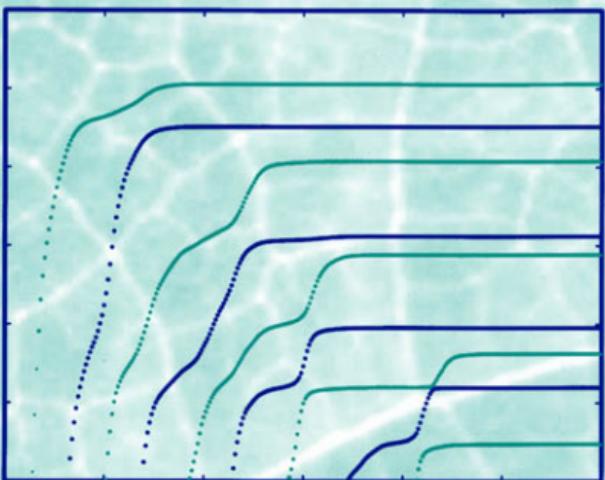


Templates for the Solution of Algebraic Eigenvalue Problems

A Practical Guide



Edited by

Zhaojun Bai
James Demmel
Jack Dongarra
Axel Ruhe
Henk van der Vorst

siam

SOFTWARE • ENVIRONMENTS • TOOLS

Templates for the Solution of Algebraic Eigenvalue Problems

Software • Environments • Tools

The series includes handbooks and software guides as well as monographs on practical implementation of computational methods, environments, and tools.

The focus is on making recent developments available in a practical format to researchers and other users of these methods and tools.

Editor-in-Chief

Jack J. Dongarra

University of Tennessee and Oak Ridge National Laboratory

Editorial Board

James W. Demmel, University of California, Berkeley

Dennis Gannon, Indiana University

Eric Grosse, AT&T Bell Laboratories

Ken Kennedy, Rice University

Jorge J. Moré, Argonne National Laboratory

Software, Environments, and Tools

Zhaojun Bai, James Demmel, Jack Dongarra, Axel Ruhe, and Henk van der Vorst,

Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide

Lloyd N. Trefethen, *Spectral Methods in MATLAB*

E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz,

A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide, Third Edition*

Michael W. Berry and Murray Browne, *Understanding Search Engines: Mathematical Modeling and Text Retrieval*

Jack J. Dongarra, Iain S. Duff, Danny C. Sorensen, and Henk A. van der Vorst,

Numerical Linear Algebra for High-Performance Computers

R. B. Lehoucq, D. C. Sorensen, and C. Yang, *ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*

Randolph E. Bank, *PLTMG: A Software Package for Solving Elliptic Partial Differential Equations, Users' Guide 8.0*

L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra,

S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley,

ScaLAPACK Users' Guide

Greg Astfalk, editor, *Applications on Advanced Architecture Computers*

Françoise Chaitin-Chatelin and Valérie Frayssé, *Lectures on Finite Precision Computations*

Roger W. Hockney, *The Science of Computer Benchmarking*

Richard Barrett, Michael Berry, Tony F. Chan, James Demmel, June Donato, Jack Dongarra,

Victor Eijkhout, Roldan Pozo, Charles Romine, and Henk van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*

E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum,

S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen, *LAPACK Users' Guide, Second Edition*

Jack J. Dongarra, Iain S. Duff, Danny C. Sorensen, and Henk van der Vorst,

Solving Linear Systems on Vector and Shared Memory Computers

J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W. Stewart, *Linpack Users' Guide*

Templates for the Solution of Algebraic Eigenvalue Problems

A Practical Guide

Edited by

Zhaojun Bai
University of California
Davis, California

James Demmel
University of California
Berkeley, California

Jack Dongarra
University of Tennessee
Knoxville, Tennessee

Axel Ruhe
Chalmers Tekniska Högskola
Göteborg, Sweden

Henk van der Vorst
Utrecht University
Utrecht, The Netherlands



Society for Industrial and Applied Mathematics
Philadelphia

© 2000 by the Society for Industrial and Applied Mathematics.

10 9 8 7 6 5 4 3 2 1

All rights reserved. Printed in the United States of America. No part of this book may be reproduced, stored, or transmitted in any manner without the written permission of the publisher. For information, write to the Society for Industrial and Applied Mathematics, 3600 University City Science Center, Philadelphia, PA 19104-2688.

Library of Congress Cataloging-in-Publication Data

Templates for the solution of algebraic eigenvalue problems / edited by Zhaojun Bai... [et al.]

p. cm. -- (Software, environments, tools)

Includes bibliographical references and index.

ISBN 0-89871-471-0 (pbk. : alk. paper)

1. Eigenvalues—Data processing. I. Bai, Zhaojun. II. Series.

QA193.T46 2000

512.9'434-dc21

00-059507



Royalties from the sale of this book are placed in a fund to help students attend SIAM meetings and other SIAM-related activities. This fund is administered by SIAM and qualified individuals are encouraged to write directly to SIAM for guidelines.

This book is also available in html form on the Internet. To view the html file use the following URL: <http://www.netlib.org/etemplates/>



is a registered trademark.

List of Contributors

Bai, Zhaojun, Department of Computer Science, University of California, One Shield Avenue, Davis, CA 95616. bai@cs.ucdavis.edu

Chen, Tzu-Yi, Computer Science Division, Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA 94720. tzuyi@cs.berkeley.edu

Day, David, Sandia National Laboratories, P.O. Box 5800, Albuquerque, NM 87185. dday@cs.sandia.gov

Demmel, James W., Department of Mathematics and Computer Science Division, Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA 94720. demmel@cs.berkeley.edu

Dongarra, Jack J., Department of Computer Science, University of Tennessee, Knoxville, TN 37996. dongarra@cs.utk.edu

Edelman, Alan, Department of Mathematics, Massachusetts Institute of Technology, Cambridge, MA 02139. edelman@math.mit.edu

Ericsson, Thomas, Chalmers Tekniska Högskola, Matematiska Institutionen, S-412 96 Göteborg, Sweden. thomas@math.chalmers.se

Freund, Roland W., Computing Sciences Research Center, Bell Laboratories, Lucent Technologies, Murray Hill, NJ 07974. freund@research.bell-labs.com

Gu, Ming, Department of Mathematics, University of California, Los Angeles, CA 90055. mgu@math.ucla.edu

Kågström, Bo, Department of Computing Science, Umeå University, S-901 87 Umeå, Sweden. bokg@cs.umu.se

Knyazev, Andrew, Department of Mathematics, University of Colorado at Denver, Denver, CO 80217. andrew.knyazev@cudenver.edu

Koev, Plamen, Department of Mathematics, University of California, Berkeley, CA 94720. plamen@math.berkeley.edu

Kowalski, Thomas, MSC Software Corporation, 2975 Redhill Avenue, Costa Mesa, CA 92626. tom.kowalski@mscsoftware.com

Lehoucq, Richard B., Sandia National Laboratories, P.O. Box 5800, Albuquerque, NM 87185. rlehoucq@sandia.gov

Li, Ren-Cang, Department of Mathematics, University of Kentucky, Lexington, KY 40506. rccli@ms.uky.edu

Li, Xiaoye S., National Energy Research Scientific Computing Center, Lawrence Berkeley National Laboratory, MS 50F, One Cyclotron Rd, Berkeley, CA 94720. xiaoye@nersc.gov

Lippert, Ross, Sandia National Laboratories, P.O. Box 5800, Albuquerque, NM 87185. ripper@cs.sandia.gov

Maschhoff, Kristi, Cray Inc., Merrill Place, 411 First Ave. South, Ste. 600, Seattle, WA 98104-2860. kristyn@cray.com

Meerbergen, Karl, Free Field Technologies, 5 rue Charlemagne, 1348 Louvain-la-Neuve, Belgium. Karl.Meerbergen@fft.be

Morgan, Ronald, Department of Mathematics, Baylor University, Waco, TX 76798. Ronald_Morgan@baylor.edu

Ruhe, Axel, Chalmers Tekniska Högskola, Matematiska Institutionen, S-412 96 Göteborg, Sweden. ruhe@math.chalmers.se

Saad, Yousef, Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN 55455. saad@cs.umn.edu

Sleijpen, Gerard L.G., Mathematical Institute, Utrecht University, Mailbox 80.010, 3508 TA Utrecht, the Netherlands. sleijpen@math.uu.nl

Sorensen, Danny C., Department of Computational and Applied Mathematics, Rice University, Houston, TX 77005. sorensen@caam.rice.edu

van der Vorst, Henk A., Mathematical Institute, Utrecht University, Mailbox 80.010 3508 TA Utrecht, the Netherlands. vorst@math.uu.nl

How to Use This Book

This book is a guide to the numerical solution of eigenvalue problems. It attempts to present the many available methods in an organized fashion, to make it easier for you to identify the most promising methods.

Chapter 1 gives the motivation for this book and the use of templates.

Chapter 2 provides the top level of a *decision tree* for classifying eigenvalue problems, and their corresponding solution methods, into six categories. It is important to read enough of this chapter to make sure you pick the most efficient and accurate method for your problem.

Chapter 3 summarizes the two mathematical principles used by most algorithms for large eigenvalue problems: projection onto subspaces and spectral transformations.

Chapters 4 through 9 give details for each of the six categories of eigenvalue problems identified in Chapter 2, including algorithm templates and pointers to available software. These chapters form the core of the book.

Chapter 4: Hermitian eigenvalue problems.

Chapter 5: Generalized Hermitian eigenvalue problems.

Chapter 6: Singular value decomposition.

Chapter 7: Non-Hermitian eigenvalue problems.

Chapter 8: Generalized non-Hermitian eigenvalue problems.

Chapter 9: Nonlinear eigenvalue problems.

Chapter 10 describes common issues of sparse matrix representation and computation, both sequentially and in parallel, shared by all algorithms. Chapter 11 describes some preconditioning techniques that are the subject of current research.

It is inevitable that a large amount of important material needs to be excluded from this book. In the appendix, we list some of the subjects not covered in this book, giving references for the reader who is interested in pursuing a particular subject in depth.

The field of numerical methods for solving eigenvalue problems is in constant flux, with new methods and approaches continually being created, modified, tuned, and

eventually, in some cases, discarded. We expect the material in this book to undergo changes from time to time as some of these new methods mature and become the state of the art. Therefore, we plan to update the material included in this book periodically in future editions. We welcome your comments and criticisms of this work to help us in that updating process. Please send your comments and questions by email to etemplates@cs.utk.edu.

The book's Web site describes how to access software discussed in this book and many other related topics. The URL for the book's homepage will be referred to as ETHOME throughout the text, which stands for <http://www.netlib.org/etemplates/>

How to Cite This Book

The full reference to the book should be:

Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst, editors. *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*. SIAM, Philadelphia, 2000.

The reference to a section with the indicated authorship should be, for example:

R. Lehoucq and D. Sorensen. Implicitly Restarted Lanczos Method (Section 4.5). In Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst, editors, *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*, pages 67–80, SIAM, Philadelphia, 2000.

This page intentionally left blank

Acknowledgments

We gratefully acknowledge the valuable comments of many people at the different stages of this project. We are particularly indebted to Beresford N. Parlett, who has provided invaluable comments and encouragement from the beginning.

In addition, Zhaojun Bai would like to thank Greg Ammar, Nicholas Higham, Zhongxiao Jia, G. W. Stewart, Hongguo Xu, and Qiang Ye for the fruitful discussion on the project. Part of his work was done while he was on the faculty of the Department of Mathematics, University of Kentucky. Bai and Tom Kowalski were supported in part by NSF grants ASC-9313958 and ACI-9813362 and DOE grant DE-FG03-94ER25219.

Tzu-Yi Chen was supported in part by an NSF graduate fellowship.

David Day, Rich Lehoucq, and Ross Lippert are affiliated with Sandia National Laboratories. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the U.S. Department of Energy under contract DE-AC04-94AL85000.

Jim Demmel's research was supported in part by DOE grants DE-FG03-94ER25219 and DE-FC03-98ER25351 and DOE contract W-7405-ENG-48; by NSF grants ASC-9313958 and ACI-9813361, NSF cooperative agreement ACI-9619020, and NSF Infrastructure grant EIA-9802069; and by gifts from the IBM Shared University Research Program, Sun Microsystems, and Intel. His work also utilized resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Energy Research of DOE.

Jack Dongarra would like to acknowledge the supports of NSF grants ASC-9313958 and ACI-9813362 and DOE grant DE-FG03-94ER25219 for this work.

Ren-Cang Li was supported by NSF grant ACI-9721388 and by an NSF CAREER award under grant CCR-9875201.

Xiaoye Li's work was supported by the Director, Office of Science, Division of Mathematical, Information, and Computational Sciences of the U.S. Department of Energy under contract number DE-AC03-76SF00098.

Ross Lippert and Alan Edelman would like to thank Steve Smith for many invaluable discussions. Ross Lippert and Alan Edelman were supported by NSF grant DMS-9501278.

Bo Kågström would like to acknowledge his coauthors on several papers on singular matrix pencils, namely, J. Demmel, A. Edelman, E. Elmroth, and P. Johansson.

Andrew Knyazev was partially supported by NSF grant DMS-9501507.

Axel Ruhe started his work while on sabbatical at the University of California at Berkeley, where he enjoyed the kind hospitality of Beresford N. Parlett. He received travel grants from the Swedish research councils for natural science (NFR), engineering science (TFR), and the Royal Society of Arts and Sciences in Göteborg.

Gerard Sleijpen and Henk van der Vorst were supported by the Netherlands organization for scientific research, NWO, through the Priority Program MPR, grant number 95MPR04.

The information presented here does not necessarily reflect the position or the policy of the funding agencies and no official endorsement should be inferred.

Finally, the editors thank Victor Eijkhout who helped with editing and L^AT_EX-related issues. We are also indebted to the staff at SIAM, in particular Vickie Kearn, Deborah Poulson, Kelly Thomas, and Marianne Will for their great job of production.

Contents

List of Symbols and Acronyms	xxi
List of Iterative Algorithm Templates	xxiii
List of Direct Algorithms	xxv
List of Figures	xxvii
List of Tables	xxix
1 Introduction	1
1.1 Why Eigenvalue Templates?	1
1.2 Intended Readership	2
1.3 Using the Decision Tree to Choose a Template	3
1.4 What Is a Template?	3
1.5 Organization of the Book	4
2 A Brief Tour of Eigenproblems	7
2.1 Introduction	7
2.1.1 Numerical Stability and Conditioning	10
2.2 Hermitian Eigenproblems	
<i>J. Demmel</i>	11
2.2.1 Eigenvalues and Eigenvectors	11
2.2.2 Invariant Subspaces	12
2.2.3 Equivalences (Similarities)	12
2.2.4 Eigendecompositions	12
2.2.5 Conditioning	12
2.2.6 Specifying an Eigenproblem	13
2.2.7 Related Eigenproblems	13
2.2.8 Example	14
2.3 Generalized Hermitian Eigenproblems	
<i>J. Demmel</i>	14
2.3.1 Eigenvalues and Eigenvectors	15
2.3.2 Eigenspaces	15
2.3.3 Equivalences (Congruences)	15
2.3.4 Eigendecompositions	15
2.3.5 Conditioning	16

2.3.6	Specifying an Eigenproblem	16
2.3.7	Related Eigenproblems	17
2.3.8	Example	17
2.4	Singular Value Decomposition	
<i>J. Demmel</i>	18
2.4.1	Singular Values and Singular Vectors	18
2.4.2	Singular Subspaces	19
2.4.3	Equivalences	19
2.4.4	Decompositions	19
2.4.5	Conditioning	20
2.4.6	Specifying a Singular Value Problem	20
2.4.7	Related Singular Value Problems	21
2.4.8	Example	22
2.5	Non-Hermitian Eigenproblems	
<i>J. Demmel</i>	23
2.5.1	Eigenvalues and Eigenvectors	23
2.5.2	Invariant Subspaces	24
2.5.3	Equivalences (Similarities)	24
2.5.4	Eigendecompositions	24
2.5.5	Conditioning	26
2.5.6	Specifying an Eigenproblem	26
2.5.7	Related Eigenproblems	27
2.5.8	Example	28
2.6	Generalized Non-Hermitian Eigenproblems	
<i>J. Demmel</i>	28
2.6.1	Eigenvalues and Eigenvectors	29
2.6.2	Deflating Subspaces	29
2.6.3	Equivalences	29
2.6.4	Eigendecompositions	29
2.6.5	Conditioning	31
2.6.6	Specifying an Eigenproblem	32
2.6.7	Related Eigenproblems	33
2.6.8	Example	33
2.6.9	Singular Case	34
2.7	Nonlinear Eigenproblems	
<i>J. Demmel</i>	36
3	An Introduction to Iterative Projection Methods	37
3.1	Introduction	37
3.2	Basic Ideas	
<i>Y. Saad</i>	37
3.3	Spectral Transformations	
<i>R. Lehoucq and D. Sorensen</i>	43
4	Hermitian Eigenvalue Problems	45
4.1	Introduction	45
4.2	Direct Methods	49

4.3 Single- and Multiple-Vector Iterations	
<i>M. Gu</i>	51
4.3.1 Power Method	51
4.3.2 Inverse Iteration	52
4.3.3 Rayleigh Quotient Iteration	53
4.3.4 Subspace Iteration	54
4.3.5 Software Availability	56
4.4 Lanczos Method	
<i>A. Ruhe</i>	56
4.4.1 Algorithm	57
4.4.2 Convergence Properties	59
4.4.3 Spectral Transformation	60
4.4.4 Reorthogonalization	61
4.4.5 Software Availability	63
4.4.6 Numerical Examples	63
4.5 Implicitly Restarted Lanczos Method	
<i>R. Lehoucq and D. Sorensen</i>	67
4.5.1 Implicit Restart	67
4.5.2 Shift Selection	69
4.5.3 Lanczos Method in GEMV Form	70
4.5.4 Convergence Properties	71
4.5.5 Computational Costs and Tradeoffs	72
4.5.6 Deflation and Stopping Rules	73
4.5.7 Orthogonal Deflating Transformation	74
4.5.8 Implementation of Locking and Purging	79
4.5.9 Software Availability	80
4.6 Band Lanczos Method	
<i>R. Freund</i>	80
4.6.1 The Need for Deflation	81
4.6.2 Basic Properties	82
4.6.3 Algorithm	85
4.6.4 Variants	86
4.7 Jacobi–Davidson Methods	
<i>G. Sleijpen and H. van der Vorst</i>	88
4.7.1 Basic Theory	88
4.7.2 Basic Algorithm	91
4.7.3 Restart and Deflation	95
4.7.4 Computing Interior Eigenvalues	99
4.7.5 Software Availability	102
4.7.6 Numerical Example	103
4.8 Stability and Accuracy Assessments	
<i>Z. Bai and R. Li</i>	105
5 Generalized Hermitian Eigenvalue Problems	109
5.1 Introduction	109
5.2 Transformation to Standard Problem	112
5.3 Direct Methods	113

5.4	Single- and Multiple-Vector Iterations <i>M. Gu</i>	114
5.5	Lanczos Methods <i>A. Ruhe</i>	116
5.6	Jacobi–Davidson Methods <i>G. Sleijpen and H. van der Vorst</i>	123
5.7	Stability and Accuracy Assessments <i>Z. Bai and R. Li</i>	127
	5.7.1 Positive Definite B	128
	5.7.2 Some Combination of A and B is Positive Definite	130
6	Singular Value Decomposition	135
6.1	Introduction	135
6.2	Direct Methods	138
6.3	Iterative Algorithms <i>J. Demmel</i>	140
	6.3.1 What Operations Can One Afford to Perform?	140
	6.3.2 Which Singular Values and Vectors Are Desired?	141
	6.3.3 Golub–Kahan–Lanczos Method	142
	6.3.4 Software Availability	145
	6.3.5 Numerical Example	146
6.4	Related Problems <i>J. Demmel</i>	146
7	Non-Hermitian Eigenvalue Problems	149
7.1	Introduction	149
7.2	Balancing Matrices <i>T. Chen and J. Demmel</i>	152
	7.2.1 Direct Balancing	153
	7.2.2 Krylov Balancing Algorithms	154
	7.2.3 Accuracy of Eigenvalues Computed after Balancing	155
7.3	Direct Methods	157
7.4	Single- and Multiple-Vector Iterations <i>M. Gu</i>	158
	7.4.1 Power Method	159
	7.4.2 Inverse Iteration	159
	7.4.3 Subspace Iteration	159
	7.4.4 Software Availability	160
7.5	Arnoldi Method <i>Y. Saad</i>	161
	7.5.1 Basic Algorithm	161
	7.5.2 Variants	163
	7.5.3 Explicit Restarts	163
	7.5.4 Deflation	163
7.6	Implicitly Restarted Arnoldi Method <i>R. Lehoucq and D. Sorensen</i>	166
	7.6.1 Arnoldi Procedure in GEMV Form	167
	7.6.2 Implicit Restart	169

7.6.3	Convergence Properties	172
7.6.4	Numerical Stability	173
7.6.5	Computational Costs and Tradeoffs	174
7.6.6	Deflation and Stopping Rules	175
7.6.7	Orthogonal Deflating Transformation	176
7.6.8	Eigenvector Computation with Spectral Transformation	183
7.6.9	Software Availability	184
7.7	Block Arnoldi Method <i>R. Lehoucq and K. Maschhoff</i>	185
7.7.1	Block Arnoldi Reductions	185
7.7.2	Practical Algorithm	187
7.8	Lanczos Method <i>Z. Bai and D. Day</i>	189
7.8.1	Algorithm	189
7.8.2	Convergence Properties	193
7.8.3	Software Availability	195
7.8.4	Notes and References	195
7.9	Block Lanczos Methods <i>Z. Bai and D. Day</i>	196
7.9.1	Basic Algorithm	196
7.9.2	An Adaptively Blocked Lanczos Method	199
7.9.3	Software Availability	204
7.9.4	Notes and References	204
7.10	Band Lanczos Method <i>R. Freund</i>	205
7.10.1	Deflation	206
7.10.2	Basic Properties	206
7.10.3	Algorithm	210
7.10.4	Application to Reduced-Order Modeling	214
7.10.5	Variants	215
7.11	Lanczos Method for Complex Symmetric Eigenproblems <i>R. Freund</i>	216
7.11.1	Properties of Complex Symmetric Matrices	216
7.11.2	Properties of the Algorithm	217
7.11.3	Algorithm	219
7.11.4	Solving the Reduced Eigenvalue Problems	219
7.11.5	Software Availability	220
7.11.6	Notes and References	220
7.12	Jacobi–Davidson Methods <i>G. Sleijpen and H. van der Vorst</i>	221
7.12.1	Generalization of Hermitian Case	221
7.12.2	Schur Form and Restart	221
7.12.3	Computing Interior Eigenvalues	224
7.12.4	Software Availability	227
7.12.5	Numerical Example	227
7.13	Stability and Accuracy Assessments <i>Z. Bai and R. Li</i>	228

8 Generalized Non-Hermitian Eigenvalue Problems	233
8.1 Introduction	233
8.2 Direct Methods	234
8.3 Transformation to Standard Problems	235
8.4 Jacobi–Davidson Method <i>G. Sleijpen and H. van der Vorst</i>	238
8.4.1 Basic Theory	238
8.4.2 Deflation and Restart	240
8.4.3 Algorithm	241
8.4.4 Software Availability	244
8.4.5 Numerical Example	244
8.5 Rational Krylov Subspace Method <i>A. Ruhe</i>	246
8.6 Symmetric Indefinite Lanczos Method <i>Z. Bai, T. Ericsson, and T. Kowalski</i>	249
8.6.1 Some Properties of Symmetric Indefinite Matrix Pairs	250
8.6.2 Algorithm	252
8.6.3 Stopping Criteria and Accuracy Assessment	256
8.6.4 Singular B	257
8.6.5 Software Availability	258
8.6.6 Numerical Examples	258
8.7 Singular Matrix Pencils <i>B. Kågström</i>	260
8.7.1 Regular Versus Singular Problems	261
8.7.2 Kronecker Canonical Form	262
8.7.3 Generic and Nongeneric Kronecker Structures	263
8.7.4 Ill-Conditioning	264
8.7.5 Generalized Schur-Staircase Form	266
8.7.6 GUPTRI Algorithm	267
8.7.7 Software Availability	271
8.7.8 More on GUPTRI and Numerical Examples	271
8.7.9 Notes and References	276
8.8 Stability and Accuracy Assessments <i>Z. Bai and R. Li</i>	277
9 Nonlinear Eigenvalue Problems	281
9.1 Introduction	281
9.2 Quadratic Eigenvalue Problems <i>Z. Bai, G. Sleijpen, and H. van der Vorst</i>	281
9.2.1 Introduction	281
9.2.2 Transformation to Linear Form	283
9.2.3 Spectral Transformations for QEP	284
9.2.4 Numerical Methods for Solving Linearized Problems	286
9.2.5 Jacobi–Davidson Method	287
9.2.6 Notes and References	289
9.3 Higher Order Polynomial Eigenvalue Problems	289
9.4 Nonlinear Eigenvalue Problems with Orthogonality Constraints <i>R. Lippert and A. Edelman</i>	290

9.4.1	Introduction	290
9.4.2	MATLAB Templates	291
9.4.3	Sample Problems and Their Differentials	293
9.4.4	Numerical Examples	297
9.4.5	Modifying the Templates	304
9.4.6	Geometric Technicalities	308
10	Common Issues	315
10.1	Sparse Matrix Storage Formats <i>J. Dongarra</i>	315
10.1.1	Compressed Row Storage	315
10.1.2	Compressed Column Storage	316
10.1.3	Block Compressed Row Storage	316
10.1.4	Compressed Diagonal Storage	317
10.1.5	Jagged Diagonal Storage	318
10.1.6	Skyline Storage	319
10.2	Matrix-Vector and Matrix-Matrix Multiplications <i>J. Dongarra, P. Koev, and X. Li</i>	320
10.2.1	BLAS	320
10.2.2	Sparse BLAS	321
10.2.3	Fast Matrix-Vector Multiplication for Structured Matrices	324
10.3	A Brief Survey of Direct Linear Solvers <i>J. Demmel, P. Koev, and X. Li</i>	326
10.3.1	Direct Solvers for Dense Matrices	328
10.3.2	Direct Solvers for Band Matrices	328
10.3.3	Direct Solvers for Sparse Matrices	329
10.3.4	Direct Solvers for Structured Matrices	331
10.4	A Brief Survey of Iterative Linear Solvers <i>H. van der Vorst</i>	332
10.5	Parallelism <i>J. Dongarra and X. Li</i>	334
11	Preconditioning Techniques	337
11.1	Introduction	337
11.2	Inexact Methods <i>K. Meerbergen and R. Morgan</i>	339
11.2.1	Matrix Transformations	340
11.2.2	Inexact Matrix Transformations	340
11.2.3	Arnoldi Method with Inexact Cayley Transform	342
11.2.4	Davidson Method	343
11.2.5	Jacobi–Davidson Method with Cayley Transform	346
11.2.6	Preconditioned Lanczos Method	346
11.2.7	Inexact Rational Krylov Method	348
11.2.8	Inexact Shift-and-Invert	352
11.3	Preconditioned Eigensolvers <i>A. Knyazev</i>	352
11.3.1	Introduction	352
11.3.2	General Framework of Preconditioning	354

11.3.3 Preconditioned Shifted Power Method	356
11.3.4 Preconditioned Steepest Ascent/Descent Methods	357
11.3.5 Preconditioned Lanczos Methods	358
11.3.6 Davidson Method	360
11.3.7 Methods with Preconditioned Inner Iterations	361
11.3.8 Preconditioned Conjugate Gradient Methods	362
11.3.9 Preconditioned Simultaneous Iterations	363
11.3.10 Software Availability	367
Appendix. Of Things Not Treated	369
Bibliography	373
Index	405

