U\|_\infty \geq max(u_{ii})$, and $\|U^{-1}\|_\infty \geq \dfrac{1}{min(u_{ii})}$. Take

$$U = \begin{pmatrix} 0.0001 & 2 & 3 \\ 0 & 1 & 12 \\ 0 & 0 & 12 \end{pmatrix}, \; Cond(U) = 4.9025 \times 10^5.$$

32. (a) See the proof of Theorem 10.6.1 in Chapter 10.

33. (b) The proof follows from Theorem 6.6.1 by noting that $\delta b = -r$.

    (c) If Cond(A) is small and r is small, then the relative error is small. Then the algorithm is weakly stable, by definition.

34. (c) $Cond(A) = 1.999 \times 10^3$.

36. (a) A is diagonally dominant and symmetric positive definite.

38. (a) Block Gauss–Seidel. $A_{ii}x_i^{(k+1)} = -\sum_{j<i} A_{ij}x_j^{(k+1)} - \sum_{j>i} A_{ij}x_j^{(k)} + b_i$, $i = 1, \ldots, N$, $k = 0, 1, \ldots$.

44. $\Phi(x - \alpha p)$

$$= \frac{1}{2}(x - \alpha p)^T A(x - \alpha p) - b^T(x - \alpha p)$$

$$= \frac{1}{2}x^TAx - \alpha p^TAx + \frac{1}{2}\alpha^2 p^TAp + \alpha p^Tb - b^Tx$$

(note that b^Tp is a scalar)

$$= \frac{1}{2}p^TAp\alpha^2 - p^T(Ax - b)\alpha + \frac{1}{2}x^T(Ax - 2b).$$

From $\Phi'(x - \alpha p) = 0$, and $p^TAp > 0$, we have the expression for α (see Ortega 1988a, 186).

Chapter 7

1. Let A have full rank. Then for any nonzero vector x, $Ax = y$ is nonzero. Thus, $x^TA^TAx = y^Ty > 0$.

2. Let $z \in R(A)$. Then there is a y such that $Ay = z$. $z^Tr = z^T(b - Ax) = z^Tb - z^TA(A^TA)^{-1}A^Tb = y^TA^Tb - y^TA^TA(A^TA)^{-1}A^Tb = 0$ (since $x = (A^TA)^{-1}A^Tb$).

3. Let $b - Ax \in N(A^T)$. Then $A^T(b - x) = 0$, that is, $A^TAx = A^Tb$.

5. $\|A\|_2^2 = \|A^TA\|_2$, $\|A^\dagger\|_2^2 = \|A^\dagger A^{\dagger T}\|_2 = \|(A^TA)^{-1}\|_2$. Thus, $Cond_2^2(A) = \|A\|_2^2\,\|A^\dagger\|_2^2 = Cond_2(A^TA)$.

17. (a) For $k = 1, 2, \ldots, n$ do

    For $i = 1, 2, \ldots, k-1$ do

    $r_{ik} = q_i^Ta_k$, $a_k \equiv a_k - r_{ik}q_i$

    $\hat{q}_k = a_k$, $r_{kk} = \|\hat{q}_k\|_2$

    $q_k = \frac{\hat{q}_k}{r_{kk}}$

20. The basic solution

$$x_B = P\begin{pmatrix} R_{11}^{-1}c \\ 0 \end{pmatrix}$$

The minimum-norm solution is computed by taking y_2 as:

$$\min_{y_2}\left\|\begin{pmatrix} R_{11}^{-1}R_{12} \\ -I_{n-r} \end{pmatrix} y_2 - \begin{pmatrix} R_{11}^{-1}c \\ 0 \end{pmatrix}\right\|_2$$

Therefore, the basic solution x_B cannot be minimum norm solution unless $R_{12} = 0$.

21. (a) $\|Ax - b\|^2 = \|QT'W^TP^Tx - b\|^2$

$$= \|T'W^TP^Tx - Q^Tb\|^2$$
$$= \|Ty_1 - c\|^2 + \|d\|^2,$$

where

$$T' = \begin{pmatrix} T & 0 \\ 0 & 0 \end{pmatrix}, \qquad Q^Tb = \begin{pmatrix} c \\ d \end{pmatrix}$$

$$W^TP^Tx = y = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$$

Therefore, all least-squares solutions are of the form

$$x = PW\begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$$

where y_1 is the unique solution of $T\ y_1 = c$. The minimum-norm solution is obtained by setting $y_2 = 0$.

29. Since $X = (A^TA)^{-1} = R^{-1}(R^{-1})^T$, the diagonal entries of X are just the 2-norms squared of the rows of R^{-1}: $x_{ii} = \sum_{j=i}^{n} s_{ij}^2$, $i = 1, 2, \ldots, n$; where $S = (s_{ij}) = R^{-1}$.