List of Symbols and Acronyms

Symbols

A, \dots, Z	matrices
a, \dots, z	vectors
$\alpha, \beta, \dots, \omega$	scalars
A^T	matrix transpose
A^*	conjugate transpose of A
A^{-1}	matrix inverse
A^{-T}	the inverse of A^T
A^{-*}	the inverse of A^*
$A^{m \times n}$	indicates that A is m -by- n
a_{ij}	matrix element of A
$a_{i,:}$	i th row of matrix A
$a_{:,j}$	j th column of matrix A
x^*y	vector dot product
(x, y)	inner product, such as x^*y
v_j	vector v in the j th iteration
$\text{diag}(A)$	diagonal of matrix A
$\text{diag}(\alpha, \beta, \dots)$	diagonal matrix constructed from scalars α, β, \dots
$\text{span}(a, b, \dots)$	spanning space of vectors a, b, \dots
$\text{span}(A)$	$= \text{range}(A)$, subspace spanned by the columns of A
$\mathcal{K}^m(A, v)$	Krylov subspace $\text{span}\{v, Av, A^2v, \dots, A^{m-1}v\}$
$[x_1, x_2, \dots, x_m]$	matrix whose i th column is x_i
$X(i, j)$	(i, j) entry of X
$X(:, i)$ or $X(i, :)$	i th column or row of X
$X(:, [2, 3, 5])$	submatrix consisting of columns 2, 3, and 5 of X
$X([2, 3, 5], :)$	submatrix consisting of rows 2, 3, and 5 of X
\mathbb{R}, \mathbb{C}	sets of real and complex numbers
$\mathbb{R}^n, \mathbb{C}^n$	real and complex n -spaces
$\ x\ _p, \ A\ _p$	vector and matrix p -norm
$\ x\ _A$	the “ A -norm,” defined as $(Ax, x)^{1/2}$
$\ A\ _F$	matrix Frobenius norm
ϵ_M	machine precision
$\lambda(A)$	eigenvalues of A
$\lambda_{\max}(A), \lambda_{\min}(A)$	eigenvalues of A with maximum (resp., minimum) modulus
$\sigma(A)$	singular values of A

$\sigma_{\max}(A), \sigma_{\min}(A)$	largest and smallest singular values of A
$\kappa_2(A)$	condition number of matrix A , defined as $\kappa_2(A) = \ A\ _2\ A^{-1}\ _2$
$\bar{\alpha}$	complex conjugate of the scalar α
$\max\{S\}$	maximum value in set S
$\min\{S\}$	minimum value in set S
\sum	summation
$O(\cdot)$	“big-oh” asymptotic bound
$\Re e(a)$	the real part of the complex number a
$\Im m(a)$	the imaginary part of the complex number a
$u \perp v$	the vector u is orthogonal to the vector v

Acronyms

BLAS	Basic Linear Algebra Subprograms
CPU	Central processing unit
ETHOME	Abbreviation of the book’s homepage, http://www.netlib.org/etemplates/
GHEP	Generalized Hermitian eigenvalue problem
GNHEP	Generalized non-Hermitian eigenvalue problem
GSVD	Generalized singular value decomposition
HEP	Hermitian eigenvalue problem
NHEP	Non-Hermitian eigenvalue problem
PEP	Polynomial eigenvalue problem
QEP	Quadratic eigenvalue problem
SI	Shift-and-invert spectral transformation
SVD	Singular value decomposition

List of Iterative Algorithm Templates

4.1	Power Method for HEP	52
4.2	Inverse Iteration for HEP	52
4.3	RQI for HEP	53
4.4	Simple Subspace Iteration for HEP	54
4.5	Subspace Iteration with Projection and Deflation for HEP	55
4.6	Lanczos Method for HEP	57
4.7	IRLM for HEP	68
4.8	m -Step Lanczos Factorization	71
4.9	Orthogonal Deflating Transformation for IRLM	75
4.10	Stable Orthogonal Deflating Transformation for IRLM	78
4.11	Deflation for IRLM	79
4.12	Band Lanczos Method for HEP	85
4.13	Jacobi–Davidson Method for $\lambda_{\max}(A)$ for HEP	91
4.14	Modified Gram–Schmidt Orthogonalization with Refinement	93
4.15	Approximate Solution of the Jacobi–Davidson HEP Correction Equation with Left-Preconditioning	94
4.16	Approximate Solution of the Jacobi–Davidson HEP Correction Equation with Right-Preconditioning	94
4.17	Jacobi–Davidson Method for k_{\max} Exterior Eigenvalues for HEP	97
4.18	Approximate Solution of the Deflated Jacobi–Davidson HEP Correction Equation	99
4.19	Jacobi–Davidson Method for k_{\max} Interior Eigenvalues at the Right Side Nearest to τ for HEP	101
5.1	Power Method for GHEP	114
5.2	Inverse Iteration for GHEP	115
5.3	RQI for GHEP	115
5.4	Lanczos Method for GHEP	117
5.5	Shift-and-Invert Lanczos Method for GHEP	120
5.6	Jacobi–Davidson Method for k_{\max} Exterior Eigenvalues for GHEP	124
5.7	Approximate Solution of the Deflated Jacobi–Davidson GHEP Correction Equation	127
6.1	Golub–Kahan–Lanczos Bidiagonalization Procedure	143
7.1	Krylov Balancing Algorithm for NHEP (KRYLOVCUTOFF)	154
7.2	Subspace Iteration with Projection and Deflation for NHEP	160

7.3	Arnoldi Procedure	161
7.4	Explicitly Restarted Arnoldi Method for NHEP	163
7.5	Explicitly Restarted Arnoldi Method with Deflation for NHEP	165
7.6	k -Step Arnoldi Factorization	167
7.7	IRAM for NHEP	170
7.8	Orthogonal Deflating Transformation for IRAM	176
7.9	Stable Orthogonal Deflating Transformation for IRAM	181
7.10	Deflation for IRAM	182
7.11	Extending a Block Arnoldi Reduction	186
7.12	BIRAM for NHEP	187
7.13	Lanczos Method for NHEP	191
7.14	Block Lanczos Method for NHEP	198
7.15	ABLE for NHEP	201
7.16	Band Lanczos Method for NHEP	211
7.17	Complex Symmetric Lanczos Method	219
7.18	Jacobi–Davidson Method for k_{\max} Exterior Eigenvalues for NHEP	222
7.19	Jacobi–Davidson Method for k_{\max} Interior Eigenvalues for NHEP	226
8.1	Jacobi–Davidson QZ Method for k_{\max} Interior Eigenvalues Close to τ for GNHEP	242
8.2	Approximate Solution of the Deflated Jacobi–Davidson GNHEP Correction Equation	245
8.3	Rational Krylov Subspace Method for GNHEP	249
8.4	Symmetric Indefinite Lanczos Method	254
9.1	Jacobi–Davidson Method for QEP	288
9.2	SG_MIN	305
11.1	Arnoldi Method with Inexact Cayley Transform for GNHEP	343
11.2	Generalized Davidson Method for GNHEP	344
11.3	Preconditioned Lanczos Method for GHEP	347
11.4	Inexact Rational Krylov Method for GNHEP	348
11.5	Preconditioned Power Method for GHEP	356
11.6	Preconditioned Steepest Ascent Method for GHEP	358
11.7	Preconditioned Lanczos Method for GHEP	359
11.8	Preconditioned Projection Method for GHEP	360
11.9	RQI with Preconditioned Inner Iterative Solver for GHEP	361
11.10	PCG Method for GHEP	362
11.11	Simultaneous PCG Method for GHEP	363

List of Direct Algorithms

4.2 QR algorithm for HEP	49
4.2 Divide-and-conquer algorithm for HEP	49
4.2 Bisection method and inverse for HEP	49
4.2 Inverse iteration for HEP	49
4.2 Relatively robust representation algorithm for HEP	49
5.3 QR algorithm for GHEP	113
5.3 Divide-and-conquer algorithm for GHEP	113
5.3 Bisection method and inverse iteration for GHEP	113
6.2 QR algorithm for SVD	138
6.2 Divide-and-conquer QR algorithm for SVD	138
6.2 DQDS algorithm for SVD	138
6.2 Bisection method and inverse iteration for SVD	138
7.3 QR algorithm for NHEP	157
8.2 QZ algorithm for GNHEP	234
8.2 GUPTRI algorithm for GNHEP of singular pencils	260

Note that a direct algorithm must still iterate, since finding eigenvalues is mathematically equivalent to finding zeros of polynomials, for which no noniterative methods can exist. We call a method direct if experience shows that it (nearly) never fails to converge in a fixed number of iterations.

This page intentionally left blank

List of Figures

2.1	Damped, vibrating mass-spring system.	9
4.1	Residual estimates, L-shaped membrane matrix.	64
4.2	Ritz values, L-shaped membrane matrix.	65
4.3	Residual estimates, Medline SVD matrix	66
4.4	Residual estimates, shift-and-invert L-shaped membrane matrix.	66
4.5	Jacobi–Davidson for exterior eigenvalues with several strategies for solving the correction equation.	103
4.6	Jacobi–Davidson for exterior eigenvalues (top) and interior eigenvalues (bottom). The correction equations have been solved with 5 steps of plain GMRES (left) and with 5 steps of preconditioned GMRES (right).	104
5.1	Residual estimates, Lanczos with shift-and-invert, L-membrane 9-point finite difference approximation.	122
7.1	Relative accuracy of eigenvalues computed with and without direct balancing for the QH768 and TOLOSA matrices.	155
7.2	Relative accuracy of eigenvalues computed with and without Krylov balancing for the QH768 and TOLOSA matrices.	156
7.3	Comparison of the relative accuracy of the largest and smallest (in magnitude) eigenvalues of the QH768 matrix, with different Krylov-based balancing algorithms, using the default settings of five iterations and a cutoff value of 10^{-8} .	156
7.4	Comparison of the relative accuracy of the largest and smallest (in magnitude) eigenvalues of the TOLOSA matrix, with different Krylov-based balancing algorithms, using the default settings of five iterations and a cutoff value of 10^{-8} .	157
7.5	Jacobi–Davidson for exterior eigenvalues (left side) and interior eigenvalues (right side).	228
8.1	Convergence history for BFW782.	246
9.1	Procrustes problem .	299
9.2	Jordan problem .	300
9.3	Trace minimization problem .	301
9.4	LDA toy problem .	302
9.5	Simultaneous Schur problem .	303
9.6	Simultaneous diagonalization problem .	304

9.7	The unconstrained differential of $F(Y)$ can be projected to the tangent space to obtain the covariant gradient, G , of F	311
9.8	In a flat space, comparing vectors at nearby points is not problematic since all vectors lie in the same tangent space.	313
9.9	In a curved manifold, comparing vectors at nearby points can result in vectors which do not lie in the tangent space.	313
10.1	Profile of a nonsymmetric skyline or variable-band matrix.	319
11.1	Logarithmic plots of residual norms of the inexact rational Krylov method for the Olmstead problem. The circles denote $\ f_j\ $ and the bullets $\ r_j\ $	351
11.2	Conjugate gradient versus steepest ascent, $\delta_1/\delta_0 = 10$	365
11.3	Conjugate gradient versus steepest ascent, $\delta_1/\delta_0 = 100$	365
11.4	Conjugate gradient versus steepest ascent, $\delta_1/\delta_0 = 1000$	366

List of Tables

4.1	Summary of algorithms for HEPs	48
5.1	Summary of algorithms for GHEPs	111
7.1	Summary of algorithms for NHEPs	152
8.1	Five generalized eigenvalues of BFW782, computed by Jacobi–Davidson QZ	245
8.2	Summary of results for BFW782.	246
9.1	A short list of the optional arguments for sg_min	292
10.1	Software to solve sparse linear systems using direct methods.	330
11.1	Preconditioning the model problem	345
11.2	Residual norms of the inexact rational Krylov method for the Olmstead problem	351

This page intentionally left blank

Chapter 1

Introduction

1.1 Why Eigenvalue Templates?

In many large scale scientific or engineering computations, ranging from computing the frequency response of a circuit to the earthquake response of a building to the energy levels of a molecule, one needs to find eigenvalues and eigenvectors of a matrix. There are many mathematical ways to formulate eigenvalue problems, and even more ways have been proposed to solve them numerically. This book is intended to be a guide to finding the best numerical method for an eigenvalue problem.

The current state of the art is such that excellent methods exist for many eigenproblems, especially for small- to medium-sized dense matrices. These algorithms have been made available in programming environments like MATLAB [319],¹ libraries like LAPACK [12], and many other commercial and public packages. But for very large and (typically) sparse eigenvalue problems no single best method exists. The sheer number of methods and the complicated ways they depend on mathematical properties of the matrix and trade off efficiency and accuracy make it difficult for experts, let alone general users, to find the best method for a given problem. Good online search facilities and software repositories exist, notably GAMS (Guide to Available Mathematical Software)² and NETLIB.³ These facilities permit searches based on library names, subroutines names, key words, and a taxonomy of topics in numerical computing. But they will not give advice as to which method is best to use for a particular problem. As a result, the authors of this book and other experts are frequently asked for advice in choosing an algorithm. This situation has motivated us to distill our knowledge into this book to make it as widely available as possible.

Here is how we have structured the book. First, we have developed a *decision tree* to guide the user to the best method. Each node in the tree poses questions about the problem regarding its mathematical structure, the quantities to be computed, and the cost of applying certain operations to the matrix. Second, at the leaves of the decision tree we present *templates* of recommended algorithms, as well as pointers to available software. A template is a high-level description of an algorithm, suitable for understanding how it works, what parameters the user can tune to control efficiency

¹MATLAB is a registered trademark of The MathWorks, Inc.

²<http://gams.nist.gov>, developed by NIST (National Institute of Standards and Technology).

³<http://www.netlib.org>

and accuracy, and how to interpret the output. We discuss this structure in more detail below.

The original model for this book was *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods* [41]. This earlier book (which has some authors in common with this one) provided a decision tree and templates for iterative methods for the simpler problem of solving a linear system. Eigenvalue problems are inherently more difficult than linear systems, and this means that our level of understanding varies significantly, depending on the eigenvalue problem at hand. Furthermore, the algorithms are much more complicated. So in contrast to this earlier *Templates* book, we have not attempted to provide complete implementations of each template in a common style and in common programming languages. Instead, we provide a high-level template of each algorithm and point to good public domain implementations in whatever language they are available. All these implementations may be accessed through a Web site we call ETHOME in the text, which refers to <http://www.netlib.org/etemplates/>. This Web site will be updated as new and better algorithms become available.

The variety of eigenproblems covered in this book is very wide. Some algorithms are well known and have been thoroughly investigated, whereas other areas are still subject to intensive research. This has led to differences in the style of presentation. Well-understood areas with good “black box” solutions have been kept brief and are written primarily for nonexpert readers, whereas at the other extreme there are contributions that review current research.

In most sections we explicitly name the contributing authors. This is not only for credit reasons, but also reflects personal responsibility for the text. All sections provide sufficient references to serve as windows to a richer world of detail and expertise on eigenproblems.

1.2 Intended Readership

We hope our templates will address the needs of three potential user communities for eigenvalue computation tools:

- Students and teachers who want simple but generally effective algorithms and would like access to the latest developments in a form that is easy to explain and to understand.
- General engineers and scientists who desire easy-to-use, reliable software that is also reasonably efficient.
- The high-performance computing community consisting of those engineers and scientists trying to solve the largest, hardest applications in their fields. Such users desire high-speed access to algorithmic details for performance tuning and to reliability for their problem.

It may seem difficult to address the needs of such diverse communities with a single document. Nevertheless, we believe that templates allow us to organize the information in ways useful to all. For example, we expect students to use the high-level algorithm descriptions to acquire a basic understanding of how the methods work. General engineers and scientists should find the decision tree and pointers to software adequate for most of their problems. Finally, the high-performance computing community can

use the advice on performance tuning and assessing the accuracy of the output to help solve their hardest problems.

1.3 Using the Decision Tree to Choose a Template

Eigenvalue problems and their associated numerical algorithms may be classified according to the following properties:

1. *Mathematical properties of the problem:* Is it a Hermitian (real symmetric, self-adjoint) or a non-Hermitian eigenproblem? Is it a standard problem involving only one matrix or a generalized problem with two matrices? Based on these and other questions, Chapter 2 identifies six mathematical types of eigenvalue problems. This is the top of the decision tree. Each of Chapters 4 through 9 treats one of the six types.
2. *Desired spectral information:* Do we need just the smallest eigenvalue, a few eigenvalues at either end of the spectrum, a subset of eigenvalues “inside” the spectrum, or most eigenvalues? Do we want associated eigenvectors, invariant subspaces, or other quantities? How much accuracy is desired?
3. *Available operations and their costs:* Can we store the full matrix as an array and perform a similarity transformation on it? Can we solve a linear system with the matrix (or shifted matrix) by a direct factorization routine, or perhaps an iterative method? Or can we only multiply a vector by the matrix, and perhaps by its transpose? If several of these operations are possible, what are their relative costs?

Based on the desired spectral information and available operations and their costs, the first sections of Chapters 4 through 9 make recommendations on choosing one of the algorithms they present. These recommendations are summarized in tabular form when appropriate; see Tables 4.1, 5.1, and 7.1.

The state of the art is such that we do not have efficient algorithms for all eigenvalue problems that the user might pose. For example, suppose the user wants all the eigenvalues nearest the imaginary axis of an n -by- n matrix A with eigenvalues throughout the complex plane, but where the only available operation is to multiply A by a vector. Then we do not have an algorithm that is simultaneously efficient (i.e., performs many fewer than n matrix-vector multiplications) and reliable (i.e., guaranteed or very likely to find all eigenvalues within a user-specified distance of the imaginary axis). These limitations are discussed in the text.

1.4 What Is a Template?

A template will include some or all of the following information:

- A high-level description of an algorithm.
- A description of when the algorithm is effective, including conditions on the input and estimates of the time, space, or other resources required. If there are natural competitors, they will be referenced.

- A description of available refinements and user-tunable parameters, as well as advice on when to use them.
- A way to assess the accuracy.
- Numerical examples, illustrating both easy and difficult cases.
- Pointers to complete or partial implementations, perhaps in several languages or for several computer architectures.
- Pointers to texts or journal articles for further information.

This level of description is used for iterative methods where the details are needed to use the algorithm effectively. We have deliberately used the word “template” instead of “numerical recipe” to emphasize that the methods cannot be used blindly, and that instead knowledge of both the problem and algorithm is needed to effectively solve the problem. “Black box” methods that are most suitable for small- to medium-sized problems are described in less detail.

1.5 Organization of the Book

The book is divided into 11 chapters. We refer the reader to the list of symbols and acronyms on page xxi; we will use the notation listed there freely throughout the text.

Chapter 1 motivates and outlines the rest of the book.

Chapter 2 gives a tour of all the eigenvalue problems discussed in the book. If you are a first-time reader, read this chapter! It gives basic terminology and definitions used to describe eigenvalue problems, and classifies all eigenvalue problems into six types. This is the top of the decision tree. Chapters 4 through 9 give details for each of the six types.

Chapter 3 summarizes the two mathematical principles used by most algorithms for large eigenvalue problems: projection onto subspaces and spectral transformations.

Chapter 4 covers the Hermitian eigenvalue problem, $Ax = \lambda x$. Here $A = A^*$ is a Hermitian matrix (if A is real, this means A is symmetric). Chapters 4 through 9 all begin with advice on how to choose an algorithm and a brief discussion of transformation methods, which are the standard methods for small- to medium-sized dense matrices. Then there are more detailed templates for each major iterative method. For the Hermitian eigenvalue problem, which is the best understood case, these include the power method, inverse iteration, Rayleigh quotient iteration, subspace iteration, the Lanczos method, the implicitly restarted Lanczos method, the band Lanczos method, and the Jacobi–Davidson method. Each chapter ends with a summary of stability and accuracy assessment.

Chapter 5 covers generalized Hermitian eigenvalue problems $Ax = \lambda Bx$, where $A = A^*$ and $B = B^*$ are Hermitian and B is also positive definite. The methods discussed are analogs of those discussed in Chapter 4.

Chapter 6 covers the singular value decomposition $A = U\Sigma V^*$. The chapter emphasizes methods for large sparse matrices that compute just selected parts of this decomposition. The methods discussed are analogs of those discussed in Chapter 4.

Chapter 7 covers the non-Hermitian eigenvalue problem $Ax = \lambda x$, where A is a general square matrix. At the heart of the chapter are sections on Arnoldi’s method, the

non-Hermitian Lanczos method, and the Jacobi–Davidson method, including implicitly restarted, block, and band variants.

Chapter 8 covers the generalized non-Hermitian eigenvalue problem, $Ax = \lambda Bx$. This includes both the *regular case*, where A and B are n by n and have n eigenvalues that are continuous functions of the entries of A and B , and the *singular case*, where there may be fewer than n eigenvalues, discontinuous eigenvalues, or (when A and B are not square) no eigenvalues at all. In the regular case, we discuss oblique projection Jacobi–Davidson, rational Krylov, and a special variant of the Lanczos method for symmetric indefinite pairs.

Chapter 9 covers two kinds of nonlinear eigenvalue problems. First, we discuss polynomial eigenvalue problems, which are defined by three or more matrices. Second, we discuss nonlinear eigenvalue problems with orthogonality constraints.

Chapter 10 describes common issues of sparse matrix representation and computation, both sequentially and in parallel, shared by all algorithms. This includes a list of standard sparse matrix formats, algorithms for fast matrix–vector multiplication (the “BLAS”), a survey of direct linear solvers, a survey of iterative linear solvers, and a brief discussion of how parallelism may be exploited.

Chapter 11 focuses on inexact methods and preconditioning techniques that are the subject of current research. Many iterative methods depend in part on preconditioning to improve performance and ensure fast convergence.

It is inevitable that a large amount of important material needs to be excluded from this book. In the appendix, we list some of the subjects not covered in this book, giving references for the reader who is interested in pursuing a particular subject in depth.

This page intentionally left blank

Chapter 2

A Brief Tour of Eigenproblems

2.1 Introduction

Let A be a square n by n matrix, x a nonzero n by 1 vector (a column vector), and λ a scalar, such that

$$Ax = \lambda x. \quad (2.1)$$

Then λ is called an *eigenvalue* of A , and x is called a (*right*) *eigenvector*. Sometimes we refer to (λ, x) as an *eigenpair*. We refer the reader to the list of symbols and acronyms on page xxi; we will use the notation listed there freely throughout the text.

In this chapter we introduce and classify all the eigenproblems discussed in this book, describing their basic mathematical properties and interrelationships. Eigenproblems can be defined by a single square matrix A as in (2.1), by a nonsquare matrix A , by 2 or more square or rectangular matrices, or even by a matrix function of λ . We use the word “eigenproblem” in order to encompass computing eigenvalues, eigenvectors, Schur decompositions, condition numbers of eigenvalues, singular value and vectors, and yet other terms to be defined below. After reading this chapter the reader should be able to recognize the mathematical types of many eigenproblems, which are essential to picking the most effective algorithms. Not recognizing the right mathematical type of an eigenproblem can lead to using an algorithm that might not work at all or that might take orders of magnitude more time and space than a more specialized algorithm.

To illustrate the sources and interrelationships of these eigenproblems, we have a set of related examples for each one. The sections of this chapter are organized to correspond to the next six chapters of this book:

Section 2.2: Hermitian eigenproblems (HEP) (Chapter 4). This corresponds to $Ax = \lambda x$ as in (2.1), where A is Hermitian, i.e., $A = A^*$.

Section 2.3: Generalized Hermitian eigenproblems (GHEP) (Chapter 5). This corresponds to $Ax = \lambda Bx$, where A and B are Hermitian and B is positive definite (has all positive eigenvalues).

Section 2.4: Singular value decomposition (SVD) (Chapter 6). Given any rectangular matrix A , this corresponds to finding the eigenvalues and eigenvectors of the Hermitian matrices A^*A and AA^* .

Section 2.5: Non-Hermitian eigenproblems (NHEP) (Chapter 7). This corresponds to $Ax = \lambda x$ as in (2.1), where A is square but otherwise general.

Section 2.6: Generalized non-Hermitian eigenproblems (GNHEP) (Chapter 8). This corresponds to $Ax = \lambda Bx$. We will first treat the most common case of the *regular* generalized eigenproblem, which occurs when A and B are square and $\alpha A - \beta B$ is nonsingular for some choice of scalars α and β . We will also discuss the *singular case*.

Section 2.7: Nonlinear eigenproblems (Chapter 9). The simplest case of this is the *quadratic eigenvalue problem* $L(\lambda)x = (\lambda^2 M + \lambda D + K)x = 0$ and includes higher degree polynomials as well. We also discuss maximizing a real function $F(Y)$ over the space of m by n orthonormal matrices; this includes eigenproblems as a special case as well as much more complicated problems such as simultaneously reducing two or more symmetric matrices to diagonal form as nearly as possible using the same set of approximate eigenvectors for all of them.

All the eigenproblems described above arise naturally in applications arising in science and engineering. In each section we also show how one can recognize and solve closely related eigenproblems (for example, GEHPs, where $\alpha A + \beta B$ is positive definite instead of B). Chapters are presented in roughly increasing order of generality and complexity. For example, the HEP $Ax = \lambda x$ is clearly a special case of the GHEP $Ax = \lambda Bx$, because we can set $B = I$. It is also a special case of the NHEP, because we can ignore A 's Hermitian symmetry and treat it as a general matrix.

In general, the larger or more difficult an eigenvalue problem, the more important it is to use an algorithm that exploits as much of its mathematical structure as possible (such as symmetry or sparsity). For example, one can use algorithms for non-Hermitian problems to treat Hermitian ones, but the price is a large increase in time, storage, and possibly lower accuracy.

Each section from 2.2 through 2.6 is organized as follows.

1. The basic definitions of eigenvalues and eigenvectors will be given.
2. *Eigenspaces* will be defined. A *subspace* \mathcal{Y} is defined as the space $\text{span}\{y_1, \dots, y_k\}$ spanned by a chosen set of vectors y_1, \dots, y_k ; i.e., \mathcal{Y} is the set of all linear combinations of y_1, \dots, y_k . Eigenspaces are (typically) spanned by a subset of eigenvectors and may be called invariant subspaces, deflating subspaces, or something else depending on the type of eigenproblem.
3. *Equivalences* will be defined; these are transformations (such as changing A to SAS^{-1}) that leave the eigenvalues unchanged and can be used to compute a “simpler representation” of the eigenproblem. Depending on the situation, equivalences are also called *similarities* or *congruences*.
4. *Eigendecompositions* will be defined; these are commonly computed “simpler representations.”
5. *Conditioning* will be discussed. A *condition number* measures how sensitive the eigenvalues and eigenspaces of A are to small changes in A . These small changes could arise from roundoff or other unavoidable approximations made by the algorithm, or from uncertainty in the entries of A . One can get error bounds on

computed eigenvalues and eigenspaces by multiplying their condition numbers by a bound on the change in A . For more details on how condition numbers are used to get error bounds, see §2.1.1.

An eigenvalue or eigenspace is called *well-conditioned* if its error bound is acceptably small for the user (this obviously depends on the user), and *ill-conditioned* if it is much larger. Conditioning is important not just to interpret the computed results of an algorithm, but to choose the information to be computed. For example, different representations of the same eigenspace may have very different condition numbers, and it is often better to compute the better conditioned representation. Conditioning is discussed in more detail in each chapter, but the general results are summarized here.

6. Different ways of specifying an eigenproblem are listed. The most expensive eigenvalue problem is to ask for all eigenvalues and eigenvectors of A . Since this is often too expensive in time and space, users frequently ask for less information, such as the largest 10 eigenvalues and perhaps their eigenvectors. (Note that if A is sparse, typically the eigenvectors are dense, so storing all the eigenvectors can take much more memory than storing A .) Also, some eigenproblems for the same matrix A may be much better conditioned than others, and these may be preferable to compute.
7. Related eigenproblems are discussed. For example, if it is possible to convert an eigenproblem into a simpler and cheaper special case, this is shown.
8. The vibrational analysis of the mass-spring system shown in Figure 2.1 is used to illustrate the source and formulation of each eigenproblem.