Chapter 8

5. By Theorem 8.4.1, $|\lambda - a_{ii}| \le r_i = \sum_{j\ne i}^{n} |a_{ij}|$. Since A is diagonally dominant, we have $|\lambda - a_{ii}| \le \sum_{j=1}^{n} |a_{ij}| < |a_{ii}|$. This means that $\lambda \ne 0$.
7. The eigenvalues of a symmetric matrix A are real. Theorem 8.4.1 therefore gives $-r_i \le \lambda - a_{ii} \le r_i$.
9. $(A - \sigma I)v_i = Av_i - \sigma v_i = \lambda_i v_i - \sigma v_i = (\lambda - \sigma)v_i$.
10. (a) $\lambda_2 = 2.9$ is close to $\lambda_i = 3$.
19. (e) Note that all s_j in this case are 1. The conclusion, therefore, follows from Theorem 8.8.1.
20. The eigenvalues are well conditioned, because A is symmetric. But when ϵ is small, the eigenvalues $1 + \epsilon$ and $1 - \epsilon$ are close to each other. By Theorem 8.8.1, the eigenvectors corresponding to these eigenvalues are then ill conditioned.

23. (a) $X = \begin{pmatrix} 1 & 1 \\ 0 & \epsilon \end{pmatrix}$

If ϵ is small, X is ill conditioned; then, by the Bauer–Fike Theorem, the eigenvalues of A are ill conditioned.

25. Take

$$Q = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 2 & 3 & 0 \end{pmatrix}, \qquad R = \begin{pmatrix} 0 & 1 & 3 \\ 0 & -1 & 2 \\ 0 & 0 & 0 \end{pmatrix}$$

Then $H = QR$ is Hessenberg, but Q is not. Since H is upper Hessenberg and $r_{kk} \ne 0$, from $H = QR$ or $h_{k\ell} = \sum_s q_{ks}r_{s\ell}$, we have $q_{k\ell} = 0$ for $k \ge \ell + 2$.

29. (a) The matrix Q is Hessenberg. The first column of QR is therefore $(q_{11}r_{11}, q_{21}r_{11}, 0, \ldots, 0)^T$. The first column of $H - \lambda I$ is $(h_{11} - \lambda, h_{21}, 0, \ldots, 0)^T$. Since they have to be the same, the first column of Q is $\frac{1}{r_{11}}$ times the first column of $H - \lambda I$.
    (c) Each of the Givens rotations in the problem has 1 in the (1,1) entry. Thus $\tilde{Q}$ has the same first column as P_0, and has the same first column as Q_1 by part (b).
33. It has been proved in Section 8.9.5 that H_{s+2} is similar to H_s. Thus, they have the same eigenvalues.
35. (a) Apply Theorem 8.4.1 to show that $|\lambda| < 4$. The Sturm sequence evaluated at $\mu = 2$ gives: $1, -1, -1, 1, 1$. There are two agreements and two variations of sign. Thus, there are two eigenvalues of A greater than 2 and less than 2.

Chapter 9

1. Let x be the common nonzero null vector of A and B. Then $(A - \lambda B)x = 0, \forall\lambda$. Since $x \ne 0$, $\det(A - \lambda B) = 0$ for any λ.
2. (b) Let $B = P^{-1}\Lambda P$, where Λ is the Jordan Canonical form of B. Then $p(\lambda) = \det(A - \lambda B) = \det(PAP^{-1} - \lambda\Lambda)$. Now B is nonsingular if and only if Λ is nonsingular, and nonsingularity of Λ implies that $p(\lambda)$ is of degree n.
3. Q_1 is a Householder matrix that transforms the first column n_1 of the matrix N to a multiple of e_1. n_1 has three nonzero entries x, y, z and it has been shown in Section 9.3.2 that these entries can be computed by inverting the 2×2 leading principal minor of B.
8. If x is an eigenvector of (A, B) corresponding to an eigenvalue λ, then $y = V^Tx$ is an eigenvector of $(\tilde{A}, \tilde{B})$ corresponding to λ.
9. (a) Find the Shur decomposition of B: $V^TBV = D$, and let $D = LL^T$.
    (b) Form $C = (V^TL)^{-1}A(L^TV)^{-1}$.
    (c) Compute the eigenvalues and eigenvectors y_i, $i = 1, \ldots, n$, of the symmetric matrix C, using QR iteration.
    (d) Compute the eigenvectors x_i of the pencil (A, B) by solving $L^TVx_i = y_i$, $i = 1, \ldots, n$.
13. Choose $Q_0 = I_{4\times4}$. Then at the end of the 50th iteration, we have the eigenvectors for the pair (A, B).

Math Sammat

$$Q = \begin{pmatrix} -0.0093 & -0.9730 & -0.2307 & 0.0000 \\ 0.3820 & -0.1948 & 0.8059 & 0.4082 \\ 0.5590 & -0.0384 & 0.1394 & -0.8165 \\ 0.7359 & 0.1180 & -0.5271 & 0.4082 \end{pmatrix}$$

26. Let λ be the eigenvalue of the pair (A, B), and x the associated eigenvector. Then $Ax = \lambda Bx$, i.e., $B^{-1}Ax = \lambda x$. Thus $|\lambda| = \frac{\|B^{-1}Ax\|}{\|x\|} \leq \|B^{-1}A\|$. Since the eigenvalues of the pair (A, B) are real, we have the result.

Chapter 10

3. Take $\tilde{U} = UP$, and $\tilde{V} = VP$, where P permutes row i and row $i+1$ of Σ.
6. (b) Let $A = U\Sigma V^T$. Then $A^TA = V\Sigma^T\Sigma V^T$, and $AA^T = U\Sigma\Sigma^T U^T$. Because $\Sigma^T\Sigma$ and $\Sigma\Sigma^T$ have the same number of nonzero eigenvalues, $\text{rank}(\Sigma^T\Sigma) = \text{rank}(\Sigma\Sigma^T)$, and hence A^TA and AA^T have the same rank. To prove *(iii)*, note that $(Au_1)^TAu_2 = u_1^TA^TAu_2 = \lambda u_1^Tu_2 = 0$.
7. (a) In the solution of Exercise 49 of Chapter 1, we showed that
$$\max_{x\neq 0} \frac{\|Ax\|_2}{\|x\|_2} \leq \sigma_1. \quad \text{Again, } \frac{\|Av_1\|_2}{\|v_1\|_2} = \sigma_1.$$
(b) See the solutions of Exercise 45 of Chapter 1.
8. Let $E = B - A$. Let $\sigma_i, i = 1, \ldots, n$ and $\tilde{\sigma}_i, i = 1, \ldots, n$ be, respectively, the singular values of A and B. By Theorem 10.5.1, $|\tilde{\sigma}_i - \sigma_i| \leq \|E\|_2 < \sigma_r$ for each i. Thus $\tilde{\sigma}_i$ cannot be zero for $i = 1, \ldots, r$.
15. (a) Let $v \in R(A)$. Then $\exists u$ such that $v = Au$. Using Property 1 of the pseudoinverse, we then have $AA^\dagger v = AA^\dagger Au = Au = v$.
(b) Let $x \in N(A^T)$. Then $A^\dagger x = A^\dagger AA^\dagger x = A^\dagger(AA^\dagger)^Tx = A^\dagger(A^\dagger)^TA^Tx = 0$.
16. (a) Let $X = (A^TA)^{-1}A^T$. Then X exists since A has full column rank. Since $AXA = A(A^TA)^{-1}A^TA = A, XAX = (A^TA)^{-1}A^TA(A^TA)^{-1}A^T = (A^TA)^{-1}A^T$, $(AX)^T = X^TA^T = A(A^TA)^{-1}A^T = AX, (XA)^T = A^TX^T = A^TA(A^TA)^{-1} = I$, and $XA = (A^TA)^{-1}A^TA = I$, we have $A^\dagger = (A^TA)^{-1}A^T$.
21. Without loss of generality, assume that B is 2×2. Assume further that none of the diagonal entries is zero.
$$\text{Let} \quad B = \begin{pmatrix} \alpha_1 & \beta_1 \\ 0 & \alpha_2 \end{pmatrix}$$
Then
$$B^TB = \begin{pmatrix} \alpha_1^2 & \alpha_1\beta_1 \\ \beta_1\alpha_1 & \beta_1^2 + \alpha_2^2 \end{pmatrix}$$
The characteristic polynomial $p(\lambda)$ of B^TB is $\lambda^2 - (\alpha_1^2 + \alpha_2^2 + \beta_1^2)\lambda + \alpha_1^2\alpha_2^2$. Thus B has a multiple singular value if and only if $p(\lambda)$ has a multiple zero. Since α_1 and α_2 are not zero, $p(\lambda)$ can have a multiple zero only if $\beta_1 = 0$.

Chapter 11

1. (a) $L = \begin{pmatrix} 1 & 0 \\ 1.67 & 1 \end{pmatrix}$ $\quad U = \begin{pmatrix} 3 & 4 \\ 0 & -0.68 \end{pmatrix}$ $\quad E = \begin{pmatrix} 0 & 0 \\ 0.01 & 0 \end{pmatrix}$

(b) $L = \begin{pmatrix} 1 & 0 \\ 0.04 & 1 \end{pmatrix}$ $\quad U = \begin{pmatrix} 0.25 & 0.79 \\ 0 & 0.09 \end{pmatrix}$ $\quad E = \begin{pmatrix} 0 & 0 \\ 0 & 0.0016 \end{pmatrix}$

(c) $L = \begin{pmatrix} 1 & 0 \\ 0.8 & 1 \end{pmatrix}$ $\quad U = \begin{pmatrix} 10 & 9 \\ 0 & -2.2 \end{pmatrix}$ $\quad E = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$

(d) $L = \begin{pmatrix} 1 & 0 \\ 0.25 & 1 \end{pmatrix}$ $\quad U = \begin{pmatrix} 4 & 1 \\ 0 & 1.75 \end{pmatrix}$ $\quad E = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$

(e) $L = \begin{pmatrix} 1 & 0 \\ 3 & 1 \end{pmatrix}$ $\quad U = \begin{pmatrix} 0.01 & 0.05 \\ 0 & -0.14 \end{pmatrix}$ $\quad E = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$