Newton's law $F = ma$ applied to this vibrating mass-spring system yields

$$k_i(x_{i-1}(t) - x_i(t)) + k_{i+1}(x_{i+1}(t) - x_i(t)) - b_i\dot{x}_i(t) = m_i\ddot{x}_i(t),$$

where the first term on the left-hand side is the force on mass i from spring i , the second term is the force on mass i from spring $i + 1$, and the third term is the force on mass i from damper i . In matrix form, these equations can be written as

$$M\ddot{x}(t) = -B\dot{x}(t) - Kx(t),$$

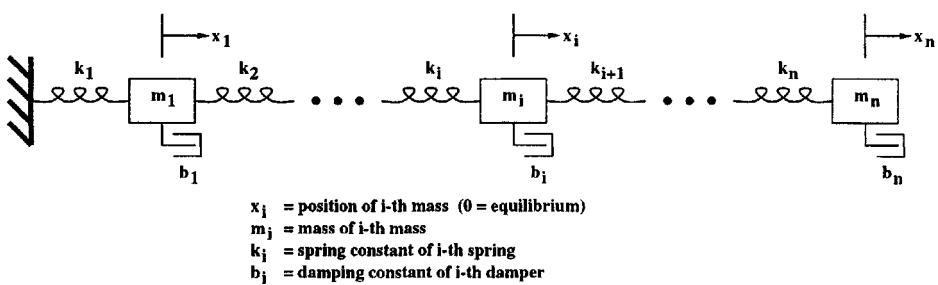


Figure 2.1: Damped, vibrating mass-spring system.

where $M = \text{diag}(m_1, \dots, m_n)$, $B = \text{diag}(b_1, \dots, b_n)$, and

$$K = \begin{bmatrix} k_1 + k_2 & -k_2 & & \\ -k_2 & k_2 + k_3 & -k_3 & \\ & \ddots & \ddots & \ddots \\ & & -k_{n-1} & k_{n-1} + k_n & -k_n \\ & & & -k_n & k_n \end{bmatrix}.$$

We assume all the masses m_i are positive. M is called the *mass matrix*, B is the *damping matrix*, and K is the *stiffness matrix*. All three matrices are symmetric. They are also positive definite (have all positive eigenvalues) when the m_i , b_i , and k_i are positive, respectively. This differential equation becomes an eigenvalue problem by seeking solutions of the form $x(t) = e^{\lambda t}x$, where λ is a constant scalar and x is a constant vector, both of which are determined by solving appropriate eigenproblems.

Electrical engineers analyzing linear circuits arrive at an analogous equation by applying Kirchoff's and related laws instead of Newton's law. In this case x represents branch currents, M represent inductances, B represents resistances, and K represents admittances (reciprocal capacitances).

Chapter 9 on nonlinear eigenproblems is organized differently, according to the structure of the specific nonlinear problems discussed.

Finally, Chapters 10 and 11 treat issues common to many or all of the above eigenvalue problems. Chapter 10 treats data structures, algorithms, and software for sparse matrices, especially sparse linear solvers, which often are the most time-consuming part of an eigenvalue algorithm. Chapter 11 treats preconditioning techniques or methods for converting an eigenproblem into a simpler one. Some preconditioning techniques are well established; others are a matter of current research.

2.1.1 Numerical Stability and Conditioning

Here we elaborate on how condition numbers are used to get error bounds. For simplicity, suppose we are just trying to evaluate a scalar function $f(x)$, using an algorithm $alg(x)$, and we want to bound the error $alg(x) - f(x)$. Further, suppose that we can show that $alg(x) = f(x + e)$ for some “small” e . If the algorithm enjoys this property, that it gets exactly the right answer $f(x + e)$ for a slightly perturbed argument $x + e$, then the algorithm is called *numerically stable*, or just *stable* for short. In this case we can estimate the error as follows:

$$\text{error} = alg(x) - f(x) = f(x + e) - f(x) \approx f'(x) \cdot e,$$

where we have assumed that $f(x)$ is differentiable. In this simple case we can therefore bound $|\text{error}| \leq |f'(x)| \cdot |e|$, where $|f'(x)|$ is the *condition number* and e is the so-called *backward error*. Note that $f'(x)$ depends only on the function $f()$ and its argument x , whereas e depends on the algorithm as well.

The same approach applies to getting error bounds for eigenproblems. In this case x would be a matrix, $x + e$ would be a perturbed matrix, and $f(x + e)$ could be an eigenvalue, a set of eigenvalues, a set of eigenvalue/eigenvector pairs, or other quantities. Furthermore $|e|$ would be replaced by a norm of the matrix e .

So to get an error bound, we need both a condition number and a bound on the backward error e . There are two ways to get a bound on e . The first way is to prove that the algorithm $\text{alg}(x)$ *always* has a small backward error e . For the transformation methods discussed in this book, this is the case, with the norm of the backward error matrix e roughly bounded by machine precision ϵ_M times the norm of the input matrix x . This kind of guaranteed stability does *not* hold for iterative methods.

For iterative methods, the story is more complicated. Here is a sketch, with the details left to Chapters 4 through 9. Suppose μ and unit vector x form an approximate eigenvalue and eigenvector pair, that is, that the residual vector $Ax - \mu x \equiv r$ is small. Then it is easy to show that a smallest matrix E such that $(A + E)x = \mu x$, i.e., where μ and x are an exact eigenvalue/eigenvector pair of $A + E$, is $E = -rx^*$, with norm $\|E\|_2 = \|r\|_2$. So to bound the backward error all we have to do is compute the residual 2-norm $\|r\|_2 = \|Ax - \mu x\|_2$. So bounding the backward error for a single eigenvalue and eigenvector pair is very easy. The story is more complicated for a set of eigenvalue/eigenvector pairs $Ax_i \approx \mu_i x_i$, $i = 1, \dots, m$. Even if each $Ax_i - \mu_i x_i$ is small, that does *not* mean that the μ_i, x_i are approximations of m different eigenpairs (some may approximate the same true eigenpair), or that the eigenvectors are orthogonal when they are supposed to be (when A is Hermitian), or any other desirable property. Some iterative methods are more effective than others at guaranteeing such properties (for examples, see implicitly restarted Lanczos and Arnoldi methods described in §4.5 and §7.6).

The reader is referred to the textbooks by Golub and Van Loan [198] and Demmel [114] for an introductory study on numerical stability. The books by Higham [228] and Stewart and Sun [425] are excellent sources for the advanced study of numerical stability and conditioning for solving linear system of equations and eigenvalue problems, respectively.

2.2 Hermitian Eigenproblems

J. Demmel

We assume that A is an n by n Hermitian matrix, i.e., $A^* = A$.

2.2.1 Eigenvalues and Eigenvectors

The polynomial $p(\lambda) = \det(\lambda I - A)$ is called the *characteristic polynomial* of A . The roots of $p(\lambda) = 0$ are called the *eigenvalues* of A . Since the degree of $p(\lambda)$ is n , it has n roots, and so A has n eigenvalues.

A nonzero vector x satisfying $Ax = \lambda x$ is a (*right*) *eigenvector* for the eigenvalue λ . Since $x^* A = \lambda x^*$, left and right eigenvectors are identical.

All eigenvalues of a Hermitian matrix A are real. This lets us write them in sorted order, $\lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$. If all $\lambda_i > 0$, then A is called *positive definite*, and if all $\lambda_i \geq 0$, then A is called *positive semidefinite*. *Negative definite* and *negative semidefinite* are defined analogously. If there are both positive and negative eigenvalues, A is called *indefinite*.

Each eigenvalue λ_i has an eigenvector x_i . We may assume without loss of generality that $\|x_i\|_2 = 1$. Each x_i is real if A is real. Though the x_i may not be unique (e.g., any vector is an eigenvector of the identity matrix), they may be chosen to all be orthogonal

to one another: $x_i^* x_j = 0$ if $i \neq j$. When an eigenvalue is distinct from all the other eigenvalues, its eigenvector is unique (up to multiplication by scalars).

2.2.2 Invariant Subspaces

An *invariant subspace* \mathcal{X} of A satisfies $Ax \in \mathcal{X}$ for all $x \in \mathcal{X}$. We also write this as $A\mathcal{X} \subset \mathcal{X}$. The simplest example is when \mathcal{X} is spanned by a single eigenvector of A . More generally any invariant subspace can be spanned by a subset of the eigenvectors of A , although the vectors spanning \mathcal{X} do not have to be eigenvectors themselves.

2.2.3 Equivalences (Similarities)

Suppose Q is a *unitary matrix*, i.e., $Q^{-1} = Q^*$. If Q is real then $Q^{-1} = Q^T$ and we say that Q is an *orthogonal matrix*. Let $B = Q^* A Q$. We say that B is *unitarily (orthogonally) similar* to A , and that Q is a *unitary (orthogonal) similarity transformation*. If A is Hermitian, so is B , and it has the same eigenvalues. The similarity transformation corresponds to introducing a new basis with the columns of Q as vectors. If y is an eigenvector of the transformed matrix B , then $x = Qy$ is an eigenvector of the original matrix A .

2.2.4 Eigendecompositions

Define $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_n)$ and $X = [x_1, \dots, x_n]$. X is called an *eigenvector matrix* of A . Since the x_i are orthogonal unit vectors, we see that $X^* X = I$; i.e., X is a *unitary (orthogonal) matrix*. The n equalities $Ax_i = \lambda_i x_i$ for $i = 1, \dots, n$ may also be written $AX = X\Lambda$ or $A = X\Lambda X^*$. The factorization

$$A = X\Lambda X^*$$

is called the *eigendecomposition* of A . In other words, A is similar to the diagonal matrix Λ , with similarity transformation X .

If we take a subset of k columns of X (say $\hat{X} = X(:, [2, 3, 5]) = \text{columns } 2, 3, \text{ and } 5$), then these columns span an invariant subspace of A . If we take the corresponding submatrix $\hat{\Lambda} = \text{diag}(\lambda_2, \lambda_3, \lambda_5)$ of Λ , then we can write the corresponding *partial eigendecomposition* as $A\hat{X} = \hat{X}\hat{\Lambda}$ or $\hat{X}^* A \hat{X} = \hat{\Lambda}$. If the columns in \hat{X} are replaced by k different vectors spanning the same invariant subspace, then we get a different partial eigendecomposition $A\check{X} = \check{X}\check{\Lambda}$, where $\check{\Lambda}$ is a k -by- k matrix whose eigenvalues are those of $\hat{\Lambda}$, though $\check{\Lambda}$ may not be diagonal.

2.2.5 Conditioning

The eigenvalues of A are always well-conditioned, in the sense that changing A in norm by at most ϵ can change any eigenvalue by at most ϵ . We refer to §4.8 for technical definitions.

This is adequate for most purposes, unless the user is interested in the leading digits of a small eigenvalue, one less than or equal to ϵ in magnitude. For example, computing $\lambda_i = 10^{-5}$ to within plus or minus $\epsilon = 10^{-4}$ means that no leading digits of the computed λ_i may be correct. See [114, 118] for a discussion of the sensitivity of small eigenvalues and of when their leading digits may be computed accurately.

Eigenvectors and eigenspaces, on the other hand, can be ill-conditioned. For example, changing

$$A_0 = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & \epsilon \\ 0 & \epsilon & 1 \end{bmatrix} \quad \text{to} \quad A_1 = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 + \epsilon \end{bmatrix}$$

rotates the two eigenvectors corresponding to the two eigenvalues near 1 by $\pi/4$, no matter how small ϵ is. Thus they are very sensitive to small changes. The condition number of an eigenvector depends on the *gap* between its eigenvalue and the closest other eigenvalue: the smaller the gap the more sensitive the eigenvectors. In this example the two eigenvalues near 1 are very close, so their gaps are small and their eigenvectors are sensitive. But the two-dimensional invariant subspace they span is very insensitive to changes in A (because their eigenvalues, both near 1, are very far from the next closest eigenvalue, at 2). So when eigenvectors corresponding to a cluster of close eigenvalues are too ill-conditioned, the user may want to compute a basis of the invariant subspace they span instead of individual eigenvectors.

2.2.6 Specifying an Eigenproblem

The following eigenproblems are typical, both because they arise naturally in applications and because we have good algorithms for them:

1. Compute all the eigenvalues to some specified accuracy.
2. Compute eigenvalues λ_i for some specified set of subscripts $i \in \mathcal{I} = \{1, 2, \dots, n\}$, including the special cases of the largest m eigenvalues λ_{n-m+1} through λ_n , and the smallest m eigenvalues λ_1 through λ_m . Again, the desired accuracy may be specified.
3. Compute all the eigenvalues within a given subset of the real axis, such as the interval $[\alpha, \beta]$. Again, the desired accuracy may be specified.
4. Count all the eigenvalues in the interval $[\alpha, \beta]$. This does not require computing the eigenvalues in $[\alpha, \beta]$, and so can be much cheaper.
5. Compute a certain number of eigenvalues closest to a given value μ .

For each of these possibilities (except 4) the user can also compute the corresponding eigenvectors. For the eigenvalues that are clustered together, the user may choose to compute the associated invariant subspace, since in this case the individual eigenvectors can be very ill-conditioned, while the invariant subspace may be less so. Finally, for any of these quantities, the user might also want to compute its condition number.

Even though we have effective algorithms for these problems, we cannot necessarily solve all large scale problems in an amount of time and space acceptable to all users.

2.2.7 Related Eigenproblems

Since the HEP is one of the best understood eigenproblems, it is helpful to recognize when other eigenproblems can be converted to it.

1. If A is non-Hermitian, but $\widehat{A} = \alpha S A S^{-1}$ is Hermitian for easily determined α and S , it may be advisable to compute the eigenvalues $\widehat{\lambda}$ and eigenvectors \widehat{x} of \widehat{A} . One can convert these to eigenvalues λ and eigenvectors x of A via $\lambda = \widehat{\lambda}/\alpha$ and $x = S^{-1}\widehat{x}$. For example, multiplying a skew-Hermitian matrix A (i.e., $A^* = -A$) by the constant $\sqrt{-1}$ makes it Hermitian. See §2.5 for further discussion.
2. If $A = B^*B$ for some rectangular matrix B , then the eigenproblem for A is equivalent to the SVD of B , discussed in §2.4. Suppose B is m by n , so A is n by n . Generally speaking, if A is about as small or smaller than B ($n \leq m$, or just a little bigger), the eigenproblem for A is usually cheaper than the SVD of B . But it may be less accurate to compute the small eigenvalues of A than the small singular values of B . See §2.4 for further discussion.
3. If one has the generalized HEP $Ax = \lambda Bx$, where A and B are Hermitian and B is positive definite, it can be converted to a Hermitian eigenproblem as follows. First, factor $B = LL^*$, where L is any nonsingular matrix (this is typically done using Cholesky factorization). Then solve the HEP for $\widehat{A} = L^{-1}AL^{-*}$. The eigenvalues of \widehat{A} and $A - \lambda B$ are identical, and if \widehat{x} is an eigenvector of \widehat{A} , then $x = L^{-*}\widehat{x}$ satisfies $Ax = \lambda Bx$. See §2.3 for further discussion.

2.2.8 Example

For the vibrating mass-spring system introduced in §2.1 and Figure 2.1, we assume that

1. all masses $m_i = 1$, so $M = I$, and
2. all damping constants $b_i = 0$, so $B = 0$.

This implies the equations of motion to $\ddot{x}(t) = -Kx(t)$. We solve them by substituting $x(t) = e^{\lambda t}x$, where x is a constant vector and λ is a constant scalar to be determined. This yields

$$Kx = -\lambda^2x.$$

Thus x is an eigenvector and $-\lambda^2$ is an eigenvalue of the symmetric positive definite tridiagonal matrix K . Thus λ is pure imaginary and we get that $x(t)$ is periodic with period $2\pi/|\lambda|$. Symmetric tridiagonal matrices have particularly fast and efficient eigenvalue algorithms.

Later sections deal with the cases of nonunit masses m_i and nonzero damping constants b_i .

2.3 Generalized Hermitian Eigenproblems

J. Demmel

We assume that A and B are n by n Hermitian matrices and that B is positive definite. We call $A - \lambda B$ a *definite matrix pencil*, or *definite pencil* for short. Here λ is a variable

rather than a constant.¹ For convenience we will refer to eigenvalues, eigenvectors, and other properties of the pencil $A - \lambda B$.

2.3.1 Eigenvalues and Eigenvectors

The polynomial $p(\lambda) = \det(\lambda B - A)$ is called the *characteristic polynomial* of $A - \lambda B$. The roots of $p(\lambda) = 0$ are *eigenvalues* of $A - \lambda B$. Since the degree of $p(\lambda)$ is n , it has n roots, and so $A - \lambda B$ has n eigenvalues.

A nonzero vector x satisfying $Ax = \lambda Bx$ is a (*right*) *eigenvector* for the eigenvalue λ . The eigenpair (λ, x) also satisfies $x^* A = \lambda x^* B$, so we can also call x a *left eigenvector*.

All eigenvalues of the definite pencil $A - \lambda B$ are real. This lets us write them in sorted order $\lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$. If all $\lambda_i > 0$, then $A - \lambda B$ is called *positive definite*, and if all $\lambda_i \geq 0$, then $A - \lambda B$ is called *positive semidefinite*. *Negative definite* and *negative semidefinite* are defined analogously. If there are both positive and negative eigenvalues, $A - \lambda B$ is called *indefinite*.

Each x_i is real if A and B are real. Though the x_i may not be unique, they may be chosen to all be B *orthogonal* to one another: $x_i^* B x_j = 0$ if $i \neq j$. This is also called *orthogonality with respect to the inner product induced by the Hermitian positive definite matrix B* . When an eigenvalue is distinct from all the other eigenvalues, its eigenvector is unique (up to multiplication by scalars).

2.3.2 Eigenspaces

An *eigenspace* \mathcal{X} of $A - \lambda B$ satisfies $B^{-1}Ax \in \mathcal{X}$ for all $x \in \mathcal{X}$. We also write this as $B^{-1}A\mathcal{X} \subset \mathcal{X}$, or $A\mathcal{X} \subset B\mathcal{X}$. The simplest example is when \mathcal{X} is spanned by a single eigenvector of $A - \lambda B$. More generally, an eigenspace can be spanned by a subset of eigenvectors of $A - \lambda B$, although the vectors spanning \mathcal{X} do not have to be eigenvectors themselves.

2.3.3 Equivalences (Congruences)

Suppose X is a nonsingular matrix. Let $\widehat{A} = X^*AX$ and $\widehat{B} = X^*BX$. We say that the pencil $\widehat{A} - \lambda \widehat{B}$ is *congruent* to $A - \lambda B$, and that X is a *congruence transformation*. If A and B are Hermitian, with B positive definite, than \widehat{A} and \widehat{B} have these same properties. Furthermore, $\widehat{A} - \lambda \widehat{B}$ and $A - \lambda B$ have the same eigenvalues, and if x is an eigenvector of $A - \lambda B$, so that $Ax = \lambda Bx$, then $\widehat{x} = X^{-1}x$ is an eigenvector of $\widehat{A} - \lambda \widehat{B}$.

2.3.4 Eigendecompositions

Define $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_n)$ and $X = [x_1, \dots, x_n]$. X is called an *eigenvector matrix* of $A - \lambda B$. Since the x_i are unit vectors orthogonal with respect to the inner product induced by B , we see that $X^*BX = \Lambda_B$, a nonsingular diagonal matrix. The n equalities $Ax_i = \lambda_i Bx_i$ for $i = 1, \dots, n$ may also be written $AX = BX\Lambda$ or $X^*AX = X^*BX\Lambda = \Lambda_B\Lambda$. Thus $X^*AX \equiv \Lambda_A$ is diagonal too. The factorizations

$$X^*AX = \Lambda_A \quad \text{and} \quad X^*BX = \Lambda_B$$

¹The word pencil arises historically, because the set of all matrices of the form $A - \lambda B$ for constant A , constant B , and varying λ forms a “line” of matrices in matrix space, and this resembles a “pencil” or beam of light.

(or $A = X^{-*}\Lambda_AX^{-1}$ and $B = X^{-*}\Lambda_BX^{-1}$) are called an *eigendecomposition* of $A - \lambda B$. In other words, $A - \lambda B$ is congruent to the diagonal pencil $\Lambda_A - \lambda\Lambda_B$, with congruence transformation X .

If we take a subset of k columns of X (say $\hat{X} = X(:, [2, 3, 5])$ = columns 2, 3, and 5), then these columns span an eigenspace of $A - \lambda B$. If we take the corresponding submatrix $\hat{\Lambda}_A = \text{diag}(\hat{\Lambda}_{A,22}, \hat{\Lambda}_{A,33}, \hat{\Lambda}_{A,55})$ of Λ_A , and similarly define $\hat{\Lambda}_B$, then we can write the corresponding *partial eigendecomposition* as $\hat{X}^*A\hat{X} = \hat{\Lambda}_A$ and $\hat{X}^*B\hat{X} = \hat{\Lambda}_B$. If the columns in \hat{X} are replaced by k different vectors spanning the same eigensubspace, then we get a different partial eigendecomposition, where $\hat{\Lambda}_A$ and $\hat{\Lambda}_B$ are replaced by different k -by- k matrices $\check{\Lambda}_A$ and $\check{\Lambda}_B$ such that the eigenvalues of the pencil $\check{\Lambda}_A - \lambda\check{\Lambda}_B$ are those of $\hat{\Lambda}_A - \lambda\hat{\Lambda}_B$, though the pencil $\check{\Lambda}_A - \lambda\check{\Lambda}_B$ may not be diagonal.

2.3.5 Conditioning

An eigenvalue λ_i of $A - \lambda B$ may be well-conditioned or ill-conditioned. If $B = I$ (or is close), the eigenvalues are as well-conditioned (or close) as for the Hermitian eigenproblem described in §2.2.5. But if $|x_i^*Bx_i|$ is very small, where x_i is a unit eigenvector of λ_i , then λ_i can be very ill conditioned. For example, changing

$$A_0 - \lambda B_0 = \begin{bmatrix} 2 & 0 \\ 0 & 10^{-6} \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 \\ 0 & 10^{-6} \end{bmatrix}$$

to

$$A_1 - \lambda B_1 = \begin{bmatrix} 2 & 0 \\ 0 & 10^{-6} + \epsilon \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 \\ 0 & 10^{-6} \end{bmatrix}$$

changes an eigenvalue from $1 = 10^{-6}/10^{-6}$ to $(10^{-6} + \epsilon)/10^{-6} = 1 + 10^6\epsilon$; i.e., the ϵ change is magnified by 10^6 .

Eigenvectors and eigenspaces can also be ill-conditioned for the same reason as in §2.2.5: if the *gap* between an eigenvalue and the closest other eigenvalue is small, then its eigenvector will be ill-conditioned. Even if the gap is large, if the eigenvalue is ill-conditioned as in the above example, the eigenvector can also be ill-conditioned. Again, an eigenspace spanned by the eigenvectors of a cluster of eigenvalues may be much better conditioned than the individual eigenvectors. We refer to §5.7 for further details.

2.3.6 Specifying an Eigenproblem

This section is identical to the corresponding §2.2.6 for the Hermitian eigenproblem.

1. Compute all the eigenvalues to some specified accuracy.
2. Compute eigenvalues λ_i for some specified set of subscripts $i \in \mathcal{I} = \{1, 2, \dots, n\}$, including the special cases of the largest m eigenvalues λ_{n-m+1} through λ_n , and the smallest m eigenvalues λ_1 through λ_m . Again, the desired accuracy may be specified.
3. Compute all the eigenvalues within a given subset of the real axis, such as the interval $[\alpha, \beta]$. Again, the desired accuracy may be specified.

4. Count all the eigenvalues in the interval $[\alpha, \beta]$. This does not require computing the eigenvalues in $[\alpha, \beta]$, and so can be much cheaper.
5. Compute a certain number of eigenvalues closest to a given value μ .

For each of these possibilities (except 4) the user can also compute the corresponding eigenvectors. For the eigenvalues that are clustered together, the user may choose to compute the associated eigenspace, since in this case the individual eigenvectors can be very ill-conditioned, while the eigenspace may be less so. Finally, for any of these quantities, the user might also want to compute its condition number.

Even though we have effective algorithms for these problems, we cannot necessarily solve all large scale problems in an amount of time and space acceptable to all users.

2.3.7 Related Eigenproblems

1. If A and B are Hermitian, B is not positive definite, but $\alpha A + \beta B$ is positive definite for some choice of real numbers α and β , one can solve the generalized Hermitian eigenproblem $A - \lambda(\alpha A + \beta B)$ instead. Let $\widehat{B} = \alpha A + \beta B$; then the eigenvectors of $A - \lambda B$ and $A - \lambda \widehat{B}$ are identical. The eigenvalues λ_i of $A - \lambda B$ and the eigenvalues $\widehat{\lambda}_i$ of $A - \lambda \widehat{B}$ are related by $\lambda_i = \beta \widehat{\lambda}_i / (1 - \alpha \widehat{\lambda}_i)$.
2. If A and B are non-Hermitian, but $\widehat{A} = \alpha SAT$ and $\widehat{B} = \beta SBT$ are Hermitian, with \widehat{B} positive definite, for easily determined α , β and nonsingular S and T , then one can compute the eigenvalues $\widehat{\lambda}$ and eigenvectors \widehat{x} of $\widehat{A} - \lambda \widehat{B}$. One can convert these to eigenvalues λ and eigenvectors x of A via $\lambda = \widehat{\lambda} \beta / \alpha$ and $x = T\widehat{x}$. For example, if B is Hermitian positive definite but A is skew-Hermitian (i.e., $A^* = -A$), then $\sqrt{-1}A$ is Hermitian, so we may choose $\alpha = \sqrt{-1}$, $\beta = 1$, and $S = T = I$. See §2.5 for further discussion.
3. If one has the GHEP $Ax = \lambda Bx$, where A and B are Hermitian and B is positive definite, then it can be converted to a HEP as follows. First, factor $B = LL^*$, where L is any nonsingular matrix (this is typically done using Cholesky factorization). Then solve the HEP for $\widehat{A} = L^{-1}AL^{-*}$. The eigenvalues of \widehat{A} and $A - \lambda B$ are identical, and if \widehat{x} is an eigenvector of \widehat{A} , then $x = L^{-*}\widehat{x}$ satisfies $Ax = \lambda Bx$. Indeed, this is a standard algorithm for $A - \lambda B$.
4. If A and B are positive definite with $A = R^*R$ and $B = U^*U$ for some rectangular matrices R and U , then the eigenproblem for $A - \lambda B$ is equivalent to the *quotient singular value decomposition (QSVD)* of R and U , discussed in §2.4. The state of algorithms is such that it is probably better to try solving the eigenproblem for $A - \lambda B$ than computing the QSVD of R and U .

2.3.8 Example

We continue to use the example introduced in §2.1 and Figure 2.1. We now consider the case where there are arbitrary positive masses $m_i > 0$, but the damping constants b_i are zero. This simplifies the equations of motion to $M\ddot{x}(t) = -Kx(t)$. We again solve them by substituting $x(t) = e^{\lambda t}x$, where x is a constant vector and λ is a constant scalar to be determined. This yields

$$Kx = -\lambda^2 Mx.$$

Thus x is an eigenvector and $-\lambda^2$ is an eigenvalue of the generalized Hermitian eigenproblem $Kx = \mu Mx$. Since K and M are positive definite, the eigenvalues $-\lambda^2$ are positive, so λ is pure imaginary and we find that $x(t)$ is periodic with period $2\pi/|\lambda|$.

Following item 3 in §2.3.7, we may convert this to a standard Hermitian eigenvalue problem as follows. Let $M = LL^T$ be the Cholesky decomposition of M . Thus L is simply the diagonal matrix $\text{diag}(m_1^{1/2}, \dots, m_n^{1/2})$. Then the eigenvalues of $Kx = \mu Mx$ are the same as the eigenvalues of the symmetric tridiagonal matrix

$$\hat{K} = L^{-1}KL^{-T} = \begin{bmatrix} \frac{k_1+k_2}{m_1} & \frac{-k_2}{\sqrt{m_1m_2}} & & & \\ \frac{-k_2}{\sqrt{m_1m_2}} & \frac{k_2+k_3}{m_2} & \frac{-k_3}{\sqrt{m_2m_3}} & & \\ & \ddots & \ddots & \ddots & \\ & & \frac{-k_{n-1}}{\sqrt{m_{n-2}m_{n-1}}} & \frac{k_{n-1}+k_n}{m_{n-1}} & \frac{-k_n}{\sqrt{m_{n-1}m_n}} \\ & & & \frac{-k_n}{\sqrt{m_{n-1}m_n}} & \frac{k_n}{m_n} \end{bmatrix}. \quad (2.2)$$

Symmetric tridiagonal matrices have particularly fast and efficient eigenvalue algorithms.

2.4 Singular Value Decomposition

J. Demmel

We primarily consider the SVD of a single m -by- n matrix A and discuss related SVDs in §2.4.7. We assume without loss of generality that $m \geq n$. If $m < n$, consider A^* .

2.4.1 Singular Values and Singular Vectors

The square roots of the n eigenvalues of A^*A are the *singular values* of A . Since A^*A is Hermitian and positive semidefinite, the singular values are real and nonnegative. This lets us write them in sorted order $0 \leq \sigma_n \leq \dots \leq \sigma_1$.

The n eigenvectors of A^*A are called *(right) singular vectors*. We denote them by v_1, \dots, v_n , where v_i is the eigenvector for eigenvalue σ_i^2 . The m by m matrix AA^* is also Hermitian positive semidefinite. Its largest n eigenvalues are identical to those of AA^* , and the rest are zero. The m eigenvectors of AA^* are called *(left) singular vectors*. We denote them by u_1, \dots, u_m , where u_1 through u_n are eigenvectors for eigenvalues σ_1^2 through σ_n^2 , and u_{n+1} through u_m are eigenvectors for the zero eigenvalue. The singular vectors can be chosen to satisfy the identities $Av_i = \sigma_i u_i$ and $A^*v_i = \sigma_i v_i$ for $i = 1, \dots, n$, and $A^*u_i = 0$ for $i = n+1, \dots, m$.