2. (a) $P = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$, $\quad L = \begin{pmatrix} 1 & 0 \\ 0.6 & 1 \end{pmatrix}$, $\quad U = \begin{pmatrix} 5 & 6 \\ 0 & 0.4 \end{pmatrix}$

$E = LU - PA = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$

(e) $P = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$, $\quad L = \begin{pmatrix} 1 & 0 \\ 0.33 & 1 \end{pmatrix}$, $\quad U = \begin{pmatrix} 0.03 & 0.01 \\ 0 & 0.05 \end{pmatrix}$

$E = LU - PA = \begin{pmatrix} 0 & 0 \\ 0.0001 & 0.0033 \end{pmatrix}$

TABLE OF SYMBOLS

$\mathbb{R}^n$	real vector space of dimension n
$\mathbb{R}^{m\times n}$	the set of $m \times n$ real matrices
$\mathbb{C}^{m\times n}$	the set of $m \times n$ complex matrices
e_i	ith unit vector
$\mathbf{0}$	a zero matrix or a zero vector
$I_{n\times n}$, I	identity matrix of order n
I_{k-1}	the matrix of the first $(k-1)$ rows and columns of I
$R(A)$	range (or the column space) of A
$N(A)$	null space of A
$\text{null}(A)$	nullity of A
A^T	transpose of A
A^{-1}	inverse of A
A^k	kth power of A
$\det(A)$	determinant of A
$\text{rank}(A)$	rank of A
$\rho(A)$	spectral radius of A
ρ	growth factor
$\bar{A}$	complex conjugate of A
A^*	transpose of the complex conjugate of A
$fl(x)$	floating point number x
$\text{cond}(A)$, $\text{Cond}(A)$	condition member of A
$A\dagger$	pseudoinverse of A
$:=$	symbol for overwriting
$\leftrightarrow$	symbol for interchanging
$\kappa(A, b)$	Krylov subspace span $\{b, Ab, \ldots, A^{n-1}b\}$
$\dim(N)$	dimension of N

RSF	real Schur form
SVD	singular value decomposition
P_A	orthogonal projection onto $R(A)$
$P_A^{\perp}, P_N$	projection onto the orthogonal compliment of $R(A)$
b_R	projection of the vector b onto $R(A)$
$b_{\perp}, b_N$	projection of the vector b onto the orthogonal complement of $R(A)$
$\|x\|$	norm of the vector x
$\|A\|$	norm of the matrix A
$\|A\|_p$	p-norm of A, $p = 1, 2, \infty$
$\|A\|_F$	Frobenius norm of A
$\sum$	summation sign
$\prod$	product sign
δb	amount of perturbation in the vector b
ΔA	amount of perturbation in the matrix A
E	error matrix
CGS	classical Gram–Schmidt process
MGS	modified Gram–Schmidt process
$\dot{x}$	first derivative of x
$\ddot{x}$	second derivative of x
O	symbol for order
$\mathrm{diag}(d_1, d_2, \ldots, d_n)$	diagonal matrix
$\mathrm{diag}(A_{11}, \ldots, A_{kk})$	block diagonal matrix
ζ_i	model damping ratio of the ith mode
σ_i	ith largest singular value of A
μ	the machine precision

LIST OF ALGORITHMS

Arnoldi method	289
Back substitution	99
Bidiagonalization	578
Bisection method for eigenvalues in an interval	461
Block LU factorization	233
Chan SVD	583–584
Cholesky algorithm	223
Cholesky algorithm for positive definite system	224
Cholesky QR algorithm for the symmetric definite pencil	508
Condition number estimation	257–260
Conjugate gradient method	284
Demmel–Kahan algorithm for SVD	585–586
Determinant of a matrix	242–243
Divide-and-conquer method	467
Eigenvector computation from real Schur form	453–454
Forward elimination	208
Gaussian elimination with complete pivoting	129–130
Gaussian elimination with partial pivoting for linear system	213–214
Gaussian elimination with partial pivoting (triangularization)	126
Gaussian elimination without pivoting for linear systems	73, 100
Gaussian elimination without pivoting (triangularization)	118
Gauss–Seidel iterative method	270–271
Givens Hessenberg reduction	164
Givens QR factorization	160
GMRS (Generalized Minimial Residual) method	291
Golub–Kahan–Reinsch algorithm for SVD	577–581
Gram–Schmidt process for QR factorization	338–339
Householder deflation for the subdominant eigenvalue	419

Householder QR factorization 142–143
Householder QR factorization with column pivoting 170
Householder QR updating 173
Incomplete Cholesky factorization 287
Inner product computation 68
Inverse iteration 411, 452
Inverse iteration for eigenvectors of the pencil (A, B) 505
Inverse of a matrix 241
Inverse of an upper triangular matrix 70, 99
Iterative refinement for least-squares solution 353–354
Iterative refinement for linear system solution 262
Jacobi iterative method 268
Lanczos algorithm for symmetric definite pencil 536
Lanczos method for eigenvalues of (A, B) in an interval 538–539
Lanczos method for symmetric matrices 469
Least-squares by modified Gram–Schmidt 342
Least-squares by QR factorization (full rank) 336
Least-squares by QR factorization (rank-deficient) 344
Least-squares using normal equations 330
Least-squares using SVD 348, 572
Linear system solution using QR factorization 215
Linear system solution with multiple right hand side 217
Modified Gram–Schmidt process for QR factorization 339
Power method 408
Preconditioned conjugate gradient method 286
Product of a matrix by a Givens matrix 155
Product of a matrix by a Householder matrix 97
Product of a vector by a Householder matrix 136
Pseudoinverse using SVD 575
QR iteration for symmetric matrix 463
QR iteration (implicit double-shift) 446
QR iteration (single-shift) 439
QZ algorithm 503–504
Rayleigh quotient for the pair (A, B) 511–512
Reyleigh quotient iteration 416
Schur QR algorithm for symmetric definite pencil 508–509
Simultaneous diagonalization technique 510
Sturm sequence method for tridiagonal (A, B) 534
Successive overrelaxation method 278
Underdetermined system solution by normal equations 350
Underdetermined system solution by QR 352
Updating a QR factorization 173

INDEX

Absolute error, 46
Accumulation of inner product in double precision, 55
  in back substitution, 76
  in Householder Hessenberg reduction, 152
  in iterative refinement for least-squares solution, 354
Algorithm, 67
  properties of numerical effectiveness of, 92
Amplitudes, 513
A-norm, 285
Angle between vectors, 8
Applications
  of generalized eigenvalue problems in vibration, 512–532
  giving rise to eigenvalue problems, 381–400
  giving rise to least-squares problem, 316, 319–321
  giving rise to linear system problem, 190–207
  of the singular value decomposition problem, 558–560, 561–575
Arnoldi method, 289

Back substitution, 69
Backward error analysis, 75, 600–611
Backward stability, 75, 85, 86, 361. *See also* Stability
Balancing, 449
Banach lemma, 41
Band matrix, 105
Bartels-Stewart method, 491
Basis, 18
  Orthonormal, 166, 563
Bauer-Fike theorem, 425
Bisection method, 461
BLAS, 101
Block algorithm, 101
Block diagonalization, 491
Block (matrix), 11
  diagonal, 11
  triangular, 12
  tridiagonal, 202, 203, 233, 234
Block cyclic reduction, 235
Buckling problem, 387, 391–393

Catastrophic cancellation, 57
Cauchy-Schwarz inequality, 25, 40
Cayley-Hamilton theorem, 23
Chan SVD, 583–584
Characteristic polynomial, 12
  and eigenvalue computation, 422
Cholesky algorithm, 223
  backward stability of, 223
  for solving least-squares problem, 330
  for solving linear system, 224
  incomplete, 287
Cholesky factorization theorem, 218
Cholesky QR algorithm, 507–508
Companion matrix, 20
  reduction to, 423
Complete orthogonal factorization, 171
Componentwise perturbations, 260–261
Condition number, 79
  and accuracy of linear system solution, 252–253

Condition number (*continued*)
of an eigenvalue, 428
estimation, 257–260
of a linear system, 245
of a matrix, 245
and nearness to singularity, 254
and pivoting, 254–255
of a problem, 80
properties of, 250, 251
and scaling, 256
and singular values, 562
size for ill-conditioning, 253
Conjugate gradient method, 284
convergence of, 285
preconditioned, 286
and relationship of GMRES, 292
Convergence
comparison between Gauss-Siedel and Jacobi methods, 276
of conjugate gradient method, 285
of Demmel-Kahan SVD alogorithm, 587
of Golub-Kahan-Reinsch SVD algorithm, 582
of iterative methods, 272
of Jacobi and Gauss-Seidel methods, 273–275
of power method, 410–411
of QR iteration, 438, 439, 463
of Rayleigh quotient iteration, 417
of SOR iteration, 279
Convergent matrix, 30
Courant-Fischer Minimax theorem, 456
Cramer's Rule, 187
Crout's Reduction, 178

Damping, 389
proportional, 523, 524
Defective matrix, 12
Deflation, 418, 448, 582
Hotelling, 418
Householder, 418–420
Demmel-Kahan SVD algorithm, 585–588
Determinant, 11
computation of, 241–243
properties of, 12
Diagonal matrix, 17
Diagonal pivoting factorization, 225, 405
Diagonally dominant matrix, 21
Diagonally dominant system, 228–229
Divide-and-conquer method, 464–468
Doolittle reduction, 178

Eigenvalues, 12
algebraic multiplicity of, 381
basic results on, 373–381
bounds for, 403, 405
computation of the smallest, 420–421
computation using QR iteration, 435–455
computation via characteristic polynomial, 422–424
computation via Jordan Canonical form, 425
geometric multiplicity of, 381
sensitivity of, 425–429
Eigenvectors, 12, 374
basic results on, 374–381
computation of, 411, 452–455
left, 374
right, 12, 374
sensitivity of, 431
Electric circuit, 190
simulating transient current of, 393–398
Elementary lower triangular matrix, 111
EISPACK, 100, 108
Error, 46