We may assume without loss of generality that each $\|u_i\|_2 = 1$ and $\|v_i\|_2 = 1$. The singular vectors are real if A is real. Though the singular vectors may not be unique (e.g., any vector is a singular vector of the identity matrix), they may all be chosen to be orthogonal to one another: $u_i^* u_j = v_i^* v_j = 0$ if $i \neq j$. When a singular value is distinct from all the other singular values, its singular vectors are unique (up to multiplication by scalars).

2.4.2 Singular Subspaces

A pair of k -dimensional subspaces \mathcal{U} and \mathcal{V} are called (*left and right*) *singular subspaces* of A if $Av \in \mathcal{U}$ for all $v \in \mathcal{V}$ and $A^*u \in \mathcal{V}$ for all $u \in \mathcal{U}$. We also write this as $AV \subset \mathcal{U}$ and $A^*\mathcal{U} \subset \mathcal{V}$.

The simplest example is when \mathcal{U} and \mathcal{V} are spanned by a single pair of singular vectors u_i and v_i of A , respectively. More generally, any pair of singular subspaces can be spanned by a subset of the singular vectors of A , although the spanning vectors do not have to be singular vectors themselves.

2.4.3 Equivalences

Suppose Q and X are *unitary matrices*, i.e., $Q^{-1} = Q^*$ and $X^{-1} = X^*$. If Q and X are real, then $Q^{-1} = Q^T$ and $X^{-1} = X^T$, and we call them *orthogonal matrices*. Let $B = QAX^*$. We say that B is *unitarily (orthogonally) equivalent* to A and that Q and X are *unitary (orthogonal) equivalence transformations*. B has the same singular values as A . If u and v are left and right singular vectors of A , respectively, so that $Av = \sigma u$ and $A^*u = \sigma v$, then Qu and Xv are left and right singular vectors of B , respectively.

2.4.4 Decompositions

Define Σ as the m by n matrix whose top n rows contain $\text{diag}(\sigma_1, \dots, \sigma_n)$ and whose bottom $m - n$ rows are zero. Define the m by m matrix $U = [u_1, \dots, u_m]$ and the n by n matrix $V = [v_1, \dots, v_n]$. U is called the *left singular vector matrix* of A , and V is called the *right singular vector matrix* of A . Since the u_i are orthogonal unit vectors, we see that $U^*U = I$; i.e., U is a *unitary matrix*. If A is real then the u_i are real vectors, so $U^*U = I$, and we also say that U is an *orthogonal matrix*. The same discussion applies to V . The $n + m$ equalities $Av_i = \sigma_i u_i$ and $A^*u_i = \sigma_i v_i$ for $i = 1, \dots, n$ and $A^*u_i = 0$ for $i = n + 1, \dots, m$ may also be written $AV = U\Sigma$ and $A^*U = V\Sigma^*$, or $A = U\Sigma V^*$. The factorization

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

is called the *SVD* of A . In other words, A is unitarily (orthogonally) equivalent to the diagonal matrix Σ .

There are several “smaller” versions of the SVD that are often computed. Let $U_t = [u_1, \dots, u_t]$ be an m by t matrix of the first t left singular vectors, $V_t = [v_1, \dots, v_t]$ be an n by t matrix of the first t right singular vectors, and $\Sigma_t = \text{diag}(\sigma_1, \dots, \sigma_t)$ be a t by t matrix of the first t singular values. Then we can make the following definitions.

Thin SVD. $A = U_n \Sigma_n V_n^*$ is the *thin (or economy-sized) SVD* of A . The thin SVD is much smaller to store and faster to compute than the full SVD when $n \ll m$.

Compact SVD. $A = U_r \Sigma_r V_r^*$ is the *compact SVD* of A . The compact SVD is much smaller to store and faster to compute than the thin SVD when $r \ll n$.

Truncated SVD. $A_t = U_t \Sigma_t V_t^*$ is the *rank- t truncated (or partial) SVD* of A , where $t < r$. Among all rank- t matrices B , $B = A_t$ is the unique minimizer of $\|A - B\|_F$ and also minimizes (perhaps not uniquely) $\|A - B\|_2$. The truncated SVD is much smaller to store and cheaper to compute than the compact SVD when $t \ll r$, and is the most common form of the SVD computed in applications.

The thin SVD may also be written $A = \sum_{i=1}^n \sigma_i u_i v_i^*$. Each (σ_i, u_i, v_i) is called a *singular triplet*. The compact and truncated SVDs may be written similarly (the sum going from $i = 1$ to r , or $i = 1$ to t , respectively).

If A is m by n with $m < n$, then its SVD is $A = U\Sigma V^*$, where U is m by m , Σ is m by n with $\text{diag}(\sigma_1, \dots, \sigma_m)$ in its first m columns and zeros in columns $m+1$ through n , and V is n by n . Its thin SVD is $A = U_m \Sigma_m V_m^*$, and the compact SVD and truncated SVD are as above.

More generally, if we take a subset of k columns of U and V (say $\widehat{U} = U(:, [2, 3, 5])$ = columns 2, 3, and 5, and $\widehat{V} = V(:, [2, 3, 5])$), then these columns span a pair of singular subspaces of A . If we take the corresponding submatrix $\widehat{\Sigma} = \text{diag}(\sigma_2, \sigma_3, \sigma_5)$ of Σ , then we can write the corresponding *partial SVD* $\widehat{U}^* A \widehat{V} = \widehat{\Sigma}$. If the columns in \widehat{U} and \widehat{V} are replaced by k different orthonormal vectors spanning the same invariant subspace, then we get a different partial SVD $\check{U}^* A \check{V} = \check{\Lambda}$, where $\check{\Lambda}$ is a k by k matrix whose singular values are those of $\widehat{\Sigma}$, though $\check{\Lambda}$ may not be diagonal.

2.4.5 Conditioning

The singular values and singular vectors of A have largely the same conditioning properties as the eigenvalues and eigenvectors of a Hermitian matrix, as described in §2.2.5.

The singular values of A are always well-conditioned, in the sense that changing A in norm by at most ϵ can change any eigenvalue by at most ϵ .

This is adequate for most purposes, unless the user is interested in the leading digits of a small singular value, one less than or equal to ϵ in magnitude. For example, computing $\sigma_i = 10^{-5}$ to within plus or minus $\epsilon = 10^{-4}$ means that no leading digits of the computed σ_i may be correct. See [114, 118] on the discussion of the sensitivity of small singular values and of when their leading digits may be computed accurately.

Singular vectors and singular subspaces, on the other hand, can be ill-conditioned. The example in §2.2.5 illustrates this point. (The eigenvalues and eigenvectors of that matrix are identical to its singular values and singular vectors.) Thus, the condition number of a singular vector depends on the *gap* between its singular value and the closest other singular value; the smaller the gap, the more sensitive the singular vector. When singular vectors corresponding to a cluster of close singular values are too ill-conditioned, the user may want to compute a basis of the singular subspace they span instead of individual singular vectors.

2.4.6 Specifying a Singular Value Problem

The following problems are typical, both because they arise naturally in applications and because we have good algorithms for them. They correspond to the problems in §2.2.6.

1. Compute all the singular values to some specified accuracy.
2. Compute singular values σ_i for some specified set of subscripts $i \in \mathcal{I} = \{1, 2, \dots, n\}$ including the special cases of the smallest r singular values σ_{n-r+1} through σ_n , and the largest r singular values σ_1 through σ_r . Again, the desired accuracy may be specified.

3. Compute all the singular values within a given subset of the real axis, such as the interval $[\alpha, \beta]$. Again, the desired accuracy may be specified.
4. Count all the singular values in the interval $[\alpha, \beta]$. This does not require computing the singular values in $[\alpha, \beta]$, and so can be much cheaper.
5. Compute a certain number of singular values closest to a given value μ .

For each of these possibilities (except 4) the user can also compute the corresponding singular vectors (left, right, or both). For the singular values that are clustered together, the user may choose to compute the associated singular subspace(s), since in this case the individual singular vectors can be very ill-conditioned, while the singular subspace(s) may be less so. Finally, for any of these quantities, the user might also want to compute its condition number.

Even though we have effective algorithms for these problems, we cannot necessarily achieve them all for large scale problems in an amount of time and space acceptable to all users.

2.4.7 Related Singular Value Problems

1. If $B = A^*$, and the SVD of $A = U\Sigma V^*$, then the SVD of $B = V\Sigma^*U^*$.
2. Suppose $B = A^*A$, where A is m by n with $m \geq n$. Then B is an n by n Hermitian matrix. Let $A = U\Sigma V^*$ be the SVD of A . Then the eigendecomposition of $B = V(\Sigma^*\Sigma)V^*$. Note that $\Sigma^*\Sigma = \text{diag}(\sigma_1^2, \dots, \sigma_n^2)$. In other words, the eigenvectors of B are the right singular vectors of A , and the eigenvalues of B are the squares of the singular values of A . If $B = AA^*$, where A is m by n with $m \geq n$, then B is m by m with eigendecomposition $B = U(\Sigma\Sigma^*)U^*$. In other words, the eigenvectors of B are the right singular vectors of A , and the eigenvalues of B are the squares of the singular values of A , in addition to 0 with additional multiplicity $m - n$.
3. Suppose $B = \begin{bmatrix} 0^{m \times m} & A \\ A^* & 0^{n \times n} \end{bmatrix}$, where A is m by n with $m \geq n$. Then B is an $(m+n)$ by $(m+n)$ Hermitian matrix. Let $A = U\Sigma V^*$ be the SVD of A . Write $U = [U_1^{m \times n}, U_2^{m \times (m-n)}]$ and $\Sigma = \begin{bmatrix} \widehat{\Sigma}^{n \times n} \\ 0^{(m-n) \times n} \end{bmatrix}$. Then the eigendecomposition of $B = Q\Lambda Q^*$, where $\Lambda = \text{diag}(\widehat{\Sigma}, -\widehat{\Sigma}, O^{(m-n) \times (m-n)})$ and

$$Q = \begin{bmatrix} \frac{1}{\sqrt{2}}U_1 & -\frac{1}{\sqrt{2}}U_1 & U_2 \\ \frac{1}{\sqrt{2}}V & +\frac{1}{\sqrt{2}}V & O^{n \times (m-n)} \end{bmatrix}.$$

In other words, $\pm\sigma_i$ is an eigenvalue with unit eigenvector $\frac{1}{\sqrt{2}}\begin{bmatrix} \pm u_i \\ v_i \end{bmatrix}$ for $i = 1$ to n , and 0 is an eigenvalue with eigenvector $\begin{bmatrix} u_i \\ 0^{n \times 1} \end{bmatrix}$ for $i > n$.

4. The *QSVD* or *generalized SVD (GSVD)* of A and B is defined as follows. Suppose A is m by n and B is n by n and nonsingular. Let $r = \min(m, n)$. Let the SVD of $AB^{-1} = U\Sigma V^*$. We may also write two equivalent decompositions of A and B as $A = U\Sigma_A X$ and $B = V\Sigma_B X$, where X is n by n and nonsingular, Σ_A is m by n and contains $\text{diag}(\sigma_{A,1}, \dots, \sigma_{A,r})$ in its leading r rows and columns and zeros elsewhere, and Σ_B is n by n and contains $\text{diag}(\sigma_{B,1}, \dots, \sigma_{B,n})$.

Σ_A and Σ_B can be chosen so that $\Sigma_A^T \Sigma_A + \Sigma_B^T \Sigma_B = I^{n \times n}$. Thus $\Sigma = \Sigma_A \Sigma_B^{-1}$. The diagonal entries $\sigma_i = \sigma_{A,i}/\sigma_{B,i}$ of Σ are called the *generalized singular values* of A and B .

This decomposition generalizes to the cases where B is singular or p by n , to a decomposition equivalent to the SVD of AB (the *product SVD*), and indeed to decompositions of arbitrary products of the form $A_1^{\pm 1} A_2^{\pm 1} \cdots A_k^{\pm 1}$ [108].

5. The *CS (cosine/sine) decomposition* of A and B is defined as follows. Suppose the columns of $Z = \begin{bmatrix} A^{m \times n} \\ B^{p \times n} \end{bmatrix}$ are orthonormal. The QSVD of A and B is then equivalent to the decompositions $A = U\Sigma_AX$ and $B = V\Sigma_BX$, where X is unitary. The diagonal entries of Σ_A (Σ_B) are the cosines (sines) of the *principal angles* between the subspace spanned by the columns of Z and the subspace spanned by the first m columns of $I^{(m+p) \times (m+p)}$ [424, 25]. This is used to compute the “distance” (or angles) between subspaces.
6. Suppose A is m -by- n , and B is p by n with full column rank; then the GHEP $A^*A - \lambda B^*B$ can be solved using the QSVD as follows. Write the QSVD as $A = U\Sigma_AX$ and $B = V\Sigma_BX$. The eigendecomposition of $A^*A - \lambda B^*B$ may be written $X^*A^*AX = \Sigma_A^*\Sigma_A$ and $X^*B^*BX = \Sigma_B^*\Sigma_B$. This generalizes to the case of B without full column rank.
7. Finding x that minimizes $\|Ax - b\|_2$ is the *linear least squares problem*. Suppose A is m by n with $m > n$; we say that the least squares problem is *overdetermined*. Let $A = \widetilde{U}\widetilde{\Sigma}V^*$ be the compact SVD of A . If A has full rank, the unique solution of the least squares problem is $x = V\widetilde{\Sigma}^{-1}\widetilde{U}^*b$, although solutions based on the QR decomposition or normal equations are cheaper and more commonly used [50, 12]. If A ’s rank is less than n , then the SVD is often used to solve the least squares problem. In this case the solution is not unique but its unique *minimum norm* solution is

$$x = V\widehat{\Sigma}^\dagger\widehat{U}^*b, \quad \text{where } (\widehat{\Sigma}^\dagger)_{ii} = \begin{cases} \sigma_i^{-1} & \text{if } \sigma_i > 0, \\ 0 & \text{if } \sigma_i = 0. \end{cases}$$

If $m < n$ we say that the least squares problem is *underdetermined*. In this case the solution is not unique but its unique *minimum norm* solution is $x = \widehat{V}\widehat{\Sigma}^\dagger U^*b$, where $A = U\widehat{\Sigma}\widehat{V}^*$ is the compact SVD of A .

2.4.8 Example

We continue to use the example introduced in §2.1 and Figure 2.1. We again consider the case with arbitrary masses $m_i > 0$, and zero damping constants $b_i = 0$. This simplifies the equations of motion to $M\ddot{x}(t) = -Kx(t)$. As in §2.3.8 we solve the equations of motion by substituting $x(t) = e^{\lambda t}x$, where x is a constant vector and λ is a constant scalar to be determined. This leads to $Kx = -\lambda^2 Mx$. Letting $M = LL^T$, where the Cholesky factor $L = \text{diag}(m_1^{1/2}, \dots, m_n^{1/2})$, we see that we need to compute the eigenvalues of the symmetric tridiagonal matrix $\widehat{K} = L^{-1}KL^{-T}$ shown in equation (2.2).

Now we note that the stiffness matrix K can be factored as $K = BD^2B^T$, where $D = \text{diag}(k_1^{1/2}, \dots, k_n^{1/2})$ and

$$B = \begin{bmatrix} 1 & -1 & & & \\ & \ddots & \ddots & & \\ & & \ddots & -1 & \\ & & & & 1 \end{bmatrix}.$$

This lets us write

$$\begin{aligned}\hat{K} &= L^{-1}KL^{-T} = L^{-1}BD^2B^TL^{-T} \\ &= (L^{-1}BD) \cdot (DB^T L^{-T}) = (L^{-1}BD) \cdot (L^{-1}BD)^T \equiv GG^T.\end{aligned}$$

Therefore, the singular values of the bidiagonal matrix

$$G = L^{-1}BD = \begin{bmatrix} \sqrt{k_1/m_1} & -\sqrt{k_2/m_1} & & & \\ & \sqrt{k_2/m_2} & -\sqrt{k_3/m_2} & & \\ & & \ddots & \ddots & \\ & & & \ddots & -\sqrt{k_n/m_{n-1}} \\ & & & & \sqrt{k_n/m_n} \end{bmatrix}$$

are the square roots of the eigenvalues of \hat{K} , and the left singular vectors of G are the eigenvectors of \hat{K} .

Bidiagonal matrices have particularly fast and efficient SVD algorithms.

2.5 Non-Hermitian Eigenproblems

J. Demmel

We assume that A is a general n by n matrix.

2.5.1 Eigenvalues and Eigenvectors

The polynomial $p(\lambda) = \det(\lambda I - A)$ is called the *characteristic polynomial* of A . The roots of $p(\lambda) = 0$ are called the *eigenvalues* of A . Since the degree of $p(\lambda)$ is n , it has n roots, and so A has n eigenvalues. Eigenvalues of a real matrix may be real or appear in complex pairs.

A nonzero vector x satisfying $Ax = \lambda x$ is a (*right*) *eigenvector* for the eigenvalue λ . A nonzero vector y satisfying $y^*A = \lambda y^*$ is a *left eigenvector* for the eigenvalue λ .

An n by n matrix need not have n independent eigenvectors. The simplest example is

$$A(\epsilon) = \begin{bmatrix} 0 & 1 \\ \epsilon^2 & 0 \end{bmatrix}, \quad (2.3)$$

whose eigenvalues are $\lambda_{\pm} = \pm\epsilon$. When $\epsilon \neq 0$, there are two right eigenvectors, $x_{\pm} = [1, \pm\epsilon]^T$. As ϵ approaches 0, the two eigenvectors approach one another. When $\epsilon = 0$, both eigenvalues equal 0, and there is a single independent right eigenvector parallel

to $[1, 0]^T$. The fact that n independent eigenvectors may not exist (though there is at least one for each distinct eigenvalue) will necessarily complicate both theory and algorithms for the NHEP.

Since the eigenvalues may be complex, there is no fixed way to order them. Nonetheless, it is convenient to number them as $\lambda_1, \dots, \lambda_n$, with corresponding right eigenvectors x_1, \dots, x_n and left eigenvectors y_1, \dots, y_n (if they exist).

2.5.2 Invariant Subspaces

A (*right*) *invariant subspace* \mathcal{X} of A satisfies $Ax \in \mathcal{X}$ for all $x \in \mathcal{X}$. We also write this as $A\mathcal{X} \subset \mathcal{X}$. The simplest example is when \mathcal{X} is spanned by a single eigenvector of A . More generally an invariant subspace may be spanned by a subset of the eigenvectors of A , but since some matrices do not have n eigenvectors, there are invariant subspaces that are not spanned by eigenvectors. For example, the space of all possible vectors is clearly invariant, but it is not spanned by the single eigenvector of $A(0)$ in (2.3). This is discussed further in §2.5.4 below.

A *left invariant subspace* \mathcal{Y} of A analogously satisfies $A^*y \in \mathcal{Y}$ for all $y \in \mathcal{Y}$, and may be spanned by left eigenvectors of A .

2.5.3 Equivalences (Similarities)

Suppose S is a nonsingular matrix. Let $B = S^{-1}AS$. We say that B is *similar* to A and that S is a *similarity transformation*. B has the same eigenvalues as A . If x is an eigenvector of A , so that $Ax = \lambda x$, then $y = S^{-1}x$ is an eigenvector of B .

If S is a unitary matrix, i.e., $S^{-1} = S^*$, we say B is *unitarily similar* to A . If S is real, we say *orthogonal* instead of *unitary*.

2.5.4 Eigendecompositions

We discuss four eigendecompositions, or four matrices that are similar to A and for which it is simpler to solve eigenproblems. The first one, *diagonal form*, exists only when there are n independent eigenvectors. The second one, *Jordan form*, generalizes diagonal form to all matrices. It can be very ill-conditioned (indeed, it can change discontinuously), so for numerical purposes we typically use the third one, *Schur form*, which is cheaper and more stable to compute. We briefly mention a fourth one, which we call *Jordan–Schur form*,² that is as stable as the Schur form but computes some of the detailed information about invariant subspaces provided by the Jordan form.

Define $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_n)$. If there are n independent right eigenvectors x_1, \dots, x_n we define $X = [x_1, \dots, x_n]$. X is called an *eigenvector matrix* of A . The n equalities $Ax_i = \lambda_i x_i$ for $i = 1, \dots, n$ may also be written $AX = X\Lambda$ or $A = X\Lambda X^{-1}$. The factorization

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

is called the *diagonal form* of A . Note that $X^{-1}A = \Lambda X^{-1}$. Letting y_i^* denote the i th row of X^{-1} , we see that $y_i^* A = \lambda y_i^*$. In other words, the rows of X^{-1} are the left eigenvectors of A . Thus, if A has n independent eigenvectors, then it is similar to the diagonal matrix Λ . In this case we call A *diagonalizable*.

²It is usually called *staircase form* in the literature, but we find *Jordan–Schur* more appropriate.

If we take a subset of k columns of X (say $\widehat{X} = X(:, [2, 3, 5])$ = columns 2, 3, and 5), these columns span an invariant subspace of A . If we take the corresponding submatrix $\widehat{\Lambda} = \text{diag}(\lambda_2, \lambda_3, \lambda_5)$ of Λ , and the corresponding three rows of X^{-1} (say $\widehat{Y}^* = X^{-1}([2, 3, 5], :)$), we can write the corresponding *partial diagonal form* as $A\widehat{X} = \widehat{X}\widehat{\Lambda}$ or $\widehat{Y}^*A\widehat{X} = \widehat{\Lambda}$. If the columns in \widehat{X} are replaced by k different vectors spanning the same invariant subspace, we get a different partial eigendecomposition $A\check{X} = \check{X}\check{\Lambda}$, where $\check{\Lambda}$ is a k by k matrix whose eigenvalues are those of $\widehat{\Lambda}$, though $\check{\Lambda}$ may not be diagonal. Similar procedures for producing partial eigendecompositions for the other eigendecompositions discussed below.

If all the λ_i are distinct, there are n independent eigenvectors, and the diagonal form exists. This is the simplest and most common case. For example, if one picks a matrix “at random,”³ the probability is 1 that the eigenvalues are distinct.

A diagonal form of the matrix $A(0)$ in (2.3) does not exist, since it has just one independent eigenvector. Instead, we can compute its *Jordan form*, which is a decomposition

$$A = XJX^{-1},$$

where J is a block-diagonal matrix, i.e., a matrix of the form $J = \text{diag}(J_1, \dots, J_e)$, where each J_i is a square matrix called a *Jordan block*. Each J_i is upper triangular with its single eigenvalue λ_i on the diagonal, and ones on the first superdiagonal. J_i has a single independent right and left eigenvector. $A(0)$ is a 2 by 2 Jordan block with its single eigenvalue at 0, and $A(0) + \lambda_i I$ is a 2 by 2 Jordan block with its single eigenvalue at λ_i . It can be shown that the Jordan form explicitly describes all possible invariant subspaces of A as being spanned by certain subsets of the columns of X [187].

Unfortunately, the Jordan form is generally not suitable for numerical computation. Here are two reasons. First, it can change discontinuously as A changes. For example, the matrix J for $A(\epsilon)$ with $\epsilon \neq 0$ is $\text{diag}(+\epsilon, -\epsilon)$, but for $A(0)$, $J = A(0)$ itself. Second, the eigenvector matrix X can be very ill-conditioned, i.e., hard to invert accurately. In the case of $A(\epsilon)$, the condition number of X grows like $1/\epsilon$.

So instead we use eigendecompositions of the form

$$A = QTQ^*,$$

where Q is a unitary (or orthogonal) similarity transformation, and T is triangular (or quasi-triangular). This is called the *Schur form* of A . When A is complex, or real with all real eigenvalues, then T is triangular with the eigenvalues of A on its diagonal. When A is real with complex eigenvalues and Q is real and orthogonal, then T cannot be real and triangular. Instead, we allow 2 by 2 blocks with complex eigenvalues to appear on the diagonal of T , which we call *quasi-triangular form*. For simplicity of presentation, we will consider the complex case, so that T is triangular. Let $\lambda_1 = T_{11}, \dots, \lambda_n = T_{nn}$ be the eigenvalues in the order they appear on the diagonal of T ; there is a (different) Schur form for every order. The columns of Q are called *Schur vectors*. The Schur form does not give explicit eigenvectors and bases for all invariant subspaces the way diagonal form and Jordan form do, but it can be computed quite stably. The leading k columns of Q span the invariant subspace for eigenvalues λ_1 through λ_k , but all other eigenspaces require a computation. For example, eigenvectors can be computed simply by solving a triangular system of equations taken from T , and multiplying by Q [114].

³For example, choose each entry independently and randomly from $(-1, 1)$ or any other open interval of real numbers.

Finally, we consider the *Jordan–Schur form*. It is quite complicated, so we only summarize its properties here. Like the Schur form, it only uses unitary (orthogonal) transformation, and so can be computed stably. Like the Jordan form, it explicitly shows what the sizes of the Jordan blocks are and gives explicit bases for many more invariant subspaces than Schur form.

Many textbooks give explicit solutions for problems such as computing the matrix of the exponential e^A in terms of the Jordan form of A [114]. These methods are to be avoided numerically, because of the difficulty of computing the Jordan form. Nearly all these problems have alternative solutions in terms of the Schur form, or in some cases the Jordan–Schur form.

2.5.5 Conditioning

An eigenvalue of a general matrix A can be well-conditioned or ill-conditioned. For example, if A is actually Hermitian (or close to it), its eigenvalues are well-conditioned as described in §2.2.5. On the other hand, if A is “far from Hermitian,” then eigenvalues can be very ill-conditioned. For example, matrix $A(\epsilon)$ in (2.3) shows that changing a matrix entry by ϵ^2 can change the eigenvalues by ϵ , which is much larger than ϵ^2 when $|\epsilon| \ll 1$. For example, $\epsilon = 10^{-8}$ is 10^8 times larger than the perturbation $\epsilon^2 = 10^{-16}$, which could be introduced by rounding error. In other words, the eigenvalues can be perturbed by much more than the perturbation of the matrix. As this example hints, this ill-conditioning tends to occur when two or more eigenvalues are very close together.

The eigenvectors may be similarly well-conditioned or ill-conditioned. From §2.2.5 we know that close eigenvalues can have ill-conditioned eigenvectors even for Hermitian matrices. They can even be more sensitive in the non-Hermitian case, as $A(\epsilon)$ in (2.3) again shows: an ϵ^2 perturbation to a matrix whose gap between eigenvalues is 2ϵ can rotate the eigenvectors by $\epsilon \gg \epsilon^2$ or even make one of them disappear entirely.

We refer to §7.13 for further details.

2.5.6 Specifying an Eigenproblem

The following eigenproblems are typical, because they arise naturally in applications and because we have algorithms for them:

1. Compute all the eigenvalues to some specified accuracy. This is typically done by computing the Schur form.
2. Compute the eigenvalues λ_i in some region of the complex plane. This is typically done by computing an approximate right (and perhaps left) invariant subspace corresponding to eigenvalues in the desired region. The regions of the plane for which we have effective algorithms include
 1. the eigenvalues of closest to (or farthest from) a user-selected point μ ,
 2. the eigenvalues of largest (or smallest) real (or imaginary) part,
 3. the eigenvalues closest to (or farthest from) any selected line or circle in the complex plane.

For each of these possibilities, the user can also compute a *projection* of the matrix on the specified invariant subspace; if the subspace is k -dimensional, then the projection is a k by k matrix whose eigendecomposition can be computed. The user can also compute the right (and perhaps left) eigenvectors in the computed invariant subspace. For the eigenvalues that are clustered together, the user may choose to compute the associated invariant subspace, since in this case the individual eigenvectors can be very ill-conditioned, while the invariant subspace may be less so. Finally, for any of these quantities, the user might also want to compute its condition number.