Fernando-Parlett SVD algorithm, 588
Finite difference scheme, 196–203
Finite element method, 203–206
Floating-point arithmetic, 43
addition in, 51
error bounds for basic matrix computations in, 55
fundamental laws of, 48
IEEE standard for, 43
inner product computation in, 53
multiplication in, 52
Flop, 92–93
Flop-count
for basic matrix algorithms, 94–100
for Cholesky algorithm, 225
for Gaussian elimination, 118
for Givens QR factorization, 160
for Givens Hessenberg reduction, 164
for Hessenberg system using Gausian elimination, 226
for Householder Hessenberg reduction, 152
for Householder QR factorization, 144
for least-squares using Householder QR, 336
for least-squares using normal equations, 331
for least-squares using SVD, 585
for LU factorization of a tridiagonal matrix, 230
for modified Gram-Schmidt process, 339
for product of matrix by Givens matrix, 155
for product of matrix by Householder matrix, 97
for QR iteration (nonsymmetric), 447, 450

for QZ algorithm, 504
for simultaneous diagonalization technique, 509–510
for SVD algorithms, 583
for underdetermined system solution using Householder QR, 352, 361
for underdetermined system solution using normal equations, 361
Forward elimination, 208
Frequencies, 388, 506

Gaussian elimination, 71
stability of, 131
with complete pivoting, 127–130, 211
with partial pivoting, 120, 209
without pivoting, 73, 112, 208
Gauss–Seidel method, 269–277
Gauss Jordan reduction, 178
Generalized eigenvalue problem 2, 494
computational methods for, 497–511, 534–539
Lanczos methods for, 535–539
QZ algorithm for, 497–504
Schur decomposition for, 486–497
Sturm sequence method for, 534–535
symmetric definite, 496, 506–511
Geršgorin disk theorems, 400–403
Givens rotations
creating zeros in a vector using, 155–159
QR factorization using, 159–161
QR iteration and, 437
reduction to Hessenberg form using, 163–165
GMRES, 290–292
Golub-Kahan-Reinsch SVD algorithm, 577–583
Gram-Schmidt process, 338
Growth factor, 77, 131
for a diagonally dominant system, 229
of Gaussian elimination, 132, 133
for Hessenberg system, 226
for positive definite system, 222
for tridiagonal system, 231

Hankel matrix, 107
Hessenberg (matrix), 19
and QR iteration, 437
reduction using Givens matrices, 163–165
reduction using Householder matrices, 146–151
uniqueness in Hessenberg reduction, 165
unreduced Hessenberg matrix, 19
Hessenberg system, 226
Hilbert matrix, 83, 107, 207, 252
Hoffman-Wielandt theorem, 458
Householder-Golub method for the least
Householder Hessenberg reduction, 146–151
Householder matrix, 134–141
Householder QR factorization, 138–145

Ill-conditioned eigenvalue problem, 83–85
Ill-conditioned problem, 79
Ill-conditioned system, 83, 251
Implicit Q Theorem, 165
IMSL, 103, 108
Incomplete Cholesky factorization, 287
Indefinite systems, 225
Inertia of a matrix, 405
Interlancing property, 459
Invariant subspace, 450
Inverse of a matrix, 13
computation of, 240–241
properties of, 14
Inverse iteration/Inverse power method, 411–414
eigenvectors of the pencil (A, B), 505
Hessenberg, 451–452
Isometry lemma, 40
ITPACK, 103
Iterative refinement
and condition estimation, 264
for solution of linear systems, 262–264
for solutions of least-squares problem, 353–356
Iterative methods, 264–290
convergence of, 272
stopping criteria for, 266

Jacobi iterative method, 266–269
Jordan canonical form, 30, 380–381

Lanczos algorithm
for eigenvalues of the definite pencil (A, B) in an interval, 538–539
loss of orthogonality in, 471
for symmetric definite pencil (A, B), 535–537
for symmetric matrices, 468–472
LAPACK, 101, 108, 633–634
Least-squares problem, 1, 315
existence and uniqueness, 317–319
geometric interpretation, 319
iterative refinement, 353–356
sensitivity, 324–330
solution by Gram-Schmidt, 338
solution by modified Gram-Schmidt, 341–343, 353
solution using normal Equations, 330–332, 349–351
solution using pseudoinverse, 323
solution using QR factorization, 335–347, 351–353

Linear system problem, 1
direct methods for solution of, 207–237
existence and uniqueness of solution, 189
iterative methods for solution of, 263–292
iterative refinement for solution of, 260–263
perturbation analysis for, 243–250
LINPACK, 100, 108
LU factorization, 111, 114, 115, 123, 128
and elementary matrices, 111–130
existence and uniqueness of, 115
Lyapunov equation, 491

Machine epsilon (precision), 48
Mass matrix, 388, 511, 521, 526, 527
Matrix, 9
band, 105
bidiagonal, 577, 578, 580
block, 10
block diagonal, 11
companion, 21
complex, 374
consistently ordered, 280
cyclic, 280
defective, 12
diagonal, 17
diagonally dominant, 21
elementary lower triangular, 174
exponential, 382
Hermitian, 374
Hessenberg, 19
Hilbert, 106
identity, 9
idempotent, 15
ill conditioned, 251
invertible, 13
lower triangular, 17
nonderogatory, 21
nonsingular, 13, 14
normal, 430
orthogonal, 17
permutation, 18
Pie, 107, 252
positive definite, 21, 22
Property A, 280
symmetric, 10
Toeplitz, 86
tridiagonal, 19, 230
unitary, 374
upper triangular, 17
well-conditioned, 251
Wilkinson bidiagonal 84, 107
MATCOM, 103, 622–632
MATLAB, 103, 108, 613–621
Minimum norm solution, 315, 345, 349, 350, 352, 353
Modal matrix, 511
Modes, 388–389, 506
Modal damping ratio, 523, 526, 532
Mode participation factor, 526
Normal modes, 511
Modified Gram-Schmidt process, 339
for least-squares solution, 342
versus Classical Gram-Schmidt process, 340–341
Moore-Penrose generalized inverse, 323

NAG, 102, 108
Natural frequencies, 388, 506
NETLIB, 102
Nonderogatory matrix, 21
Nonsingular matrix, 13, 14
Norm, 24
Frobenius, 27
matrix, 26
spectral, 27
subordinate, 26
vector, 24
triangle inequality, 24
Normalized floating-point number, 43
Normal equations, 319
difficulties with, 329
and least-squares solutions. *See* Least-squares problem, solutions
and polynomial fitting, 319–322
Null space, 12, 16
computation of orthonormal basis of, 166, 563
computation of projections onto, 166, 563
Numerical rank of a matrix, 569

Orthogonal matrix, 17
Orthogonal projection, 15, 16, 324–327
compuation of, 166, 563
properties of, 34–35
Orthonormal basis, 14, 166, 563
Overdetermined systems, 315–316
methods for, 330–353
Overflow, 45

Permutation matrices, 18
properties of, 19
Perturbation analysis, 78
of eigenvalue problem, 425–431
of least-squares problem, 324–330
of linear system problem, 243–252
of singular values, 560, 578
Pie matrix, 107, 252
Positive definite matrix, 21, 22

Power method, 407–411
with shift, 410–411
Principal component analysis, 398–400
Pseudocodes, 67
Pseudoinverse, 323, 570, 574, 575

QMR method, 292
QR factorization, 138
with column pivoting, 168–170
modifying, 172–173
and orthonormal basis, 166
and orthogonal projections, 166–168
uniqueness in, 161–162
using Givens matrices, 159–161
using Gram-Schmidt process, 338
using Householder matrices, 138–145
using modified Gram-Schmidt process, 339–340
QR iteration, 435
double-shift (complex), 441–443
Hessenberg, 437
implicit, 444–447
single-shift, 439
symmetric, 463–464
Quadratic eigenvalue problem, 3, 531–533
QZ algorithm, 497–504

Range of a matrix, 12, 324
computation of orthonormal basis for, 166, 563
computation of projections onto, 166, 563
Rank revealing QR factorization, 172
Rank, 13, 569
Rayleigh damping, 523
Rayleigh quotient, 415
iteration for eigenvalue problem, 416–417
iteration for generalized eigenvalue problem, 511
Real Schur form, 434
computation of, 447–449
generalized, 498–504
Resonance, 388–389
Ritz value, 412, 536
Ritz vector, 412, 536
Round off errors, 46
for back substitution, 76, 77, 603
for Cholesky factorization, 223
for Demmel-Kahan SVD algorithm, 587
for forward elimination, 601–602
for Givens Hessenberg reduction, 164
for Givens QR factorization, 160
for Golub-Kahan-Reinsch SVD algorithm, 583
for Householder Hessenberg reduction, 152
for Householder QR factorization, 145
for least-squares iterative refinement algorithm, 355
for least-squares solutions using QR factorization, 336, 337, 345, 346
for linear system solution using Gaussian elimination, 77, 610
for LU factorization using Gaussian elimination, 606
for QZ iteration algorithm, 504
for symmetric Lanczos algorithm, 471
for symmetric QR iteration, 464
for underdetermined system using QR factorization, 352
for zeroing in a vector using Householder matrix, 136

Scaling, 235
and conditioning, 256
Schur QR algorithm, 508–509
Schur triangularization theorem, 377, 433
Sensitivity of eigenvalues, 427–430
Sensitivity of eigenvectors, 431
Sensitivity of linear systems, 243–252
Sensitivity of singular values, 560
Sensitivity of singular values of a bidiagonal matrix, 578
Sherman-Morrison formula, 239
Significant digits, 47
Similar matrices, 14
Similarity transformation and eigenvalue computation, 422
Simultaneous diagonalization, 509–511
applications of, 520–531
Singular values, 16–17
Singular value decomposition (SVD), 24
and approximation of a matrix, 564–568
computation of, 575–589
distance of a matrix from the nearest matrix of lower rank, 567
and eigendecomposition, 555–556
existence theorem for, 552–554
generalized, 589
history of, 551
and least-squares problem, 571–572
and norm and condition number, 562
and numerical rank, 569
and orthonormal basis and projections, 562–563
and outer-product expansion of a matrix, 568
and pseudoinverse, 574–575
Skeel's condition number, 261
SPARSKIT, 104
Special linear systems, 218–235
Spectral radius, 31