Even though we have effective algorithms for these problems, we cannot necessarily solve all large scale problems in an amount of time and space acceptable to all users.⁴

2.5.7 Related Eigenproblems

1. Consider the GNHEP $Ax = \lambda Bx$, where A and B are square and B is nonsingular. The matrix $B^{-1}A$ has the same eigenvalues and right eigenvectors as $Ax - \lambda Bx$. If y is a left eigenvector, i.e., $y^*A = \lambda y^*B$, B^*y is a left eigenvector of $B^{-1}A$. Analogous statements are true about AB^{-1} . If $B = B_L B_R$ is a factorization of B (from Gaussian elimination, QR decomposition, or anything else), $B_L^{-1}AB_R^{-1}$ has the same eigenvalues as A , right eigenvector $B_R x$, and left eigenvector B_L^*y . If B is well-conditioned, or AB^{-1} , $B^{-1}A$, or $B_L^{-1}AB_R^{-1}$ can be accurately computed, this is an effective way to solve $Ax = \lambda Bx$. If B is ill-conditioned, it is preferable to treat it as the GNHEP, see §2.6.
2. Let $p(\lambda) = \lambda^n - \sum_{i=0}^{n-1} a_i \lambda^i$ be a monic polynomial. Define the n by n *companion matrix* of $p(\lambda)$ as

$$C = \begin{bmatrix} 0 & 1 & & & \\ & \ddots & \ddots & & \\ & & \ddots & \ddots & \\ & & & 0 & 1 \\ a_0 & a_1 & \cdots & a_{n-2} & a_{n-1} \end{bmatrix},$$

where all entries not explicitly shown are 0. Then the eigenvalues of C are the roots λ_i of $p(\lambda) = 0$, and the right eigenvectors are $x_i = [1, \lambda_i, \dots, \lambda_i^{n-1}]^T$. C is not diagonalizable if $p(\lambda)$ has multiple roots. A reliable, but not optimally efficient, algorithm for finding roots of a polynomial $p(\lambda)$ is to find all the eigenvalues of C .

3. Let $p(\lambda) = \lambda^n - \sum_{i=0}^{n-1} A_i \lambda^i$ be a monic matrix polynomial, where each A_i is an m by m matrix. An eigenpair (λ_i, x_i) of $p(\lambda)$ satisfies $p(\lambda_i)x_i = 0$. Define the nm by nm *block companion matrix* of $p(\lambda)$ as

⁴In the case of Hermitian eigenproblems, we mentioned the ability to count the number of eigenvalues in an interval $[\alpha, \beta]$. This can sometimes be done for sparse Hermitian matrices much more cheaply than computing the eigenvalues in $[\alpha, \beta]$. Although such counting algorithms exist for the NHEP [32], their costs are similar to transformation methods that actually compute the eigenvalues, so we do not pursue them.

$$C = \begin{bmatrix} 0 & I & & & \\ & \ddots & \ddots & & \\ & & \ddots & \ddots & \\ & & & 0 & I \\ A_0 & A_1 & \cdots & A_{n-2} & A_{n-1} \end{bmatrix},$$

where all entries are m by m blocks and all entries not explicitly shown are 0. Then the eigenvalues λ_i of C are the eigenvalues of $p(\lambda_i)$. Note that there are mn eigenvalues. If (λ_i, x_i) is an eigenpair of $p(\lambda)$, then $[x_i^T, \lambda_i x_i^T, \dots, \lambda_i^{n-1} x_i^T]^T$ is a right eigenvector of C [194].

2.5.8 Example

We continue to use the example introduced in §2.1 and Figure 2.1. We now consider the case of unit masses $m_i = 1$ but nonzero damping constants b_i . (The case of nonunit masses is handled in §2.6.8.) This simplifies the equations of motion to $\ddot{x}(t) = -B\dot{x}(t) - Kx(t)$. We solve them by changing variables to

$$y(t) = \begin{bmatrix} \dot{x}(t) \\ x(t) \end{bmatrix},$$

yielding

$$\begin{aligned} \dot{y}(t) &= \begin{bmatrix} \ddot{x}(t) \\ \dot{x}(t) \end{bmatrix} = \begin{bmatrix} -B\dot{x}(t) - Kx(t) \\ \dot{x}(t) \end{bmatrix} = \begin{bmatrix} -B & -K \\ I & 0 \end{bmatrix} \cdot \begin{bmatrix} \dot{x}(t) \\ x(t) \end{bmatrix} \\ &= \begin{bmatrix} -B & -K \\ I & 0 \end{bmatrix} \cdot y(t) \equiv Ay(t). \end{aligned} \quad (2.4)$$

We again solve by substituting $y(t) = e^{\lambda t}y$, where y is a constant vector and λ is a constant scalar to be determined. This yields

$$Ay = \lambda y.$$

Thus y is an eigenvector and λ is an eigenvalue of the non-Hermitian matrix A .

2.6 Generalized Non-Hermitian Eigenproblems

J. Demmel

We assume that A and B are general n by n matrices. We call $A - \lambda B$ a *matrix pencil*, or *pencil* for short. The most common case, and the one we will deal with first, is the *regular case*, which occurs when A and B are square and the *characteristic polynomial* $p(\lambda) = \det(\lambda B - A)$ is not zero for all λ .⁵ This is equivalent to assuming that there are n eigenvalues (finite or infinite) and that they are continuous functions of A and B , i.e., that small changes in A and B cause small changes in the eigenvalues (this requires an appropriate definition for the case of infinite eigenvalues).

We will deal with the singular case at the end of this section.

⁵For example, $p(\lambda)$ is identically zero for the 1-by-1 pencil $A - \lambda B = 0 - \lambda \cdot 0$.

2.6.1 Eigenvalues and Eigenvectors

The polynomial $p(\lambda) = \det(\lambda B - A)$ is the characteristic polynomial of $A - \lambda B$. The degree of $p(\lambda)$ is at most n . The roots of $p(\lambda) = 0$ are called the *finite eigenvalues* of $A - \lambda B$. If the degree of $p(\lambda)$ is $d < n$, we say that $A - \lambda B$ has $n - d$ *infinite eigenvalues* too. For example,

$$A - \lambda B = \begin{bmatrix} 1 & & \\ & 1 & \\ & & 0 \end{bmatrix} - \lambda \begin{bmatrix} 2 & & \\ & 0 & \\ & & 1 \end{bmatrix}$$

has characteristic polynomial $p(\lambda) = (1 - 2 \cdot \lambda)(1 - 0 \cdot \lambda)(0 - 1 \cdot \lambda) = 2\lambda^2 - \lambda$, and so has eigenvalues .5, ∞ , and 0.

If λ is a finite eigenvalue, a nonzero vector x satisfying $Ax = \lambda Bx$ is a (*right eigenvector* for the eigenvalue λ). A nonzero vector y satisfying $y^* A = \lambda^* B$ is a (*left eigenvector*).

If ∞ is an eigenvalue, nonzero vectors x and y satisfying $Bx = 0$ and $y^* B = 0$ are called right and left eigenvectors, respectively.

An n by n pencil $A - \lambda B$ need not have n independent eigenvectors. The simplest example is $A(\epsilon) - \lambda I$, which is defined in equation (2.3) and discussed in §2.5.1. The fact that n independent eigenvectors may not exist (though there is at least one for each distinct eigenvalue) will necessarily complicate both theory and algorithms for the GNHEP.

Since the eigenvalues may be complex or infinite, there is no fixed way to order them. Nonetheless, it is convenient to number them as $\lambda_1, \dots, \lambda_n$, with corresponding right eigenvectors x_1, \dots, x_n and left eigenvectors y_1, \dots, y_n (if they exist).

2.6.2 Deflating Subspaces

A *pair of right and left deflating subspaces* \mathcal{X} and \mathcal{Y} of $A - \lambda B$ have the same dimension and satisfy $Ax \in \mathcal{Y}$ and $Bx \in \mathcal{Y}$ for all $x \in \mathcal{X}$, and furthermore $\text{span}_{x \in \mathcal{X}}\{Ax, Bx\} = \mathcal{Y}$. We also write this as $A\mathcal{X} + B\mathcal{X} = \mathcal{Y}$. The simplest example is when \mathcal{X} is spanned by a single right eigenvector of $A - \lambda B$ and \mathcal{Y} is spanned by its image under A and/or B . More generally a right deflating subspace may be spanned by a subset of right eigenvectors of $A - \lambda B$, and the left deflating subspace by their images. But since some pencils do not have n independent eigenvectors, there are right deflating subspaces that are not spanned by eigenvectors. For example, see the example in §2.5.2.

2.6.3 Equivalences

Suppose X and Y are nonsingular matrices. Let $\widehat{A} = Y^*AX$ and $\widehat{B} = Y^*BX$. We say that the pencil $\widehat{A} - \lambda \widehat{B}$ is *equivalent* to $A - \lambda B$ and that X and Y are *equivalence transformations*. $\widehat{A} - \lambda \widehat{B}$ and $A - \lambda B$ have the same eigenvalues. If x (y) is a right (left) eigenvector of $A - \lambda B$, then $\widehat{x} = X^{-1}x$ ($\widehat{y} = Y^{-1}y$) is a right (left) eigenvector of $\widehat{A} - \lambda \widehat{B}$.

2.6.4 Eigendecompositions

We discuss four eigendecompositions, or four pencils that are equivalent to $A - \lambda B$ and for which it is simpler to solve eigenproblems. This section is analogous to §2.5.4.

The first eigendecomposition, *diagonal form*, exists only when there are n independent eigenvectors. The second one, *Weierstrass form*, generalizes diagonal form to all pencils where the characteristic polynomial $p(\lambda)$ is not identically 0. We can also describe the Weierstrass form as the generalization of the Jordan form matrices to pencils. The Weierstrass form, like the Jordan form, can be very ill-conditioned (indeed, it can change discontinuously), so for numerical purposes we typically use the third one, *generalized Schur form*, which is cheaper and more stable to compute. We briefly mention a fourth one, which we call the *Weierstrass–Schur form*,⁶ that is as stable as the Schur form but computes some of the detailed information about deflating subspaces provided by the Weierstrass form.

Define $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_n)$. If there are n independent right eigenvectors x_1, \dots, x_n we define $X = [x_1, \dots, x_n]$. X is called a (*right*) *eigenvector matrix* of A . Similarly let $Y = [y_1, \dots, y_n]$ be a *left eigenvector matrix*. The $2n$ equalities $Ax_i = \lambda_i Bx_i$ and $y_i^* A = \lambda_i y_i^* B$ for $i = 1, \dots, n$ may also be written $AX = BX\Lambda$ and $Y^*A = \Lambda Y^*B$. X and Y may furthermore be chosen so that $Y^*AX = \Lambda_A$ and $Y^*BX = \Lambda_B$ are both diagonal, and $\Lambda = \Lambda_A \Lambda_B^{-1}$.⁷ The factorization

$$A - \lambda B = Y^{-*}(\Lambda_A - \lambda \Lambda_B)X^{-1}$$

is called the *diagonal form* of $A - \lambda B$. In this case we call $A - \lambda B$ *diagonalizable*.

If we take a subset of k columns of X and of Y (say $\hat{X} = X(:, [2, 3, 5])$ = columns 2, 3, and 5 and $\hat{Y} = Y(:, [2, 3, 5])$) then these columns span a pair of deflating subspaces of $A - \lambda B$. If we take the corresponding submatrices $\hat{\Lambda}_A = \text{diag}(\lambda_{A,2}, \lambda_{A,3}, \lambda_{A,5})$ and $\hat{\Lambda}_B = \text{diag}(\lambda_{B,2}, \lambda_{B,3}, \lambda_{B,5})$, then we can write the corresponding *partial diagonal form* as $\hat{Y}^*A\hat{X} = \hat{\Lambda}_A$ and $\hat{Y}^*B\hat{X} = \hat{\Lambda}_B$. If the columns in \hat{X} and \hat{Y} are replaced by k different vectors spanning the same deflating subspaces, then we get a different partial eigendecomposition $\check{Y}^*A\check{X} = \check{A}$ and $\check{Y}^*B\check{X} = \check{B}$, where $\check{A} - \lambda \check{B}$ is a k by k pencil whose eigenvalues are those of $\check{\Lambda} = \Lambda_A \Lambda_B^{-1}$, though $\check{A} - \lambda \check{B}$ may not be diagonal. Similar procedures for producing partial eigendecompositions work for the other eigendecompositions discussed below.

If all the λ_i are distinct, then there are n independent eigenvectors, and the diagonal form exists. This is the simplest and most common case. For example, if one picks A and B “at random,”⁸ the probability is 1 that the eigenvalues are distinct.

A diagonal form of the pencil $A(0) - \lambda I$ in (2.3) in §2.5.4 does not exist, since it has just one independent eigenvector. Instead, we can compute its *Weierstrass form*, which is a decomposition

$$A - \lambda B = Y^{-*}(J_A - \lambda J_B)X^{-1},$$

where $J_A - \lambda J_B$ is a block-diagonal pencil, with one or more upper triangular diagonal blocks for each eigenvalue. When there are no infinite eigenvalues, this is identical to the Jordan form discussed in §2.5.4. When there are infinite eigenvalues, there are blocks quite similar to Jordan blocks with a single infinite eigenvalue and one right and left eigenvector.⁹ It can be shown that the Weierstrass form explicitly describes

⁶It is usually called *staircase form* in the literature, but we find Weierstrass–Schur more appropriate.

⁷To see this in the generic case when all the eigenvalues are distinct, multiply $AX = BX\Lambda$ by Y^* and $Y^*A = \Lambda Y^*B$ by X to get $\Lambda Y^*BX = Y^*AX = Y^*BX\Lambda$. $\Lambda Y^*BX = Y^*BX\Lambda$ implies that $Y^*BX = \Lambda_B$ must be diagonal, whence $Y^*AX = \Lambda_A = \Lambda_B\Lambda$ is also diagonal.

⁸For example, choose each entry independently and randomly from $(-1, 1)$ or any other open interval of real numbers.

⁹These blocks look like $I - \lambda J(0)$, where $J(0)$ is a Jordan block with 0 eigenvalue.

all possible deflating subspaces of $A - \lambda B$ as being spanned by certain subsets of the columns of X and Y [187].

Unfortunately, the Weierstrass form is generally not suitable for numerical computation, for the same reason that the Jordan form is not suitable. See §2.5.4 for discussion.

So instead we use eigendecompositions of the form

$$A - \lambda B = Z(T_A - \lambda T_B)Q^*,$$

where Q and Z are unitary (or orthogonal) matrices and $T_A - \lambda T_B$ is triangular (or quasi-triangular). This is called the *generalized Schur form* of $A - \lambda B$. When $A - \lambda B$ is complex, or real with all real or infinite eigenvalues, then $T_A - \lambda T_B$ is triangular with the eigenvalues of $A - \lambda B$ on its diagonal. When $A - \lambda B$ is real with complex eigenvalues and Q is real and orthogonal, then $T_A - \lambda T_B$ cannot be real and triangular. Instead, we allow 2 by 2 blocks with complex eigenvalues to appear on the diagonal of $T_A - \lambda T_B$, which we call *quasi-triangular form*. For simplicity of presentation, we will consider the complex case, so that $T_A - \lambda T_B$ is triangular. Let $\lambda_1 = T_{A,11}/T_{B,11}, \dots, \lambda_n = T_{A,nn}/T_{B,nn}$ be the eigenvalues in the order they appear on the diagonal of T ; there is a (different) generalized Schur form for every order. The columns of Q and Z are called *generalized Schur vectors*. The generalized Schur form does not give explicit eigenvectors and bases for all deflating subspaces the way diagonal form and Weierstrass form do, but it can be computed quite stably. The leading k columns of Q and of Z do span the right and left deflating subspaces for eigenvalues λ_1 through λ_k , but all other eigenspaces require a computation. For example, right eigenvectors can be computed simply by solving a triangular system of equations taken from $T_A - \lambda T_B$ and multiplying by Q .

Finally, we consider the *Weierstrass–Schur form*. It is quite complicated, so we only summarize its properties here. Like the Schur form, it only uses unitary (orthogonal) transformation and so can be computed stably. Like the Weierstrass form, it explicitly shows what the sizes of the (Jordan) blocks are and gives explicit bases for many more invariant subspaces than the Schur form.

Many textbooks give explicit solutions for problems such as solving ordinary differential equations $B\dot{x}(t) = Ax(t)$ in terms of the Weierstrass form of $A - \lambda B$ [114]. These methods are to be avoided numerically, because of the difficulty of computing the Weierstrass form. Nearly all these problems have alternative solutions in terms of the generalized Schur form, or in some cases the Weierstrass–Schur form.

2.6.5 Conditioning

Since the eigenproblem for a general matrix A can also be written as the generalized eigenproblem $A - \lambda I$, all the comments about conditioning in §2.5.5 apply here: eigenvalues can be well-conditioned or ill-conditioned and small perturbations can even make eigenvectors disappear entirely.

When B is singular, then $A - \lambda B$ has at least one infinite eigenvalue. In order to speak of small perturbations to B causing “small perturbations” to an infinite eigenvalue, we need an appropriate definition. This is supplied by the *chordal metric*, which we describe in several equivalent ways. First, any eigenvalue λ_i , finite or infinite, can always be written as a ratio α_i/β_i , where $|\alpha_i|^2 + |\beta_i|^2 = 1$. For example, if $|\lambda_i| = \tan \theta$ for some θ , then $|\alpha_i| = \sin \theta$ and $|\beta_i| = \cos \theta$. The *chordal metric* $\chi(\lambda_1, \lambda_2) = \sin \tau$,

where τ is the acute angle between the two vectors (α_1, β_1) and (α_2, β_2) . We may compute this as $\chi(\lambda_1, \lambda_2) = |\alpha_1\beta_2 - \alpha_2\beta_1|$ or $\chi(\lambda_1, \lambda_2) = \frac{|\lambda_1 - \lambda_2|}{\sqrt{1+|\lambda_1|^2}\sqrt{1+|\lambda_2|^2}}$ if neither λ_1 nor λ_2 is infinite. For example, If $\lambda_1 = 1/0$ is infinite and $\lambda_2 = \sqrt{1 - 10^{-16}}/10^{-8} \approx 10^8$ is very large, then $\chi(\lambda_1, \lambda_2) = 10^{-8}$ is small, so that λ_1 and λ_2 are close.

Another, geometric way to understand the chordal metric is with the *Riemann sphere* [4]: Imagine a three-dimensional ball of radius 1 with center at the origin of the complex plane; the ball's surface is the Riemann sphere. For any λ in the complex plane, draw a straight line connecting λ to the north pole of the Riemann sphere, and let $\hat{\lambda}$ be the point where the straight line and the Riemann sphere intersect. Then $\chi(\lambda_1, \lambda_2)$ is just the Euclidean distance $\|\hat{\lambda}_1 - \hat{\lambda}_2\|_2$ between the two points on the $\hat{\lambda}_1$ and $\hat{\lambda}_2$ on the Riemann sphere.

The chordal metric is used to measure changes in eigenvalues in §8.8. For example, the single infinite eigenvalue of $1 - \lambda \cdot 0$ is well-conditioned, because changing 0 to 10^{-8} can change the infinite eigenvalue to 10^8 , and $\chi(10^8, \infty) \approx 10^{-8}$.

We refer to §8.8 for further details.

2.6.6 Specifying an Eigenproblem

This section is nearly identical to the corresponding §2.5.6.

1. Compute all the eigenvalues to some specified accuracy. This is typically done by computing the Schur form.
2. Compute eigenvalues λ_i in some region of the complex plane. This is typically done by computing an approximate right (and perhaps left) invariant subspace corresponding to eigenvalues in the desired region. The regions of the plane for which we have effective algorithms include
 - a. the eigenvalues closest to (or farthest from) a user-selected point μ (including ∞),
 - b. the eigenvalues of largest (or smallest) real (or imaginary) part,
 - c. the eigenvalues closest to (or farthest from) any selected line or circle in the complex plane.

For each of these possibilities, the user can also compute a *projection* of the matrix on the specified invariant subspace; if the subspace is k -dimensional, then the projection is a k by k matrix whose eigendecomposition can be computed. The user can also compute the right (and perhaps left) eigenvectors in the computed invariant subspace. For the eigenvalues that are clustered together, the user may choose to compute the associated invariant subspace, since in this case the individual eigenvectors can be very ill-conditioned, while the invariant subspace may be less so. Finally, for any of these quantities, the user might also want to compute its condition number.

Even though we have effective algorithms for these problems, we cannot necessarily solve all large scale problems in an amount of time and space acceptable to all users.¹⁰

¹⁰In the case of GHEPs, we mentioned the ability to count the number of eigenvalues in an interval $[\alpha, \beta]$. This can sometimes be done for sparse Hermitian pencils much more cheaply than computing the eigenvalues in $[\alpha, \beta]$. Although such counting algorithms exist for the regular GNHEP [32], their costs are similar to transformation methods that actually compute the eigenvalues, so we do not pursue them.

2.6.7 Related Eigenproblems

1. If B is nonsingular, then the NHEP $B^{-1}Ax = \lambda x$ has the same eigenvalues λ_i and corresponding right eigenvectors x_i as $Ax = \lambda Bx$. Similarly, $AB^{-1}z = \lambda z$ has the same eigenvalues λ_i as $Ax = \lambda Bx$ and right eigenvectors $z_i = Bx_i$. If A is nonsingular, $A^{-1}Bx = \mu x$ has the same right eigenvectors x_i as $Ax = \lambda Bx$, and its eigenvalues are reciprocals $\mu_i = 1/\lambda_i$. Finally, if A is nonsingular, $BA^{-1}z = \mu z$ has reciprocal eigenvalues $\mu_i = 1/\lambda_i$ and right eigenvectors $z_i = Ax_i$. Analogous statements can be made about left eigenvectors.
2. More generally, suppose $Ax = \lambda Bx$ has eigenvalues λ_i and corresponding right eigenvectors x_i . Let $\alpha_1, \alpha_2, \beta_1$, and β_2 be scalars such that $\alpha_1\beta_2 - \alpha_2\beta_1 \neq 0$. Then $(\alpha_1A + \beta_1B)x = \mu(\alpha_2A + \beta_2B)x$ has the same eigenvectors x_i as $Ax = \lambda Bx$ and eigenvalues $\mu_i = (\alpha_1\lambda_i + \beta_1)/(\alpha_2\lambda_i + \beta_2)$. If one or both of $\alpha_2A + \beta_2B$ and $\alpha_1A + \beta_1B$ are nonsingular, then the method in item 1 above can be applied.
3. Let $p(\lambda) = A_n\lambda_n - \sum_{i=0}^{n-1} A_i\lambda^i$ be an m -by- m matrix polynomial, where $\det p(\lambda)$ is not identically zero. An eigenpair (λ_i, x_i) of $p(\lambda)$ satisfies $p(\lambda_i)x_i = 0$. Define the mn by mn block companion pencil of $p(\lambda)$ as

$$A - \lambda B = \begin{bmatrix} 0 - \lambda I & I & & & \\ & \ddots & \ddots & & \\ & & \ddots & \ddots & \\ & & & 0 - \lambda I & I \\ A_0 & A_1 & \cdots & A_{n-2} & A_{n-1} - \lambda A_n \end{bmatrix},$$

where all entries are m by m blocks and all entries not explicitly shown are 0. Then $Ax = \lambda Bx$ is a regular generalized eigenvalue problem, and the eigenvalues of $A - \lambda B$ are the eigenvalues of $p(\lambda)$. Note that there are mn eigenvalues. If (λ_i, x_i) is an eigenpair of $p(\lambda)$, then $[x_i^T, \lambda_i x_i^T, \dots, \lambda_i^{n-1} x_i^T]^T$ is a right eigenvector of C [194].

2.6.8 Example

We continue to use the example introduced in §2.1 and Figure 2.1. We now consider the fully general case of nonzero masses m_i and damping constants b_i . This leads to the equations of motion $M\ddot{x}(t) = -B\dot{x}(t) - Kx(t)$. We solve them by changing variables to

$$y(t) = \begin{bmatrix} \dot{x}(t) \\ x(t) \end{bmatrix},$$

yielding

$$\begin{bmatrix} M & 0 \\ 0 & I \end{bmatrix} \dot{y}(t) = \begin{bmatrix} M\ddot{x}(t) \\ \dot{x}(t) \end{bmatrix} = \begin{bmatrix} -B\dot{x}(t) - Kx(t) \\ \dot{x}(t) \end{bmatrix} = \begin{bmatrix} -B & -K \\ I & 0 \end{bmatrix} \cdot y(t)$$

or $A\dot{y}(t) = Cy(t)$. We solve this by substituting $y(t) = e^{\lambda t}y$, where y is a constant vector and λ is a constant scalar to be determined. This yields

$$Cy = \lambda Ay.$$

Thus y is an eigenvector and λ is an eigenvalue of the generalized nonsymmetric eigenvalue problem.

2.6.9 Singular Case

Eigenvalues and Eigenvectors. The singular case of $A - \lambda B$ corresponds to either

- $A - \lambda B$ is square and singular for all values of λ , or
- $A - \lambda B$ is rectangular.

Both cases arise in practice, and are significantly more challenging than the regular case. We outline the theory here and leave details to §8.7.

Consider

$$A - \lambda B = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}. \quad (2.5)$$

Then $p(\lambda) = \det(A - \lambda B) \equiv 0$ for all λ , so $A - \lambda B$ is singular. For any λ , $Ax = \lambda Bx = 0$ for $x = [0, 1]^T$. But rather than calling all λ eigenvalues, we only call 1 an eigenvalue of this pencil, because $Ax = 1 \cdot Bx$ for $x = [1, 0]^T$, and the rank of $A - 1 \cdot B$ is 0, which is lower than the rank of $A - \lambda B$ for any other value of λ . In general, if $A - \mu B$ has a lower rank than the rank of $A - \lambda B$ for almost all other values of λ , then μ is an eigenvalue.

Eigenvalues are discontinuous functions of the matrix entries when the pencil is singular, which is one reason we have to be careful about definitions. This discontinuity is further discussed below.

Eigenvectors are also no longer so simply defined. For example, consider

$$A - \lambda B = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \quad (2.6)$$

Then 1 is an eigenvalue but $Ax = 1 \cdot Bx$ for any $x = [z, z, 1]^T$ for any value of z . Instead we consider *reducing subspaces*, as defined below.

Reducing Subspaces. A pair of right and left reducing subspaces \mathcal{X} and \mathcal{Y} of $A - \lambda B$ satisfy $Ax \in \mathcal{Y}$ and $Bx \in \mathcal{X}$ for all $x \in \mathcal{X}$, and furthermore $\text{span}_{x \in \mathcal{X}}(Ax, Bx) = \mathcal{Y}$. Also, the dimension of \mathcal{X} exceeds the dimension of \mathcal{Y} by $\dim(N_r)$, the dimension of the right null space of $A - \lambda B$ over the field of all rational functions of λ [119]. It is easy to express $\dim(N_r)$ in terms of the Kronecker canonical form, described below. There is still a correspondence between subsets of eigenvalues and reducing subspaces (as there was a correspondence between subsets of eigenvalue of invariant subspaces for single matrices), but reducing subspaces are no longer spanned by eigenvectors, which are no longer well defined.

Consider the singular pencil (2.6). It has a nontrivial right reducing subspace \mathcal{X} spanned by the first two columns of the 3 by 3 identity matrix and corresponding left reducing subspace \mathcal{Y} spanned by $[1, 0]^T$. Here N_r is spanned by $[1, \lambda, 0]^T$.

Equivalences. Suppose X and Y are nonsingular matrices. Let $\widehat{A} = Y^*AX$ and $\widehat{B} = Y^*BX$. We say that the pencil $\widehat{A} - \lambda \widehat{B}$ is *equivalent* to $A - \lambda B$ and that X and Y are *equivalence transformations*. $\widehat{A} - \lambda \widehat{B}$ and $A - \lambda B$ have the same eigenvalues. If \mathcal{X} (\mathcal{Y}) is a right (left) reducing subspace of $A - \lambda B$, then $\widehat{\mathcal{X}} = X^{-1}\mathcal{X}$ ($\widehat{\mathcal{Y}} = Y^*\mathcal{Y}$) is a right (left) reducing subspace of $\widehat{A} - \lambda \widehat{B}$.

Eigendecompositions. We discuss two eigendecompositions, or pencils that are equivalent to $A - \lambda B$, and for which it is simpler to solve eigenproblems. The decompositions are discussed in more detail in §8.7.

There is no analogy to the *diagonal form*; singular pencils are too complicated for this. There are only the *Kronecker form* and the *generalized Schur-staircase form*, which is sometimes also called the *GUPTRI form* (for generalized upper triangular form) or *Kronecker–Schur form*. The Kronecker form generalizes the Jordan and the Weierstrass forms to singular pencils. In addition to their discontinuities and other numerical difficulties, it adds its own difficulties associated with singular pencils. The generalized Schur-staircase form generalized both the generalized Schur form and the Weierstrass–Schur form. It can be computed stably with orthogonal transformations, but retains the discontinuities inherent in singular pencils.

Given the difficulty associated with singular pencils, partial eigendecompositions are usually not computed; all known algorithms compute essentially complete decompositions.

Conditioning. As stated before, eigenvalues of singular pencils are discontinuous functions of the matrix entries. For example, consider the singular pencil in (2.5). Changing A_{22} to $\epsilon_1 \ll 1$ and B_{22} to $\epsilon_2 \ll 1$ makes this pencil regular, with an eigenvalue at 1 and one at ϵ_1/ϵ_2 , which can be arbitrary, no matter how small the ϵ values are. Furthermore, changing A_{12} to $\epsilon_{A12} \ll 1$, A_{21} to $\epsilon_{A21} \ll 1$, B_{12} to $\epsilon_{B12} \ll 1$, and B_{21} to $\epsilon_{B21} \ll 1$, leads to eigenvalues $\epsilon_{A12}/\epsilon_{B12}$ and $\epsilon_{A21}/\epsilon_{B21}$, both of which can be arbitrary, no matter how small the ϵ values are. Reducing subspaces can also change discontinuously.

Nonetheless, there are important situations where eigenvalues and reducing subspaces change continuously. This happens because the perturbations may be *constrained* to keep the dimension of a selected reducing subspace constant. In fact, the algorithms can be told to enforce such constraints when computing eigenvalues and reducing subspaces. This leads to useful bounds presented in detail in §8.7 and [119].

Specifying an Eigenproblem. All known algorithms for singular pencils compute an essentially complete Kronecker–Schur form of dense matrices $A - \lambda B$, so little advantage can be taken of sparsity or when only partial information is desired. Eigenvalues, reducing subspaces, and their condition numbers may be computed.

Related Eigenproblems. Singular pencils arise naturally in the study of linear control systems $\dot{E}\dot{x}(t) = Ax(t) + Bu(t)$, $y(t) = Cx(t)$. Here $u(t)$ is a *control input* that the user attempts to select to make sure the *state variable* $x(t)$ and *output variable* $y(t)$ attain desired values. Concepts such as *controllability*, *controllable subspace*, *controllable and uncontrollable modes*, *observability*, etc. can all be formulated and computed in terms of reducing subspaces and eigenvalues [460, 447, 451, 450].

Example. Consider the mass-spring system introduced in §2.1. If the system is undamped, it will continue to vibrate forever once started. The question we ask is, Can the user grab and move one of the masses, say the k th one, in such a way as to eventually bring all the masses to rest? The question is about *controllability*, and the answer depends both on the value of k and on the number and values of the

spring constants. The calculation involves computing the smallest reducing subspace of $[Y, A - \lambda I]$, where A is defined in (2.4) and Y depends on k .

2.7 Nonlinear Eigenproblems

J. Demmel

This chapter collects several examples of nonlinear eigenvalue problems for which effective algorithms exist. There is no single approach for all nonlinearities, so each example is treated differently.

The simplest kind of nonlinear eigenproblem is the *quadratic eigenvalue problem* (QEP)

$$L(\lambda)x = (\lambda^2 M + \lambda B + K)x = 0.$$

Here λ is called an eigenvalue, and x is a (right) eigenvector. It is called *regular* if $\det L(\lambda)$ is not zero for all λ , and *singular* otherwise. It can be converted to a generalized linear eigenvalue problem as illustrated in item 3 of §2.5.7 and §2.6.7. QEPs arise in the damped mass-spring system introduced in §2.2.8: the equations of motion $M\ddot{x}(t) + B\dot{x}(t) + Kx(t) = 0$ can be solved by substituting $x(t) = e^{\lambda t}x$, where x is a constant vector and λ is a constant scalar to be determined. This yields $\lambda^2 Mx + \lambda Bx + Kx = 0$ as above. See §9.2.

Higher degree polynomial eigenproblems $(\sum_{i=0}^m \lambda^i A_i)x = 0$ can be similarly treated. See §9.3.

Finally, §9.4 considers nonlinear eigenproblems which can be expressed as maximizing a scalar function $F(Y)$ over the set of n by m orthonormal matrices Y . The simplest case is maximizing $F(Y) = Y^*AY$, where A is Hermitian and $m = 1$; the answer is the largest eigenvalue of A . If $F(Y) = \text{tr}Y^*AY$, i.e., the *trace* or sum of diagonal entries of Y^*AY , then the answer is the sum of the m largest eigenvalues of the Hermitian matrix A . For these problems more effective algorithms are available than the conjugate-gradient-based optimization scheme presented here, but its advantage is that it generalizes to far more functions F . We give two more examples. First, if A_1, \dots, A_m are n by n real symmetric matrices that should have a common set of eigenvectors but have been corrupted by noise so that this is no longer so, then we seek an orthogonal Y that minimizes the sums of norms of the offdiagonal entries of all the Y^*A_iY . Second, in quantum mechanical calculations using the local density approximation, one wishes to maximize $\text{tr}Y^*AY + g(Y)$, where $g(Y)$ is a complicated nonlinear term representing the energies of electron-electron interactions. The optimization approach presented here handles quite general functions $g(Y)$.

Chapter 3

An Introduction to Iterative Projection Methods

3.1 Introduction

Since most algorithms and templates presented in detail in this book are iterative projection methods designed for solving large sparse eigenvalue problems, in this chapter, we begin with a discussion of the general framework of these iterative projection methods. It is well known that most of these methods provide rapid approximations to well-separated extremal eigenvalues. However, quite often, the sought-after eigenvalues may not be separated and are interior. The spectral transformation to be outlined in §3.3 is a practical tool for transforming the sought-after eigenvalues to well-separated extremal eigenvalues.

3.2 Basic Ideas

Y. Saad

Most of the standard eigenvalue algorithms exploit projection processes in order to extract approximate eigenvectors from a given subspace. The basic idea of a projection method is to extract an approximate eigenvector from a specified low-dimensional subspace. Certain conditions are required in order to be able to extract these approximations. Once these conditions are expressed, a small matrix eigenvalue problem is obtained.

A somewhat trivial but interesting illustration of the use of projection is when the dominant eigenvalue of a matrix is computed by the *power method* as follows:

```
select  $u_0$ 
for  $i = 0, \dots$ , until convergence do:
   $u = Au_i$ 
   $u_{i+1} = u/\|u\|_2$ 
   $\tilde{\lambda}_{i+1} = u_{i+1}^T Au_{i+1}$ 
```

In the simplest case when the dominant eigenvalue, i.e., the eigenvalue with largest modulus, is real and simple, the pair $u_{i+1}, \tilde{\lambda}_{i+1}$ converges to the eigenvector and associated (largest in modulus) eigenvalue under mild assumptions on the initial vector. If this eigenvalue is complex and real arithmetic is used, then the algorithm will fail to converge. It is possible to work in complex arithmetic by introducing a complex initial vector, and in this way convergence will be recovered. However, it is also possible to extract these eigenvectors by working in real arithmetic. This is based on the observation that two successive iterates u_i and u_{i+1} will tend to belong to the subspace spanned by the two complex conjugate eigenvectors associated with the desired eigenvalue. Let u_i and u_{i+1} be denoted by v_1 and v_2 , respectively. To extract the approximate eigenvectors, we write them as linear combinations of v_1 and v_2 :

$$\tilde{u} = \eta_1 v_1 + \eta_2 v_2.$$

We seek a vector and a scalar $\tilde{\lambda}$ such that

$$A\tilde{u} = \tilde{\lambda}\tilde{u}.$$

This is an underdetermined system. In addition to $\tilde{\lambda}$, two degrees of freedom are available, namely, the scalars η_1 and η_2 . They can be determined by imposing two constraints. It is most common to express these constraints in the form of orthogonality conditions. These are the *Galerkin conditions*, which require that the residual $r = A\tilde{u} - \tilde{\lambda}\tilde{u}$ be orthogonal to the subspace of approximation, namely, to v_1 and v_2 :

$$A\tilde{u} - \tilde{\lambda}\tilde{u} \perp v_1, \quad A\tilde{u} - \tilde{\lambda}\tilde{u} \perp v_2.$$

Define the $n \times 2$ matrix $V \equiv [v_1, v_2]$ and call y the 2-vector of unknowns η_1, η_2 , i.e., $y = (\eta_1, \eta_2)^T$. Then, clearly $\tilde{u} = Vy$ and the above equations give rise to

$$(A - \tilde{\lambda}I)Vy \perp v_1, \quad (A - \tilde{\lambda}I)Vy \perp v_2$$

or equivalently,

$$V^*(A - \tilde{\lambda}I)Vy = 0.$$

This yields the 2×2 generalized eigenvalue problem:

$$V^*AVy = \tilde{\lambda} V^*Vy.$$

The (complex) approximate pair of conjugate eigenvalues obtained from this problem will now converge under the same conditions as those stated for the real case. The iteration uses real vectors, yet the process is able to extract complex eigenvalues. Clearly, the approximate eigenvectors are complex but they need not be computed in complex arithmetic. This is because the (real) basis $[v_1, v_2]$ is also a suitable basis for the complex plane spanned by v_1 and v_2 .

The above idea can now be generalized from two dimensions to m dimensions. Thus, a projection method consists of approximating the exact eigenvector u , by a vector \tilde{u} belonging to some subspace \mathcal{K} referred to as the subspace of approximants or the right subspace. If the subspace has dimension m , this gives m degrees of freedom and the next step is to impose m constraints to be able to extract a unique solution. This is done by imposing the so-called *Petrov-Galerkin condition* that the residual vector of \tilde{u} be orthogonal to some subspace \mathcal{L} , referred to as the left subspace. Though this may

seem artificial, we can distinguish between two broad classes of projection methods. In an *orthogonal projection* technique the subspace \mathcal{L} is the same as \mathcal{K} . In an *oblique projection* method \mathcal{L} is different from \mathcal{K} and can be completely unrelated to it.

The construction of \mathcal{K} can be done in different ways. The above-suggested generalization of the power method leads, if one works with all generated vectors, to the Krylov subspace methods, that is, methods that seek the solution in the subspace

$$\mathcal{K}^m(A, u_0) = \text{span}\{u_0, Au_0, \dots, A^{m-1}u_0\},$$

the so-called *Krylov subspace*. One can also attempt to force these vectors to be more in the direction of the desired eigenvector, by inexact inversion techniques (preconditioning), which leads to Davidson-type methods. Alternatively, one can start with a set (block) of vectors instead of a single vector and this leads to the so-called block variants of these subspace methods.

Orthogonal Projection Methods. Let A be an $n \times n$ complex matrix and \mathcal{K} be an m -dimensional subspace of \mathbb{C}^n and consider the eigenvalue problem of finding u belonging to \mathbb{C}^n and λ belonging to \mathcal{C} such that

$$Au = \lambda u. \quad (3.1)$$

An orthogonal projection technique onto the subspace \mathcal{K} seeks an approximate eigenpair $\tilde{\lambda}, \tilde{u}$ to the above problem, with $\tilde{\lambda}$ in \mathcal{C} and \tilde{u} in \mathcal{K} . This approximate eigenpair is obtained by imposing the following Galerkin condition:

$$A\tilde{u} - \tilde{\lambda}\tilde{u} \perp \mathcal{K}, \quad (3.2)$$

or, equivalently,

$$v^*(A\tilde{u} - \tilde{\lambda}\tilde{u}) = 0 \quad \forall v \in \mathcal{K}. \quad (3.3)$$

In order to translate this into a matrix problem, assume that an orthonormal basis $\{v_1, v_2, \dots, v_m\}$ of \mathcal{K} is available. Denote by V the matrix with column vectors v_1, v_2, \dots, v_m , i.e., $V = [v_1, v_2, \dots, v_m]$. Because we seek a $\tilde{u} \in \mathcal{K}$, it can be written as

$$\tilde{u} = Vy. \quad (3.4)$$

Then, equation (3.4) becomes

$$v_j^*(AVy - \tilde{\lambda}Vy) = 0, \quad j = 1, \dots, m.$$

Therefore, y and $\tilde{\lambda}$ must satisfy

$$B_my = \tilde{\lambda}y \quad (3.5)$$

with

$$B_m = V^*AV. \quad (3.6)$$

The approximate eigenvalues $\tilde{\lambda}_i$ resulting from the projection process are all the eigenvalues of the matrix B_m . The associated eigenvectors are the vectors Vy_i in which y_i is an eigenvector of B_m associated with $\tilde{\lambda}_i$.

This procedure for numerically computing the Galerkin approximations to the eigenvalues/eigenvectors of A is known as the Rayleigh–Ritz procedure.

RAYLEIGH-RITZ PROCEDURE

1. Compute an orthonormal basis $\{v_i\}_{i=1,\dots,m}$ of the subspace \mathcal{K} . Let $V = [v_1, v_2, \dots, v_m]$.
2. Compute $B_m = V^*AV$.
3. Compute the eigenvalues of B_m and select the k desired ones $\tilde{\lambda}_i, i = 1, 2, \dots, k$, where $k \leq m$ (for instance the largest ones).
4. Compute the eigenvectors $y_i, i = 1, \dots, k$, of B_m associated with $\tilde{\lambda}_i, i = 1, \dots, k$, and the corresponding approximate eigenvectors of A , $\tilde{u}_i = Vy_i, i = 1, \dots, k$.

In implementations of this approach, one often does not compute the eigenpairs of B_m for each set of generated basis vectors. The values $\tilde{\lambda}_i$ computed from this procedure are referred to as *Ritz values* and the vectors \tilde{u}_i are the associated *Ritz vectors*. The numerical solution of the $m \times m$ eigenvalue problem in steps 3 and 4 can be treated by standard library subroutines such as those in LAPACK [12]. Another important note is that in step 4 one can replace eigenvectors by Schur vectors to get approximate Schur vectors \tilde{u}_i instead of approximate eigenvectors. Schur vectors y_i can be obtained in a numerically stable way and, in general, eigenvectors are more sensitive to rounding errors than are Schur vectors.

Oblique Projection Methods. In an oblique projection method we are given two subspaces \mathcal{L} and \mathcal{K} and seek an approximation $\tilde{u} \in \mathcal{K}$ and an element $\tilde{\lambda}$ of \mathcal{C} that satisfy the Petrov–Galerkin condition,

$$v^*(A - \tilde{\lambda}I)\tilde{u} = 0 \quad \forall v \in \mathcal{L}. \quad (3.7)$$

The subspace \mathcal{K} will be referred to as the right subspace and \mathcal{L} as the left subspace. A procedure similar to the Rayleigh–Ritz procedure can be devised, and this can be conveniently described in matrix form by expressing the approximate eigenvector \tilde{u} in matrix form with respect to some basis and formulating the Petrov–Galerkin conditions (3.7) for the basis of \mathcal{L} . This time we will need two bases, one which we denote by V for the subspace \mathcal{K} and the other, denoted by W , for the subspace \mathcal{L} . We assume that these two bases are biorthogonal, i.e., that $w_i^*v_j = \delta_{ij}$ or

$$W^*V = I,$$

where I is the identity matrix. Then, writing $\tilde{u} = Vy$ as before, the above Petrov–Galerkin condition yields the same approximate problem as (3.5) except that the matrix B_m is now defined by

$$B_m = W^*AV.$$

In order for a biorthogonal pair V, W to exist, the following additional assumption for \mathcal{L} and \mathcal{K} must hold.

For any two bases V and W , of \mathcal{K} and \mathcal{L} , respectively,

$$\det(W^*V) \neq 0. \quad (3.8)$$

Obviously this condition does not depend on the particular bases selected and it is equivalent to requiring that no vector of \mathcal{K} be orthogonal to \mathcal{L} .

The approximate problem obtained for oblique projection methods has the potential of being much worse conditioned than with orthogonal projection methods. Therefore, one may wonder whether there is any need for using oblique projection methods. However, methods based on oblique projectors can offer some advantages. In particular, they may be able to compute good approximations to left as well as right eigenvectors simultaneously. As will be seen later, there are also methods based on oblique projection techniques which require far less storage than similar orthogonal projections methods.

The disadvantages of working with the nonorthogonal V and W can be reduced by combining this technique with a few steps of a (more expensive) orthogonal projection method.

Harmonic Ritz Values. Our introduction of the Ritz values, in relation with Krylov subspaces, suggests that they tend to approximate exterior eigenvalues better than interior ones. This is supported by theory as well as borne out by experience. Ritz values that are located in the interior part of the spectrum can usually be considered as bad approximations for some exterior eigenvalues, and the corresponding Ritz vectors often have a small component in the eigenvector direction corresponding to the eigenvalue in the vicinity of which the Ritz value lies. One may say that this Ritz value is *on its way to converging towards an exterior eigenvalue*. This means that the Ritz vectors corresponding to interior eigenvalues are also often not suitable for restarting purposes if one is interested in interior eigenpairs. These restarts are necessary, in particular for interior eigenpairs, in order to avoid high-dimensional bases for V and W .

An interesting particular case of oblique projection methods is the situation in which \mathcal{L} is chosen as $\mathcal{L} = A\mathcal{K}$. Similar to previous notation, let V be a basis of \mathcal{K} . Assuming that A is nonsingular, we can take as a basis of \mathcal{L} the system of vectors $W = AV$. The approximate eigenvector \tilde{u} to be extracted from the subspace \mathcal{K} can be expressed in the form

$$\tilde{u} = Vy,$$

where y is an m -dimensional vector. The approximate eigenvalue $\tilde{\lambda}$ is obtained from the Petrov–Galerkin condition, which yields

$$(AV)^*(A - \tilde{\lambda}I)Vy = 0$$

or

$$V^*A^*AVy = \tilde{\lambda} V^*A^*Vy. \quad (3.9)$$

This gives a generalized eigenvalue problem of size m for which the left-hand-side matrix is Hermitian positive definite. A standard eigenvalue problem can be obtained by requiring that AV be an orthonormal system. In this case, (3.9) becomes

$$V^*A^*Vy = W^*A^{-1}Wy = \frac{1}{\tilde{\lambda}}y. \quad (3.10)$$

Since the matrix W is orthonormal, this leads to the interesting observation that the method is mathematically equivalent to using an orthogonal projection process for computing eigenvalues of A^{-1} . The subspace of approximants in this case is $A\mathcal{K}$. For this reason the approximate eigenvalues $\tilde{\lambda}$ are referred to as *harmonic Ritz values*. Note that one does not have to invert A , but if one maintains the formal relation $W = AV$ by

carrying out the orthogonal transformations on W also on V , then one can use the left-hand side of (3.10) for the computation of the reduced matrix. Since the harmonic Ritz values are Ritz values for A^{-1} (although with respect to a subspace that is generated for A), they tend to be increasingly better approximations for interior eigenvalues (those closest to the origin). One can show, for Hermitian A , that the harmonic Ritz vectors maximize Rayleigh quotients for A^{-1} so that they can be interpreted as the best information that one has for the smallest (in absolute value) eigenvalues. This makes the harmonic Ritz vectors suitable for restarts and this was proposed, for symmetric matrices, by Morgan [331].

The formal introduction of harmonic Ritz values and vectors was given in [349], along with interesting relations between the Ritz pairs and the harmonic Ritz pairs. It was shown that the computation of the projected matrix $W^* A^{-1} W$ can be obtained as a rank-one update of the matrix $V^* A V$, in the case of Krylov subspaces, so that the harmonic Ritz pairs can be generated as cheap side-products of the regular Krylov processes. The generalization of the harmonic Ritz values for more general subspaces was published in [411].

Since the projection of A^{-1} is carried out on a subspace that is generated for A , one should not expect this method to do as well as a projection on a Krylov subspace that has been generated with A^{-1} . In fact, the harmonic Ritz values are increasingly better approximations for interior eigenvalues, but the improvement for increasing subspace can be very marginal (although steady). Therefore, they are in general no alternative for shift-and-invert techniques, unless one succeeds in constructing suitable subspaces, for instance by using cheap approximate shift-and-invert techniques, as in the (Jacobi-) Davidson methods.

We will discuss the behavior of harmonic Ritz values and Ritz value in more detail for the Hermitian case $A^* = A$. Assume that the eigenvalues of A are ordered by magnitude:

$$\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n.$$

A similar labeling is assumed for the approximations $\tilde{\lambda}_i$.

As has been mentioned before, the Ritz values approximate eigenvalues of a Hermitian matrix A “inside out,” in the sense that the rightmost eigenvalues are approximated from below and the leftmost ones are approximated from above, as is illustrated in the following diagram.



This principle can be applied to harmonic Ritz values: since these are the result of an orthogonal projection method applied to A^{-1} , it follows that the harmonic Ritz eigenvalues $1/\lambda_i$ obtained from the process will approximate corresponding eigenvalues $1/\lambda_i$ of A in an inside-out fashion. For positive eigenvalues we have,

$$\tilde{\lambda}_i^{-1} \leq \lambda_i^{-1}; \quad i = 1, 2, \dots \quad \rightarrow \quad \tilde{\lambda}_i \geq \lambda_i; \quad i = 1, 2, \dots$$

Observe that the largest positive eigenvalues are now approximated from above, in contrast with the standard orthogonal projection methods. These types of techniques

were popular a few decades ago as a strategy for obtaining intervals that were certain to contain a given eigenvalue. In fact, as was shown in [349], the Ritz values together with the harmonic Ritz values form the so-called Lehmann intervals, which have nice inclusion properties for eigenvalues of A . In a sense, they provide the optimal information for eigenvalues of A that one can derive from a given Krylov subspace. For example, a lower and upper bound to the (algebraically) largest positive eigenvalue can be obtained by using an orthogonal projection method and a harmonic projection method, respectively.

We conclude our discussion on harmonic Ritz values with the observation that they can be computed also for the shifted matrix $A - \sigma I$, so that one can force the approximations to improve for eigenvalues close to σ .

Refined Projection Methods. To improve the convergence of the standard Rayleigh–Ritz procedure, a so-called *refined projection procedure* has been recently proposed. The idea is to retain the selected Ritz values $\tilde{\lambda}$ that approximate desired eigenvalues, but to discard the Ritz vectors. Instead, we seek a new unit length vector $\tilde{u} \in \mathcal{K}$ that satisfies the optimality condition

$$\|(A - \tilde{\lambda}I)\tilde{u}\|_2 = \min_{u \in \mathcal{K}, \|u\|_2=1} \|(A - \tilde{\lambda}I)u\|_2 \quad (3.11)$$

and to use that vector as a new approximate eigenvector. The vector \tilde{u} is called a *refined Ritz vector* or simply a *refined approximate eigenvector*.

The refined projection procedure is mathematically different from the orthogonal and oblique projection methods described above, because it no longer uses Ritz vectors. In particular, the refined vector is not a Ritz vector if the new residual $(A - \tilde{\lambda}I)\tilde{u} \neq 0$. The refined residual vector $(A - \tilde{\lambda}I)\tilde{u}$ is then not orthogonal to either \mathcal{K} itself or a left subspace \mathcal{L} such as $A\mathcal{K}$.

If a conventional SVD algorithm is used to solve the least squares problem (3.11), the computational cost is $O(nm^2)$ floating point operations, which is too expensive. Fortunately, if $\tilde{\lambda}$ is the Ritz value obtained by orthogonal projection methods, the refined approximate eigenvector \tilde{u} can be computed in $O(m^3)$ floating point operations. Recall that a Rayleigh–Ritz procedure typically requires $O(nm^2)$ floating point operations to produce V and a complete set of m Ritz or harmonic Ritz vectors. Therefore, the cost of computing \tilde{u} is negligible. Hence, the refined approximate eigenvectors provide a viable alternative for approximating eigenvectors.

Several refined projection algorithms have been developed in [244, 245]. Analyses of these methods are presented in [247, 246].

3.3 Spectral Transformations

R. Lehoucq and D. Sorensen

It is well known that the iterative projection methods outlined in the previous section rapidly provide approximations to well-separated extremal eigenvalues. Quite often, however, the wanted eigenvalues may not be well separated or located in the interior of the convex hull of eigenvalues. In these situations, iterative projection methods need many steps to generate acceptable approximations, to these eigenvalues, if they converge at all. An alternative is to employ a spectral transformation of A so that these

poorly separated eigenvalues are transformed to well-separated extremal eigenvalues. Of course, the eigenvalues of A should be easily recoverable from the eigenvalues of the transformed problem, and the transformation should be efficiently applicable in terms of computer time and storage.

The notion of well-separated (or nonclustered) eigenvalues can be roughly explained as follows. A well-separated eigenvalue λ_i is such that $|\lambda_i - \lambda_j| > \tau |\lambda_n - \lambda_1|$ for all $j \neq i$ with $\tau \gg \epsilon_M$. For Hermitian matrices, we suppose that $\lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$, while for non-Hermitian matrices, $\Re(\lambda_1) \leq \Re(\lambda_2) \leq \dots \leq \Re(\lambda_n)$.

The shift-and-invert spectral transformation (SI) is the one typically used. If (λ, x) is an eigenpair for A , $Ax = \lambda x$, and $\sigma \neq \lambda$ then the shift-and-invert eigenvalue problem is

$$(A - \sigma I)^{-1}x = \nu x, \quad \text{where } \nu = \frac{1}{\lambda - \sigma}. \quad (3.12)$$

This SI is effective for finding eigenvalues near σ because the nearby eigenvalues ν_j of $C \equiv (A - \sigma I)^{-1}$ that are largest in magnitude correspond to the nearby eigenvalues λ_j that are nearest to the shift σ . These transformed eigenvalues of largest magnitude are precisely the eigenvalues that are the easiest to compute. Once they are found, they may be transformed back to eigenvalues of the original problem. The direct relation is

$$\lambda_j = \sigma + \frac{1}{\nu_j},$$

and the eigenvector x_j associated with ν_j in the transformed problem is also an (generalized) eigenvector of the original problem corresponding to λ_j . Other spectral transformation techniques include the generalized Cayley transformation; see [324] for further details.

The SI is extremely effective in terms of iteration steps (that is the dimension of the subspace) and should be used whenever possible. This is particularly the case when interior eigenvalues are sought, or when the desired eigenvalues are significantly smaller in magnitude than the eigenvalues largest in magnitude, or when the desired eigenvalues are clustered.

The potential drawback of a spectral transformation is that linear systems involving $A - \sigma I$ must be solved. This can either be accomplished via a matrix factorization or with an iterative method. The user should use a sparse direct method to factor the appropriate matrix whenever feasible. If an iterative method is used for the linear system solves, the accuracy of the solutions must be commensurate with the convergence tolerance used for the eigensolver. A heuristic is that a slightly more stringent tolerance is needed for the iterative linear system solves (relative to the desired accuracy of the eigenvalue calculation). See [283, 291, 414] for a discussion and further references.

It is also possible to carry out the transformation in an approximate way and to work with subspaces that no longer have a Krylov structure. This is the idea behind several algorithms, including the Jacobi–Davidson method. The latter, and other methods, will also be discussed in more detail in the remainder of the book.

Chapter 4

Hermitian Eigenvalue Problems

4.1 Introduction

In this chapter we treat the standard Hermitian, or most often real symmetric, eigenvalue problem (HEP),

$$Ax = \lambda x, \quad (4.1)$$

where $A = A^*$. It is the most commonly occurring algebraic eigenvalue problem, for which we have the most reliable theory and the most powerful algorithms available.

A summary of the basic theory about the Hermitian eigenvalue problem (4.1) is given in §2.2. It is known that the problem (4.1) has n real eigenvalues λ_i , which we may order increasingly so that $\lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$. Note that several eigenvalues may coincide. In many applications the matrix A is positive definite, $\lambda_1 > 0$ (or positive semidefinite, $\lambda_1 \geq 0$). Even in the cases where positive definiteness can be used to advantage, we choose to treat Hermitian A with a general distribution of eigenvalues in this chapter.

When A is Hermitian, it is always possible to find n mutually orthogonal eigenvectors, x_i , $i = 1, \dots, n$, so that

$$A = X\Lambda X^*, \quad (4.2)$$

where $\Lambda = \text{diag}(\lambda_i)$, $X = [x_1, x_2, \dots, x_n]$, and $X^*X = I$. The eigenvectors x_i are not unique; what is unique is the *invariant subspace* for each different eigenvalue. For a Hermitian matrix A , the invariant subspace is of the same dimension as the multiplicity of the eigenvalue. It is important to keep in mind that when certain eigenvalues coincide, their eigenvectors lose their individuality: there is no way of saying that one set of vectors are the eigenvectors of a multiple eigenvalue. Two different algorithms, and even two different runs with the same algorithm, will give different representative eigenvectors in the invariant subspace. On the other hand, a user is entitled to demand that an algorithm produce eigenvector approximations that are orthogonal to each other, even if the matrix has multiple eigenvalues.