Spectrum of a matrix, 375
Stability (backward), 75
of arithmetic operations, 76
of back substitution process, 77
of Cholesky algorithm for symmetric positive definite system, 225
of Cholesky factorization, 223
of Cholesky QR algorithm for definite pencil, 508
of Gaussian elimination for a diagonally dominant system, 229
of Gaussian elimination for a Hessenberg system, 226
of Gaussian elimination for a positive definite system, 222
of Gaussian elimination for a tridiagonal system, 231
of Gaussian elimination with complete pivoting, 132
of Gaussian elimination with partial pivoting, 132
of Gaussian elimination without pivoting, 78, 133, 213
of Givens Hessenberg reduction, 164
of Givens QR factorization, 160–161
of Householder Hessenberg reduction, 152
of Householder QR factorization, 135
of inverse iteration, 412
of least-squares algorithms. *See* Table of comparisons
of QR iteration (implicit double shift), 450
of QR iteration (symmetric), 464
of QZ algorithm, 504
of the SVD algorithms, 583 , 587–588
Stability of differential and difference equations, 381–386
Stiffness matrix, 388, 511, 521, 526, 527
Sturm sequence method, 458–462
for tridiagonal A and B, 534, 535
Subspace, 8, 9
Successive overrelaxation method, 277–283
Sylvester equation, 490
Sylvester law of inertia, 404
Symmetric matrix, 10

Table of Comparisons
for least-squares algorithms, 361
for linear systems algorithms, 237
for matrix factorization algorithms, 176
TEMPLATES, 104, 109
Toeplitz matrix, 86, 107
Trace, 27, 29, 40, 374
Triangular matrix, 17
Tridiagonal matrix, 19
Tridiagonal reduction, 152
Tridiagonal system, 199, 230–231

Underdetermined system, 348–353
Underflow, 45

Vandermonde matrix, 107
Variance-covariance matrix, 357, 570
Vector, 7, 8, 24
Vector space, 7
Vibration, 387
forced harmonic, 517
of a building, 389, 514–517
of a building during earthquake, 528–530
of a spring-mass system, 512–513
period of, 388

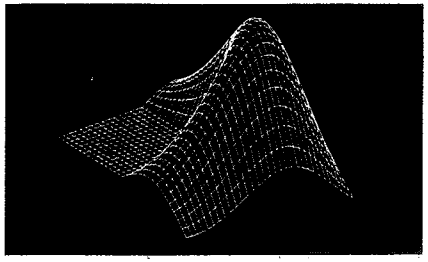
Well-conditioned problem, 79
Well-conditioned system, 251
Wilkinson bidiagonal matrix, 84, 107, 252
Wilkinson polynomial, 81
Wilkinson shift, 463, 484
Woodbury formula, 240

2 easy ways to receive the free *MATCOM Program* and more information on MATLAB:

1. Fax the card below to (508) 653-6284

2. Mail the card below to:

The MathWorks, Inc.
24 Prime Park Way
Natick, MA 01760



NAME
TITLE
COMPANY
DEPT. OR M/S
STREET
CITY/STATE/ZIP
PHONE
FAX EMAIL
WHERE DID YOU PURCHASE THIS BOOK?

Computer platform – check all that apply:
☐ PC/Macintosh ☐ UNIX Workstation ☐ VAX/Supercomputer

▶ **Call us with any questions at (508) 653-1415, or send email to info@mathworks.com.**

MATLAB® Technical Computing Environment

R-BK-DTA

MATLAB is used in **Numerical Linear Algebra and Applications** by Biswa Nath Datta (Brooks/Cole Publishing, 1995) to present theory and to solve problems. The *MATCOM Program*, consisting of MATLAB M-files, has been developed by the author to illustrate the concepts presented in the text.

- **MATLAB Application Toolboxes** add functions for symbolic math, signal processing, control design, neural networks, and other areas.
- **The Student Edition** is a limited-matrix-size version of MATLAB for use on students' own personal computers.
- **Educational discount plans** support classroom instruction and research.
- **Classroom Kits** provide cost-effective support for PC or Mac teaching labs.
- **MATLAB-based books** use MATLAB to illustrate basic and advanced material in a wide range of topics.

I am interested in The MathWorks product information for:

☐ Simulation ☐ Control System Design ☐ Math & Visualization
☐ Signal Processing ☐ Symbolic Math ☐ Educational Discounts
☐ System Identification ☐ Chemometrics ☐ Classroom Kits
☐ Neural Networks ☐ Optimization ☐ Student Edition
☐ Statistics ☐ Image Processing ☐ MATLAB Books

Send me the free *MATCOM Program* disk for my:
(check one) ☐ PC ☐ Macintosh