The eigenvalues of A are always well-conditioned. If a perturbation E is added to the matrix A , then each of the eigenvalues is perturbed at most as far as the spectral norm $\|E\|_2$,

$$|\lambda_i(A + E) - \lambda_i(A)| \leq \|E\|_2. \quad (4.3)$$

There are several stronger results available, like the Wielandt–Hoffman inequality that says that the sum of squares of the differences between the eigenvalues is majorized by the sum of squares of the elements of E ; see §4.8 and references therein. In many cases, one is interested in eigenvalues that are small compared to the norm $\|A\|$, and for such eigenvalues the inequality (4.3) is not very satisfactory, since it allows large *relative* perturbations for such small eigenvalues. It is possible to develop perturbation bounds for relative perturbations under certain additional assumptions.

There is no result as simple as (4.3) available for bounding the perturbation of *eigenvectors*: one has to add an assumption of separation between the eigenvalues. Let us cite the venerable $\sin \theta$ theorem of Davis and Kahan from [101].

Let us assume that (μ, z) is an approximation of the eigenpair (λ_i, x_i) of A , where z is normalized so that $\|z\|_2 = 1$. The “best” μ corresponding to z is the Rayleigh quotient $\mu = z^* A z$, so we assume that μ has this value. Suppose that μ is closer to λ_i than any other eigenvalues of A , and let δ be the *gap* between μ and any other eigenvalue: $\delta = \min_{\lambda_j \neq \lambda_i} |\mu - \lambda_j|$. Define the residual vector r as $r = Az - \mu z$; then we have

$$\sin \angle(z, x_i) \leq \frac{\|r\|_2}{\delta}. \quad (4.4)$$

Under the same assumptions, we have the improved bound on the eigenvalue approximation,

$$|\mu - \lambda_i| \leq \min \left\{ \|r\|_2, \frac{\|r\|_2^2}{\delta} \right\}. \quad (4.5)$$

The first alternative in this minimum is equivalent to the previous bound (4.3), since μ is an eigenvalue of the perturbed matrix $A + E$ with $E = -rz^*$ a rank-one perturbation of norm $\|E\|_2 = \|r\|_2$ (it is not necessary for E to be Hermitian in the bound (4.3)). The reader is referred to §4.8 for further details.

Overview of Available Algorithms. The reader might remember that eigenvalue problems of moderate size, which in this case means that a full $n \times n$ matrix can be stored conveniently, are solved by *direct methods*, sometimes also called *transformation methods*, where similarity transformations are applied until the eigenvalues can be easily read out.¹ We give a short review of some current direct methods in §4.2.

The main emphasis in this book is on *iterative methods* where sparse matrix storage can be used to advantage. Here the matrix operator is applied to one or a few starting vectors, eigenvalue approximations are computed from subspaces of low dimension, and the iteration is continued until convergence to a subset of the eigenvalues is indicated.

Six basic iterative algorithms will be described in this chapter:

Power method, described in §4.3, is the basic iterative method. It takes a starting vector and lets the matrix operate on it until we get vectors that are parallel

¹Note that “direct” methods must still iterate, since finding eigenvalues is mathematically equivalent to finding zeros of polynomials, for which no noniterative methods can exist. We can call a method *direct* if experience shows that it (nearly) never fails to converge in a fixed number of iterations.

to the leading eigenvector. It converges when there is one unique eigenvalue of largest magnitude, but even in these favorable cases it is slower than other algorithms discussed in this chapter.

Subspace iteration method, sometimes also called *simultaneous iteration*, and described in §4.3, lets the matrix operate on a set of vectors simultaneously, until the iterated vectors span the invariant subspace of the leading eigenvalues. It is the basis of several structural engineering software packages and has a simple implementation and theory of convergence to recommend it. It is, however, slower to converge than algorithms based on Lanczos orthogonalization.

Lanczos method, §4.4, builds up an orthogonal basis of the Krylov sequence of vectors produced by repeated application of the matrix A to a starting vector. In this orthogonal basis, the matrix operator is represented by a tridiagonal matrix T , whose eigenvalues yield Ritz approximations to several of the eigenvalues of the original matrix A . Its main advantage, compared to the power method, is that it yields approximations to several eigenvalues from one sequence of vectors and that these converge after much fewer matrix-vector multiplications. On the other hand, there are potential complications: even if a simple three-term recurrence is enough to give a mathematically orthogonal basis, rounding errors will destroy orthogonality as soon as the first eigenvalue converges, and one has to apply some kind of reorthogonalization. If one needs only eigenvalues, the Lanczos algorithm is economical in storage space. We will describe some variants in §4.4.

Implicitly restarted Lanczos method, §4.5, is a variant where the size of the orthogonal basis is kept bounded and continuously updated, until its subspace contains a prescribed set of eigenvectors. In many cases, it can get the same number of eigenvalues to converge as the original variant of Lanczos, using only marginally more matrix-vector multiplications but significantly less storage space. This variant is especially useful if one needs eigenvectors.

Band Lanczos method, §4.6, is the Lanczos counterpart of the simultaneous power method. For matrices stored on backing storage, or in a memory hierarchy, it is much more efficient to operate on several vectors simultaneously than on one after the other. This is the reason that several *state-of-the-art* structural engineering packages use this kind of algorithm [206]. In multiple-input multiple-output control problems, one is actually interested in reduced-order models based on a given set of starting vectors, which is the main reason to develop the variant described here.

Jacobi–Davidson method, §4.7, is a development of an algorithm that is popular in quantum chemistry computing. It updates a sequence of subspaces, but now by operating with a preconditioned matrix. If one can only use preconditioned iterations to solve linear systems with the underlying matrix, this is the way to get interior eigenvalues or a cluster of eigenvalues.

Summary of Choices. In Table 4.1, we have listed the above algorithms and added some information that may be helpful to decide which algorithm to use in a specific situation.

Table 4.1: Summary of algorithms for HEPs

	Appl	Orth	IE	CE	M	# vec	Fact
Power	Dir		Yes	Very slow	No	2	-
	SI		-	Yes	Yes	2	LU
Subspace iter	Dir	FO	Yes	Slow	No	Moderate	-
	SI	FO	-	Yes	Yes	Moderate	LU
Lanczos	Dir	local	Yes	No	No	3	-
	Dir	SO	Yes	Slow	No	Many	-
	SI	FO	-	Yes	Yes	Moderate	LU
IR Lanczos	Dir	FO	Yes	Slow	No	Few	-
	SI	FO	-	Yes	Yes	Fewer	LU
Band Lanczos	Dir	FO	Yes	Yes	No	Many	-
	SI	FO	-	Yes	Yes	Moderate	LU
Jac-Dav	Dir	FO	Slow	Slow	No	Few	-
	Prec	FO	Yes	Yes	Slow	Few	ILU
	SI	FO	-	Yes	Yes	Few	LU

Matrix Preparation. For each algorithm we have distinguished between different ways of preparing the matrix prior to running the algorithm.

“Dir” is direct application, where we perform a matrix multiply on a vector in each step. It is the simplest variant to apply, since the matrix can be stored in any compact way. On the other hand, most algorithms need many matrix-vector multiplications to converge and are restricted to seeking eigenvalues at the ends of the spectrum.

“SI” is the shift-and-invert which needs a factorization routine to enable solutions of systems $(A - \sigma I)x = b$ for x , but gives the ability to compute a wider choice of eigenvalues in fewer iterations.

“Prec” means application with a preconditioner, for instance, a sparse approximate factorization. This requires less space than the shift-and-invert, but most often it also needs a larger number of matrix-vector multiplies.

Orthogonalization. We have also indicated which type of orthogonalization is needed for the bases of subspaces that most of the algorithms use to find eigenvector approximations. See the algorithm descriptions for details for each algorithm.

Eigenvalues Sought. We give three typical cases:

“IE” stands for one or a few isolated eigenvalues at the end of the spectrum. This is the simplest case, a typical example being the computation of a few leading singular values of a data matrix.

“CE” stands for one or several eigenvalues at the end of the spectrum, possibly clustered, a typical example being the computation of the

lowest eigenmodes of a vibrating mechanical system, modeled by the finite element method.

“M,” finally, means some eigenvalues in the middle of the spectrum, most often in a specified interval, like seeking all eigenmodes of a mechanical or electrical system excited by an external load of a prescribed frequency.

Storage. In the final columns of Table 4.1, we give an estimate of the storage needed.

“#vec” gives how many vectors one needs to store. The meaning of 2 or 3 is obvious; “Moderate” means a multiple of the number m of eigenvalues sought, say $3m$ to $5m$. “Few” is smaller than moderate, say $m + 10$, and “Many” is larger.

“Fact” indicates whether we need extra matrix storage. “LU” means a sparse LU factorization, “ILU,” an incomplete factorization. This is supposed to be more compact than LU decomposition.

To this end, we note that tasks such as counting the number of eigenvalues of A that are smaller than a given real number α or are in a given interval $[\alpha, \beta]$ do not require computing the eigenvalues and so can be performed much more cheaply. The key tool is the matrix inertia. The inertia of A is the triplet of integers $(\nu(A), \zeta(A), \pi(A))$, where $\nu(A)$ is the number of negative eigenvalues of A , $\zeta(A)$ is the number of zero eigenvalues of A , and $\pi(A)$ is the number of positive eigenvalues of A . Sylvester’s law of inertia states that the inertia of a matrix is invariant under congruence; that is, for all nonsingular matrices F , A , and $F^T AF$ have the same inertia. See, for examples, [198, p. 403] or [114, p. 202].

Suppose that the shifted matrix $A - \alpha I$ has the LDL^T factorization

$$A - \alpha I = LDL^T,$$

where D is a diagonal matrix. Then by Sylvester’s law of inertia, $\nu(A - \alpha I) = \nu(D)$. Therefore, the number of negative diagonal elements in D from the LDL^T factorization of $A - \alpha I$ gives the number of eigenvalues of A smaller than α . In the engineering literature, $\nu(A - \alpha I)$ is often called the Sturm sequence number.

It is easy to see that $\nu(A - \beta I) - \nu(A - \alpha I)$ is the number of eigenvalues in the interval $[\alpha, \beta]$, assuming that $\alpha < \beta$ and the two shifted matrices $A - \alpha I$ and $A - \beta I$ are nonsingular. Thus, the cost of counting the number of eigenvalues of A in a given interval $[\alpha, \beta]$ is equivalent to the cost of two LDL^T factorizations of $A - \alpha I$ and $A - \beta I$, respectively. This is much cheaper than finding all eigenvalues in $[\alpha, \beta]$. We refer to §10.3 for the software availability of the LDL^T factorization.

In practice, the counting technique is frequently used as a verification tool for the so-called trust interval of a numerical method for finding all eigenvalues in a given interval $[\alpha, \beta]$.

4.2 Direct Methods

In this section, we briefly discuss direct methods for the computation of eigenvalues of symmetric matrices that can be stored in the computer as full matrices. These

direct methods are sometimes called transformation methods and are built up around similarity transformations.

They are implemented in LAPACK [12], ScaLAPACK [52], and the `eig` command in MATLAB.²

All the subspace-based algorithms we are going to discuss later in this chapter need to use dense, tridiagonal, or banded matrix routines as inner iterations to get Ritz approximations to subspace eigenvalues.

The most common direct methods have two phases:

1. Find an orthogonal matrix Q such that $Q^*AQ = T$ is a tridiagonal matrix.
2. Compute the eigendecomposition of the tridiagonal matrix T .

The initial reduction to tridiagonal form is made by a sequence of $n - 2$ orthogonal Householder reflections and takes $\frac{4}{3}n^3$ floating point operations, or $\frac{8}{3}n^3$ if eigenvectors are also desired. The symmetric or Hermitian driver routines of LAPACK start with this reduction: it is computed by the computational routines `xSYTRD` for the real and `xHETRD` for the complex case (`PxSYTRD` and `PxHETRD` in ScaLAPACK). There are also implementations for packed and banded storage.

There is a choice of methods for finding the eigendecomposition of a tridiagonal matrix:

QR algorithm: This algorithm finds all the eigenvalues and optionally all the eigenvectors. It takes $O(n^2)$ floating point operations for finding all the eigenvalues of a tridiagonal matrix. Since reducing a dense matrix to tridiagonal form costs $\frac{4}{3}n^3$ floating point operations, $O(n^2)$ is negligible for large enough n . For finding all the eigenvectors as well, QR iteration takes a little over $6n^3$ floating point operations on average. It is implemented as `xSTEQR` and `xSTERF` in LAPACK.

This is the algorithm underlying the MATLAB command `eig`.³

Divide-and-conquer method: It divides the tridiagonal matrix into two halves, solves the eigenproblems of each of the halves, and glues the solutions together by solving a special rational equation. It is implemented in LAPACK as `xSTEVD`. `xSTEVD` can be many times faster than `xSTEQR` for large matrices but needs more workspace ($2n^2$ or $3n^2$).

Bisection method and inverse iteration: Bisection may be used to find just a subset of the eigenvalues, say those in an interval $[a, b]$. It needs only $O(nk)$ floating point operations, where k is the number of eigenvalues desired. Thus the bisection method could be much faster than the QR method when $k \ll n$. It can be highly accurate, but may be adjusted to run faster if lower accuracy is acceptable.

Inverse iteration can then be used to find the corresponding eigenvectors. In the best case, when the eigenvalues are well separated, inverse iteration also costs only $O(nk)$ floating point operations. This is much less than either QR or divide-and-conquer, even when all eigenvalues and eigenvectors are desired ($k = n$). On the

²It has been announced that an LAPACK-based numerics library will be part of the next major release of MATLAB; see *The Newsletter for MATLAB, Simulink and Toolbox Users*, Winter 2000, available at <http://www.mathworks.com/company/newsletter/clevescorner/winter2000.cleve.shtml>

³MATLAB checks to see whether the argument of `eig` is symmetric or not and uses the symmetric QR algorithm when appropriate.

other hand, when many eigenvalues are clustered close together, Gram–Schmidt orthogonalization will be needed to make sure that we do not get several identical eigenvectors. This will add $O(nk^2)$ floating point operations to the operation count in the worst case.

Bisection method and inverse iteration are implemented in the LAPACK routines `xSTEBZ` and `xSTEIN`, and they are available as options in `xSYEVX`. In ScaLAPACK, they are subroutines `PxSTEBZ` and `PxSTEIN`.

Relatively robust representation algorithm: This algorithm uses an LDL^T factorization of a number of translates $T - sI$ of T , for one shift s near each cluster of eigenvalues. For each translate the algorithm computes very accurate eigenpairs for the tiny eigenvalues. It is implemented as `xSTEGR` in LAPACK. `xSTEGR` is faster than all the routines except in a few cases and uses the least workspace. See [358, 128, 360].

Finally, a classical method for solving Hermitian eigenvalue problems is the *Jacobi method*. This method constructs an orthogonal transformation to diagonal form,

$$A = X \Lambda X^*$$

by applying a sequence of elementary orthogonal rotations, each time reducing the sum of squares of the nondiagonal elements of the matrix, until it is of diagonal form to working accuracy.

The Jacobi algorithm has been very popular, since it is implemented by a very simple program and gives eigenvectors that are orthogonal to working accuracy. However, it cannot compete with the QR method in terms of operation counts: Jacobi needs $2sn^3$ multiplications for s sweeps ($s = 3$ to 5 usually), which is more than the $4/3n^3$ needed for tridiagonal reduction.

There is one important advantage to the Jacobi algorithm. Properly implemented it can deliver eigenvalue approximations with a small error in the relative sense, in contrast to algorithms based on tridiagonalization, which only guarantee that the error is bounded relative to the norm of the matrix (4.3). See [124].

4.3 Single- and Multiple-Vector Iterations

M. Gu

In this section, we discuss the power method and its variations. These methods are very simple to implement and often find a few extreme eigenvalues quickly. They also serve as a motivation for the more sophisticated methods to be discussed later.

4.3.1 Power Method

The simplest eigenvalue problem is to compute just the dominating eigenvalue along with its eigenvector. The *power method* presented in Algorithm 4.1 is the simplest iterative method for this task. Under mild assumptions it finds the eigenvalue of A which has the largest absolute value, and a corresponding eigenvector.

ALGORITHM 4.1: Power Method for HEP

- (1) start with vector $y = z$, the initial guess
- (2) for $k = 1, 2, \dots$
- (3) $v = y/\|y\|_2$
- (4) $y = Av$
- (5) $\theta = v^* y$
- (6) if $\|y - \theta v\|_2 \leq \epsilon_M |\theta|$, stop
- (7) end for
- (8) accept $\lambda = \theta$ and $x = v$

Let x_1 be the eigenvector corresponding to $\lambda_1 = \lambda_{\max}(A)$. The angle $\angle(z, x_1)$ between x_1 and z is defined by the relation

$$\cos \angle(z, x_1) = \frac{z^* x_1}{\|z\|_2 \|x_1\|_2}.$$

If the starting vector z and the eigenvector x_1 are perpendicular to each other, then $\cos \angle(z, x_1) = 0$. In this case the power method does not converge in exact arithmetic. On the other hand, if $\cos \angle(z, x_1) \neq 0$, the power method generates a sequence of vectors that become increasingly parallel to x_1 . This condition on $\angle(z, x_1)$ is true with very high probability if z is chosen at random.

The convergence rate of the power method depends on $|\lambda_2/\lambda_1|$, where λ_2 is the second largest eigenvalue of A in magnitude. This ratio is generally smaller than 1, allowing adequate convergence. But there are cases where this ratio can be very close to 1, causing very slow convergence. For detailed discussions on the power method, see Demmel [114, Chap. 4], Golub and Van Loan [198], and Parlett [353].

4.3.2 Inverse Iteration

The drawbacks of the power method can be partially overcome by applying the power method to $(A - \sigma I)^{-1}$ instead of A , where σ is called a *shift*. This will let the method converge to the eigenvalue closest to σ , rather than just λ_{\max} . This method is called *inverse iteration*, or the *inverse power method*.

ALGORITHM 4.2: Inverse Iteration for HEP

- (1) start with vector $y = z$, the initial guess
- (2) for $k = 1, 2, \dots$
- (3) $v = y/\|y\|_2$
- (4) $y = (A - \sigma I)^{-1} v$
- (5) $\theta = v^* y$
- (6) if $\|y - \theta v\|_2 \leq \epsilon_M |\theta|$, stop
- (7) end for
- (8) accept $\lambda = \sigma + 1/\theta$ and $x = y/\theta$

Note that there is no need to compute the inverse $(A - \sigma I)^{-1}$ explicitly at step (4). We only have to solve a system like $(A - \sigma I)y = v$ for y efficiently, which formally gives $y = (A - \sigma I)^{-1} v$.

As in the power method, we should expect the sequence of v vectors generated by the inverse power method to become increasingly parallel to the eigenvector corresponding to the largest eigenvalue of $(A - \sigma I)^{-1}$ in magnitude. More specifically, assume that λ_j and x_j are an eigenvalue and eigenvector pair of A so that $|\lambda_j - \sigma|^{-1}$ is the largest eigenvalue of $(A - \sigma I)^{-1}$ in magnitude. The inverse power method converges if z is not perpendicular to x_j . The convergence rate is $|(\lambda_j - \sigma)/(\lambda_k - \sigma)|$, where λ_k is an eigenvalue of A such that $|\lambda_k - \sigma|^{-1}$ is the second largest eigenvalue of $(A - \sigma I)^{-1}$ in magnitude.

One advantage of inverse iteration over the power method is the ability to converge to any desired eigenvalue (the one nearest σ). By choosing σ very close to a desired eigenvalue, inverse iteration can converge very quickly. This method is particularly effective when we have a good approximation to an eigenvalue and only want to compute this eigenvalue and its corresponding eigenvector. However, inverse iteration does require a factorization of the matrix $A - \sigma I$, making it less attractive when this factorization is expensive.

4.3.3 Rayleigh Quotient Iteration

A natural extension of inverse iteration is to vary the shift at each step. It turns out that the best shift which can be derived from an eigenvector approximation $x \neq 0$ is the *Rayleigh quotient* of x , namely, $\rho(x) \equiv x^* A x / x^* x$.

In general, Rayleigh quotient iteration (RQI) will need fewer iterations to find an eigenvalue than inverse iteration with a constant shift; it ultimately has cubical convergence, while inverse iteration converges linearly. However, it is not obvious how to choose the starting vector y to make RQI converge to any particular eigenvalue/eigenvector pair. For example, the RQI can converge to an eigenvalue which is not the closest to the starting Rayleigh quotient $\rho(y)$ and to an eigenvector which is not closest to the starting vector y . Furthermore, there is the tiny but nasty possibility that it may not converge to an eigenvalue/eigenvector pair at all. RQI is more expensive than inverse iteration, requiring a factorization of $A - \rho_k I$ at every iteration, and this matrix will be singular when ρ_k hits an eigenvalue. Hence RQI is practical only if such factorizations can be obtained cheaply at every iteration. See Parlett [353] for more details.

ALGORITHM 4.3: RQI for HEP

- (1) $v = y/\|y\|_2$ and $\rho_1 = \rho(v)$ for the starting vector $y = z$
- (2) **for** $k = 1, 2, \dots$
- (3) $y = (A - \rho_k I)^{-1} v$
- (4) **if** singular, **stop**
- (5) $\theta = \|y\|_2$
- (6) $\rho_{k+1} = \rho_k + y^* v / \theta^2$
- (7) $v = y/\theta$
- (8) **if** $\theta \geq \epsilon_M^{-1/2}$, **stop**
- (9) **end for**
- (10) **accept** $\lambda = \rho_k$ for most recent k and $x = v$

4.3.4 Subspace Iteration

Another important improvement over the power method permits us to compute a ($p > 1$)-dimensional invariant subspace, rather than one eigenvector at a time. It is called *orthogonal iteration* (and sometimes *subspace iteration* or *simultaneous iteration*).

ALGORITHM 4.4: Simple Subspace Iteration for HEP

```
(1)   QR-factorize  $VR = Z$  for the starting matrix  $Z$ 
(2)   for  $k = 1, 2, \dots$ 
(3)        $Y = AV$ 
(4)        $H = V^*Y$ 
(5)       if  $\|Y - VH\|_2 \leq \epsilon_M$ , stop
(6)       QR-factorize  $VR = Y$ 
(7)   end for
```

This algorithm is a straightforward generalization of the power method. The QR factorization is a normalization process that is similar to the normalization used in the power method. The eigenvalues of the Hermitian $p \times p$ matrix $H = V^*AV$ will approach the p eigenvalues of A that are largest in absolute value.

Several modifications are needed to make the simple subspace iteration an efficient and practically applicable code. First, it is natural to orthonormalize as infrequently as possible, i.e., to perform several iterations before performing an orthogonalization. Second, we may choose to operate on a subspace whose dimension p is larger than n_{ev} , the number of eigenvalues wanted, and use a Rayleigh–Ritz process to get eigenvalue approximations. Third, some eigenvalues will converge faster than others, and if this happens it is a good idea to lock these and let the matrix operate only on those vectors that have not yet converged.

In addition, the method is rarely used without some form of acceleration; we describe some of those techniques at the end of this section.

Subspace Dimension. In many cases it is useful to choose p the dimension of the subspace larger than n_{ev} (the number of eigenvalues sought). In that case the slowest eigenvalue converges at a rate $|\lambda_{p+1}/\lambda_{n_{ev}}|$ which may be significantly smaller than $|\lambda_{n_{ev}+1}/\lambda_{n_{ev}}|$. Practitioners in structural engineering have used choices like $p = \min(2n_{ev}, n_{ev} + 8)$; see [42].

Locking. Because of the different rates of convergence of each of the approximate eigenvalues computed by the subspace iteration, it is common practice to extract them one at a time and perform a type of deflation. Thus, as soon as the first eigenvector has converged there is no need to continue to multiply it by A in the subsequent iterations. Indeed, we can freeze this vector and work only with the vectors v_2, \dots, v_m . However, we will still need to perform the subsequent orthogonalizations with respect to the frozen vector v_1 whenever such orthogonalizations are needed. The term used for this strategy is *locking*; that is, we do not further attempt to improve the locked approximation for v_1 .

The following algorithm describes a practical subspace iteration with deflation (locking) for computing the n_{ev} dominant eigenvalues.

ALGORITHM 4.5: Subspace Iteration with Projection and Deflation for HEP

```

(1)    QR-factorize  $V R = Z$  for the starting  $Z$ 
(2)    set  $j = 0$ 
(3)    while  $j \leq n_{\text{ev}}$  do
(4)         $\hat{Y} = [V_j, A^{\text{iter}} V_{p-j}]$ 
(5)        orthonormalize  $\hat{Y} = VR$  (first  $j$  columns will be invariant)
(6)         $H = V_{p-j}^* A V_{p-j}$ 
(7)        compute the eigendecomposition  $H = S \Theta S^*$ 
(8)        test the diagonal entries in  $\Theta$  for eigenvalue convergence
(9)         $V = [V_j, V_{p-j} S]$ 
(10)       set  $j = j + i_{\text{conv}}$ ;  $i_{\text{conv}}$  is the number of newly converged eigenvalues
(11)    end while

```

We now describe some implementation details.

- (1) The initial starting matrix Z should be constructed to be dominant in eigenvector directions of interest in order to accelerate convergence. When no such information is known a priori, a random matrix is as good a choice as any other.
- (4) The iteration parameter iter should be chosen to minimize orthonormalization cost while maintaining a reasonable amount of numerical accuracy. The amplification factor $(\lambda_1/\lambda_p)^{\text{iter}}$, where the eigenvalues λ_i are ordered in decreasing absolute values, gives the loss of accuracy. Rutishauser [381] plays it safe and allows an amplification factor of only 10, losing one decimal, while Stewart and Jennings [426] let the algorithm run to $\epsilon_M^{-1/2}$, half the machine accuracy, but not to more than 10 iterations.

Acceleration. If eigenvalues near a *shift* σ are desired and a factorization $A - \sigma I = LU$ can be easily obtained (see §10.3), then one can apply the above algorithm to $(A - \sigma I)^{-1}$. The eigenvalues near σ will converge fast.

One can also use *polynomial acceleration* to speed up the computation by replacing the power A^m by a polynomial $T_m[(A - \sigma I)/\rho]$, in which T_m is the Chebyshev polynomial of the first kind of degree m , and σ and ρ give a translation and scaling of the part of the spectrum one wants to suppress. Ideally one should take $\sigma = (\lambda_{p+1} + \lambda_n)/2$ as the center and $\rho = (\lambda_{p+1} - \lambda_n)/2$ as the half-width of the interval containing the eigenvalues that are *not* of interest for some reasonable estimates of those eigenvalues. We assume that the eigenvalues are ordered along the real axis and that we want p of them in one of the ends.

With these enhancements, subspace iteration may be a reasonably efficient method that has the advantage of being easy to code and to understand. Some of the methods to be discussed later are often preferred, however, because they tend to find eigenvalues/eigenvectors more quickly.

Much of the material in this section is drawn from Demmel [114], Golub and Van Loan [198], and Saad [387]. For further discussion on subspace iteration, the reader is recommended to refer to Chatelin [79], Lehoucq and Scott [292], Stewart [422], and Wilkinson [457]. See also Bathe and Wilson [42] and Jennings [242] for structural engineering approaches.

4.3.5 Software Availability

There are several pieces of software available for subspace iteration. EA12 is a routine by Duff and Scott for symmetric subspace iteration [143]. It is part of the Harwell Subroutine Library. Given a real (sparse) symmetric matrix, this routine computes the r largest eigenvalues and their corresponding eigenvectors. There is a classical implementation of subspace iteration by Rutishauser [382] in [458], where most of the refinements discussed here have been included.

There are also two subspace iteration-based software packages for the nonsymmetric eigenvalue problem, SRRIT by Bai and Stewart [37] and LOPSI by Stewart and Jennings [427] (see §7.4 for more details). These packages can be used to solve the symmetric eigenvalue problem, but with some loss in efficiency and accuracy.

For more information about this software, including how to access it, see the book's homepage ETHOME.

4.4 Lanczos Method

A. Ruhe

The Lanczos algorithm is closely related to the iterative algorithms discussed in the previous section in that it needs to access the matrix only in the form of matrix-vector operations. It is different in that it makes much better use of the information obtained by remembering all the directions computed and always lets the matrix operate on a vector orthogonal to all those previously tried.

In this section we describe the Hermitian Lanczos algorithm applicable to the eigenvalue problem,

$$Ax = \lambda x , \quad (4.6)$$

where A is a Hermitian, or in the real case symmetric, matrix operator.

The algorithm starts with a properly chosen starting vector v and builds up an orthogonal basis V_j of the Krylov subspace,

$$\mathcal{K}^j(A, v) = \text{span}\{v, VA, A^2v, \dots, A^{j-1}v\} , \quad (4.7)$$

one column at a time. In each step just one matrix-vector multiplication

$$y = Ax \quad (4.8)$$

is needed. In the new orthogonal basis V_j the operator A is represented by a real symmetric tridiagonal matrix,

$$T_j = \begin{bmatrix} \alpha_1 & \beta_1 & & \\ \beta_1 & \alpha_2 & \ddots & \\ \ddots & \ddots & \ddots & \beta_{j-1} \\ & \beta_{j-1} & \alpha_j & \end{bmatrix} , \quad (4.9)$$

which is also built up one row and column at a time, using the basic recursion,

$$AV_j = V_j T_j + r e_j^* \quad \text{with} \quad V_j^* r = 0 . \quad (4.10)$$

At any step j , we may compute an eigensolution of T_j ,

$$T_j s_i^{(j)} = s_i^{(j)} \theta_i^{(j)}, \quad (4.11)$$

where the superscript (j) is used to indicate that these quantities change for each iteration j . The Ritz value $\theta_i^{(j)}$ and its Ritz vector,

$$x_i^{(j)} = V_j s_i^{(j)}, \quad (4.12)$$

will be a good approximation to an eigenpair of A if the residual has small norm; see §4.8.

Let us compute the residual for this Ritz pair,

$$r_i^{(j)} = Ax_i^{(j)} - x_i^{(j)}\theta_i^{(j)} = AV_j s_i^{(j)} - V_j s_i^{(j)}\theta_i^{(j)} = (AV_j - V_j T_j)s_i^{(j)} = v_{j+1}\beta_j s_{j,i}^{(j)}.$$

We see that its norm satisfies

$$\|r_i^{(j)}\|_2 = |\beta_j s_{j,i}^{(j)}| = \beta_{j,i}, \quad (4.13)$$

so we need to monitor only the subdiagonal elements β_j of T and the last elements $s_{i,j}^{(j)}$ of its eigenvectors to get an estimate of the norm of the residual. As soon as this estimate is small, we may flag the Ritz value $\theta_i^{(j)}$ as converged to the eigenvalue λ_i . Note that the computation of the Ritz values does not need the matrix-vector multiplication (4.12). We can save this time-consuming operation until the step j , when the estimate (4.13) indicates convergence.

4.4.1 Algorithm

We get the following algorithm.

ALGORITHM 4.6: Lanczos Method for HEP

- (1) start with $r = v$, starting vector
- (2) $\beta_0 = \|r\|_2$
- (3) **for** $j = 1, 2, \dots$, **until** convergence,
- (4) $v_j = r/\beta_{j-1}$
- (5) operate $r = Av_j$
- (6) $r = r - v_{j-1}\beta_{j-1}$
- (7) $\alpha_j = v_j^* r$
- (8) $r = r - v_j \alpha_j$
- (9) reorthogonalize if necessary
- (10) $\beta_j = \|r\|_2$
- (11) compute approximate eigenvalues $T_j = S\Theta^{(j)}S^*$
- (12) test bounds for convergence
- (13) **end for**
- (14) compute approximate eigenvectors $X = V_j S$

Let us comment on some of the steps of Algorithm 4.6:

- (1) If a good guess for the wanted eigenvector is available, use it. For instance, if, for a discretized partial differential equation, it is known that the wanted eigenvector is smooth with respect to the grid, one could start with a vector with all ones. In other cases choose a random direction, for instance, one consisting of normally distributed random numbers.
- (5) This matrix-vector operation is most often the CPU-dominating work. Any routine that performs a matrix-vector multiplication can be used. In the shift-and-invert case, solve systems with the factored matrix. There is, as of now, little solid experience of using iterative equation solvers in the inverted case.
- (9) When computing in finite precision, orthogonality is lost as soon as one eigenvalue converges. The choices for reorthogonalization are:
 1. *Full*: Simple to implement; see §4.4.4 below. Iterations will need successively more work as j increases. This is the preferred choice in the shift-and-invert case, when several eigenvalues converge in few steps j .
 2. *Selective*: The most elaborate scheme, necessary when convergence is slow and several eigenvalues are sought. This option is also discussed later in §4.4.4.
 3. *Local*: Used for huge matrices, when it is difficult to store the whole basis V_j . We make sure that v_{j+1} is orthogonal to v_{j-1} and v_j by doing an extra orthogonalization against these two vectors, subtracting $r = r - v_{j-1}(v_{j-1}^* r)$ and $r = r - v_j(v_j^* r)$ once in this step. Still we need to detect and discard spurious eigenvalue approximations using the Cullum device; see §4.4.4.
- (11) For each step j , or at appropriate intervals, compute the eigenvalues $\theta_i^{(j)}$ and eigenvectors $s_i^{(j)}$ of the tridiagonal matrix T_j (4.9).

If fast convergence is expected, as for shift-and-invert problems, j will be much smaller than n ; then this computation is very fast and standard software from, e.g., LAPACK [12] is recommended.

If the Lanczos algorithm is applied directly to A and a large tridiagonal matrix is computed, there are special tridiagonal eigenvalue algorithms based on bisection and a Givens recursion for eigenvectors; see [356, 358, 128] and §4.2.

- (12) The algorithm is stopped when a sufficiently large basis V_j has been found, so that eigenvalues $\theta_i^{(j)}$ of the tridiagonal matrix T_j (4.9) give good approximations to all the eigenvalues of A sought.

The estimate $\beta_{j,i}$ (4.13) for the residual may be too optimistic if the basis V_j is not fully orthogonal. Then the Ritz vector $x_i^{(j)}$ (4.12) may have a norm smaller than 1, and we have to replace the estimate (4.13) by

$$\|r_i^{(j)}\| = |\beta_j s_{i,j}^{(j)}| / \|V_j s_i^{(j)}\| .$$

- (14) The eigenvectors of the original matrix A are computed only when the test in step (12) has indicated that they have converged. Then the basis V_j is used in a matrix-vector multiplication to get the eigenvector (4.12),

$$x_i^{(j)} = V_j s_i^{(j)},$$

for each i that is flagged as having converged.

The great beauty of this type of algorithm is that the matrix A is accessed only in a matrix-vector operation in step (5) in Algorithm 4.6. Any type of sparsity and any storage scheme can be taken advantage of.

It is necessary to keep only three vectors, r , v_j , and v_{j-1} , readily accessible; older vectors can be left on backing storage. There is even a variant where only two vectors are needed (see [353, Chap. 13.1]), but it demands some dexterity to code it. Except for the matrix-vector multiplication, only scalar products and additions of a multiple of a vector to another are needed. We recommend using Level 1 BLAS routines; see §10.2.

4.4.2 Convergence Properties

There is a beautiful theory, based on the properties of orthogonal polynomials, that describes when eigenvalues converge. The characteristic polynomials of the tridiagonal matrices T_j are orthogonal polynomials with respect to a scalar product, defined by the expansion of the starting vector v as a sum of eigenvectors. One gets bounds on differences $|\theta_i^{(j)} - \lambda_i|$ between the eigenvalues $\theta_i^{(j)}$ of T_j and λ_i of A by replacing these unknown orthogonal polynomials with the well-known Chebyshev polynomials, the so-called Kaniel–Paige–Saad bounds; see [353].

This theory says that we get convergence to those eigenvalues that are represented in the starting vector and faster convergence to those in the ends of the spectrum. The better separated these are from the rest of the eigenvalues, the faster will they converge.

In practical cases, we are often interested just in the lowest eigenvalues, and fortunately these are among the first to converge. On the other hand, the relative separation of the lowest eigenvalues is often poor, since the separation is relative to the whole spread of the spectrum, not to the distance to the origin.

Multiple Eigenvalues. In theory, we get convergence only to eigenvectors that are represented in the starting vector. Normally this is not of any great concern, since rounding errors will also introduce those directions that were not present at the outset into the computation. However, in one case this is important, namely, when the matrix A has multiple eigenvalues. In that case, we will get only one vector from the multidimensional invariant subspace of this multiple eigenvalue. Note that any vector in the subspace is equally valid: there is no way for this, or any other algorithm, to choose a special vector and each Lanczos run with a different starting vector will yield a new suggestion for an eigenvector to a multiple eigenvalue.

There are two different ways to get a set of linearly independent eigenvectors to a multiple eigenvalue. The first is to restart and run a projected operator on the subspace orthogonal to the converged eigenvector(s), where all the converged eigendirections have been projected away. This corresponds to orthogonalizing the vector r in step (9) of

Algorithm 4.6 to all converged eigenvectors. This procedure is repeated as long as new vectors converge; see, e.g., [318]. There is a second, more radical way to deal with possibly multiple eigenvalues: the *block Lanczos* method. It starts with several, say p , starting directions, forming a block V_1 , and lets A operate on all of V_j in step j , to compute a new orthogonal block V_{j+1} . The matrix T will be a block-tridiagonal, or more properly band matrix; see the description in §4.6.

In the block Lanczos method, the convergence is governed by the separation of the desired eigenvalue λ_i from other p eigenvalues away in the spectrum, say λ_{i+p} , if we assume the eigenvalues sorted from smallest to largest. However, the degree of the polynomial is j , not jp , as we would have if we ran single-vector Lanczos for the same number of basis vectors, and this will slow down the convergence in the general case.

Regardless of the above objection, block Lanczos is advantageous for efficiency reasons, whenever computing $Y = AX$ for an $n \times p$ matrix X is much cheaper than computing $y = Ax$ for p vectors, one after the other. This is the case for most machines with cache memories and when the matrices are so large that they need to be stored in secondary storage.

4.4.3 Spectral Transformation

If the extreme eigenvalues are not well separated and when we want interior eigenvalues, it is greatly advantageous to replace A by a shift-and-invert operator

$$C = (A - \sigma I)^{-1} \quad (4.14)$$

for an appropriately chosen shift σ , for instance, in the interval $\alpha \leq \sigma \leq \beta$, where we are interested in knowing the eigenvalues. The shift-and-invert operator C has the eigenvalues

$$\theta_i = \frac{1}{\lambda_i - \sigma} \quad \text{or} \quad \lambda_i = \sigma + \frac{1}{\theta_i}, \quad (4.15)$$

and now the eigenvalues θ_i that correspond to eigenvalues λ_i close to the shift σ will be at the ends of the spectrum and well separated from the rest; see [162].

If we use a shift-and-invert operator, we start the algorithm by factoring

$$LDL^* = P^T(A - \sigma I)P, \quad (4.16)$$

using some appropriate sparse Gaussian elimination scheme. Here, P is a permutation and L is unit lower triangular. If there are eigenvalues λ at both sides of the shift σ , we cannot use a scalar diagonal D , but we have to make a symmetric indefinite factorization, as in MA47 of Duff and Reid [141]. Here D is block diagonal with one by one and two by two blocks, and we get the inertia of $A - \sigma I$ as a by-product. We can get a count of the number of eigenvalues in an interval by recording the inertia of $(A - \sigma I)$ for two values σ at the ends of the interval. Obtaining this count is used to make sure that no multiple eigenvalues are missed. See §10.3 for further information about sparse matrix factorizations.

During the actual iteration, we use the factors P , L , and D , computing

$$r = P(L^{-*}(D^{-1}(L^{-1}(P^T v_j)))) , \quad (4.17)$$

in the order indicated by the parentheses, in step (5) of Algorithm 4.6.

4.4.4 Reorthogonalization

The Lanczos recursion is constructed to make the basis V orthogonal, but this is true only for infinite precision computation. In the algorithm we only make sure that the new vector v_{j+1} is orthogonal to working precision to the two latest vectors v_{j-1} and v_j , and orthogonality to the earlier vectors follows from the symmetry of A and the recursion (4.10). As soon as one eigenvalue converges, i.e., the Ritz pair has a small residual (4.13), all the basis vectors v_j get perturbations in the direction of the eigenspace of the converged eigenvalue. As a result of this, a duplicate copy of that eigenvalue will soon show up in the tridiagonal matrix T . Paige [347] was the first to discover this; the reader is referred to the monograph [353] for a detailed discussion.

Let us consider three different strategies to handle this.

Full Reorthogonalization. We could play it safe and do a *full reorthogonalization* either by modified or classical Gram–Schmidt, in the latter case computing the orthogonalization coefficients in a vector

$$h = V_j^* v_{j+1}$$

and then subtracting

$$v'_{j+1} = v_{j+1} - V_j h$$

to get the improved vector v'_{j+1} . If the norm is decreased by a nontrivial amount, say $\|v'_{j+1}\| < \frac{1}{\sqrt{2}} \|v_{j+1}\|$, this will have to be repeated, but this will hardly ever happen when treating properly computed Lanczos vectors.

Selective Reorthogonalization. As the size j of the basis grows, full reorthogonalization will need substantial arithmetic work, and we look for a more economical scheme. It can be proved that most of the desired properties of the Lanczos algorithm are preserved as long as the basis is *semiorthogonal*, i.e., orthogonal to half the machine precision,

$$W_j = V_j^* V_j = I_j + E, \quad \|E\| < \sqrt{\epsilon_M}. \quad (4.18)$$

The reader is referred to the investigation by Simon [402] for a detailed discussion of these matters. It is proved that if the tridiagonal matrix T is computed by a Lanczos algorithm with a semiorthogonal basis V_j (4.18), then T is a fully accurate projection of A on the subspace spanned by the computed basis V_j ,

$$T_j = N_j^* A N_j + G, \quad \|G\| = O(\epsilon_M \|A\|).$$

Here N_j is an orthogonal basis of the subspace spanned by V_j . Even if N_j is not known explicitly, we know that the eigenvalues of T_j are accurate approximations to those of A .

One way of making sure that the computed basis is semiorthogonal is to use a recurrence derived by Paige to monitor how losses of orthogonality in earlier steps are propagated to later steps and to do a reorthogonalization when this recurrence indicates that a threshold of $\sqrt{\epsilon_M}$ is exceeded. The recurrence is between successive elements $w_{j,k}$ of the product matrix W_j (4.18) and is derived from the basic recursion (4.10) in the following way.

First we note that the computed vector v_{j+1} satisfies (4.10):

$$\beta_j v_{j+1} = Av_j - \alpha_j v_j - \beta_{j-1} v_{j-1} + f_j,$$

where f_j is a vector of rounding errors. To get elements $\omega_{j,k}$ of the product matrix W_j (4.18), we premultiply this with v_k^* and get

$$\beta_j \omega_{j+1,k} = v_k^* Av_j - \alpha_j \omega_{j,k} - \beta_{j-1} \omega_{j-1,k} + v_k^* f_j.$$

This multiplication with indices j and k exchanged gives

$$\beta_k \omega_{j,k+1} = v_j^* Av_k - \alpha_k \omega_{j,k} - \beta_{k-1} \omega_{j,k-1} + v_j^* f_k,$$

and subtracting the two gives

$$\beta_j \omega_{j+1,k} = \beta_k \omega_{j,k+1} + (\alpha_k - \alpha_j) \omega_{j,k} + \beta_{k-1} \omega_{j,k-1} - \beta_{j-1} \omega_{j-1,k} - v_j^* f_k + v_k^* f_j. \quad (4.19)$$

Note that the terms involving A cancel out since $v_k^* Av_j = v_j^* Av_k$ for a Hermitian matrix A .

We use this recursion to fill the lower triangular part of the symmetric matrix W one row at a time. The diagonal elements $\omega_{k,k} = 1$ and the elements closest above and below the diagonal $\omega_{k-1,k} = \omega_{k,k-1} = O(\epsilon_M)$ at the rounding error level, due to symmetry and the explicit orthogonalization. This gives starting values for the recursion (4.19). We estimate the absolute value of the sum of the two last terms in (4.19) by $2\epsilon_M \|A\|$, with an appropriate guess for the norm of A , and feed these values, that also are at rounding error level, into the recursion with the sign chosen so that the absolute value of the new $|\omega_{j+1,k}|$ is maximized,

$$\begin{aligned} \tilde{\omega} &= \beta_k \omega_{j,k+1} + (\alpha_k - \alpha_j) \omega_{j,k} + \beta_{k-1} \omega_{j,k-1} - \beta_{j-1} \omega_{j-1,k}, \\ \omega_{j+1,k} &= (\tilde{\omega} + \text{sign}(\tilde{\omega}) 2\epsilon_M \|A\|) / \beta_j. \end{aligned}$$

As soon as one element $\omega_{j+1,k}$ exceeds the $\sqrt{\epsilon_M}$ sized threshold, we orthogonalize the two current vectors v_j and v_{j+1} to all the previous vectors v_k , $k = 1, \dots, j$, and put the corresponding $\omega_{j,k}$ and $\omega_{j+1,k}$ down at the rounding error level. We need to orthogonalize two vectors because of the three-term recurrence in the Lanczos method, and as a consequence in the ω recursion (4.19). It is not necessary to store more than the two latest rows, $j-1$ and j , of the matrix W of ω estimates.

The above is one of the techniques described in [402]. There is another method that uses explicit orthogonalization only to those vectors v_k for which $\omega_{j+1,k}$ exceeds the threshold. This is what Simon [402] called *partial* reorthogonalization, but the variant described here, *periodic* reorthogonalization, which was first described by Grcar [204], is simpler to implement and can use Level 2 BLAS operations.

Local Reorthogonalization and Detecting Spurious Ritz Values. If we choose, or are forced to, use only local reorthogonalization, some of the eigenvalues of the tridiagonal matrix T will be new copies of already converged eigenvalues and we will also get *spurious* eigenvalues of T . Such an eigenvalue occurs suddenly at a certain step j only to disappear at the next step.

Cullum [90] has devised a way to weed out such extra copies and spurious values. She takes the tridiagonal matrix T_j and another \widehat{T}_2 , which is obtained from T_j by

deleting the first row and column. All eigenvalues of T_j that are very close to eigenvalues of \widehat{T}_2 need special consideration. If such an eigenvalue is a multiple eigenvalue of T_j , keep one of them and discard the rest as copies, remembering that an unreduced tridiagonal matrix by definition has only simple eigenvalues. If a simple eigenvalue of T_j is also an eigenvalue of \widehat{T}_2 , it is spurious and should be discarded.

4.4.5 Software Availability

There are a few software packages available for the Lanczos methods described in this section. One, LANCZOS, was published by Cullum and Willoughby [91, 92]. It uses the Lanczos method without reorthogonalization but contains a mechanism to detect and discard spurious eigenvalue approximations. Scott developed LASO, a block Lanczos code with selective orthogonalization, as described in the pioneering paper [363]. The package LANZ was written by Jones and Patrick [249] and includes the SI (4.14) and uses selective (partial) reorthogonalization, as described in §4.4.4 here.

Finally there is a code, LANSO, a FORTRAN package that implements selective orthogonalization as developed by the group led by Parlett at UC Berkeley. The presentation of this section follows the ideas behind LANSO very closely. LANSO makes periodic reorthogonalization as recommended in §4.4.4 and calls a standard tridiagonal QL algorithm to compute eigenvalue approximations (4.11) and estimate residuals (4.13), at appropriate intervals. PLANSO is a parallel version developed by Wu and Simon [462].

For more information about these codes, including how to access them, see the book's homepage, ETHOME.

4.4.6 Numerical Examples

We consider the Lanczos algorithm applied to two simple but typical examples using the LANSO software.

The first example is the computation of some eigenmodes of an L-shaped membrane. The standard five-point finite difference approximation with a grid spacing $h = 1/64$ gives a sparse symmetric positive definite matrix of order $n = 2945$. It will have a very regular sparse band structure with at most five nonzeros elements in each row. Its eigenvalues will be in the interval $0 < \lambda_i < 8$, symmetrically distributed around $\lambda = 4$ and more densely distributed towards the ends of the spectrum.

The second test example is taken from an information retrieval application and involves a term document matrix, where each column stands for one document and each row for one term. The element $x_{i,j}$ is 1 if term i occurs in document j , zero otherwise. The space of a set of leading singular values can be used to find connections between some of the documents. The specific matrix MEDLINE is rectangular of size 7014×1033 and moderately sparse with 53,287 filled elements, or about 8 elements in each row. It is not a good idea to form the product $A = X^T X$ explicitly, since this matrix will be nearly full with 910,755 elements (or 85%) nonzero. We implement the product $y = Ax$ by first computing $z = Xx$ followed by $y = X^T z$.

Results for L-Shaped Membrane. The extremal eigenvalues are the first to converge for a direct application of a Lanczos algorithm, and in this case the convergence will be very similar in either end of the spectrum.

We have plotted the estimated residuals (4.13) for the six largest eigenvalues as a function of the number j of Lanczos steps in Figure 4.1. The curves show the residual estimates (4.13) for each step and were computed only for illustration purposes after the actual computation. The LANSO algorithm called the QL algorithm to compute eigenvalues and last elements of eigenvectors to test for convergence (4.13), at the iterations marked with dashdotted vertical lines in the plot. The selective orthogonalization triggered reorthogonalization at the steps we marked with dashed lines, altogether only three times during all these 300 steps. This is typical for situations with a slow convergence: orthogonality is preserved until the first Ritz value converges.

Note that a relatively large number of steps, about 150, are needed to bring the residual of the leading eigenvalue down to 10^{-4} , and that another 150 steps are taken before full machine accuracy is reached. Looking at the Ritz values $\theta_i^{(j)}$ of (4.11) plotted versus j in Figure 4.2, we see that the largest Ritz value grows with j until around step $j = 60$, when it stabilizes. Frequently a Ritz value will start to approach one eigenvalue, but will later move to another, not yet found, eigenvalue. This happens with the third Ritz value at steps 60 to 80 and becomes more pronounced with the sixth between steps 150 and 210. This phenomenon shows up in a less evident fashion in Figure 4.1, where the third and sixth curves from the bottom have plateaus at the steps when a Ritz value shifts allegiance. A user of a direct Lanczos method is advised to take care when deciding whether all the largest eigenvalues really have converged.

Results for Medline SVD. The second test example, Medline SVD, behaves quite differently. The leading eigenvalues ($\lambda_i(A) = \sigma_i(X)^2$) are quite well separated (the

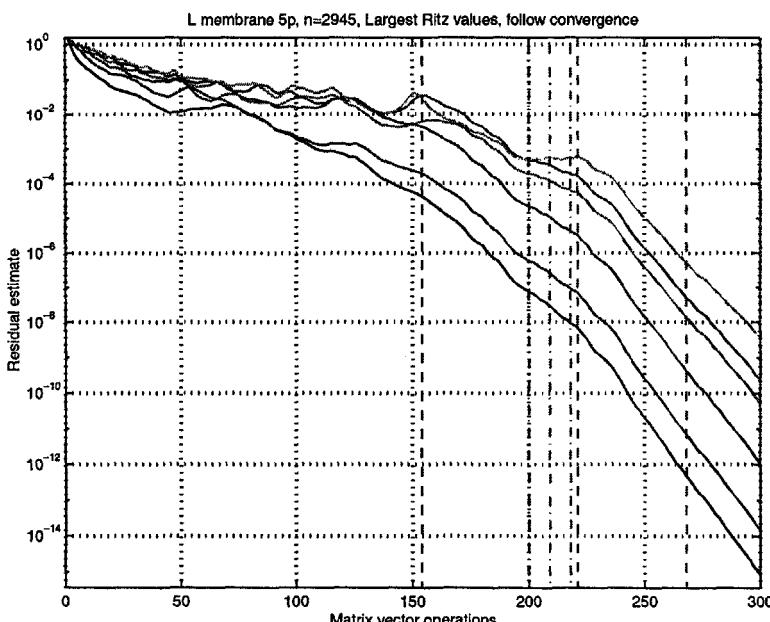


Figure 4.1: Residual estimates, L-shaped membrane matrix.

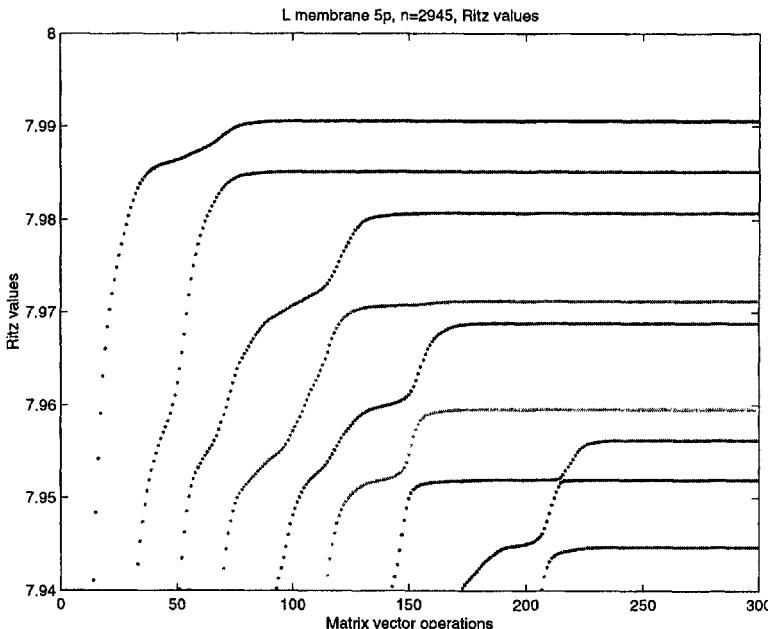


Figure 4.2: Ritz values, L-shaped membrane matrix.

largest one is $\lambda_1 = 3442.5$ and the next one is $\lambda_2 = 756.6$), and even if the quotients λ_i/λ_{i+1} are smaller for larger i , we will get fast convergence as indicated in Figure 4.3. The first eigenvalue reaches full accuracy already at step $j = 14$, and after $j = 50$ steps the first 6 eigenvalues are converged. After $j = 300$ steps we had 100 eigenvalues.

There is another interesting difference between this well-separated problem and the L-shaped membrane with its more clustered eigenvalues in that reorthogonalization is triggered more often. We mark reorthogonalizations with a dashed vertical line at step $j = 7$, one at $j = 12$, and every four steps from there on. Since each reorthogonalization involves two vectors, selective reorthogonalization demands about half as much work as a full reorthogonalization. In such cases the user is advised to use full reorthogonalization, since it gives full orthogonality of the basis vectors at a moderate extra cost in arithmetic work.

Results for L-Shaped Membrane with Shift-and-Invert. To illustrate the effect of applying a Lanczos algorithm to a shift-and-invert operator (4.14), we follow the convergence for the L-shaped membrane matrix with a shift applied at the origin $\sigma = 0$. The picture in Figure 4.4 is now very similar to the Medline SVD example, and we get full accuracy in the six smallest eigenvalues after $j = 36$ steps. In this run we used full reorthogonalization as advocated above. Even if the number of steps j is reduced by more than a factor of 10, we have to take into consideration that the factorization (4.16) takes some time and that the factors are denser than the original A , so applying their inverses in each step (4.17) needs more work than the application of the original A of (4.8) in the direct Lanczos method.

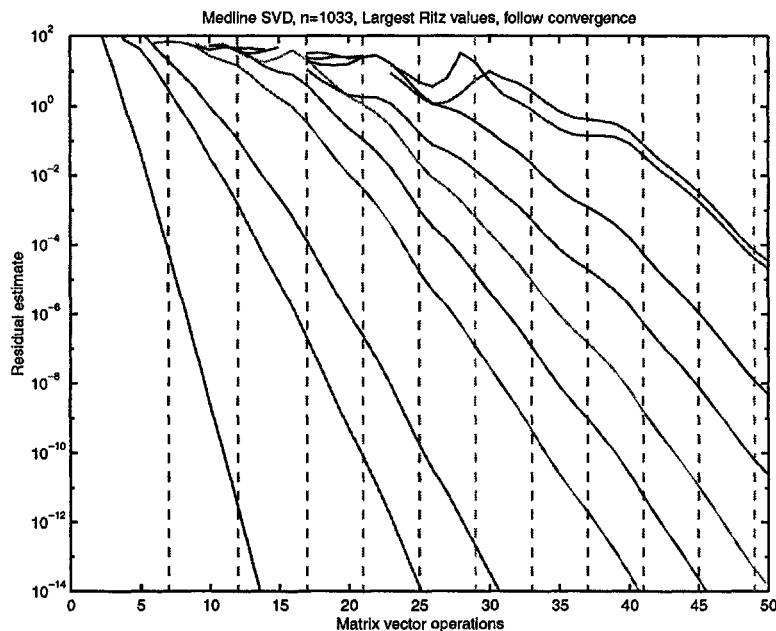


Figure 4.3: Residual estimates, Medline SVD matrix

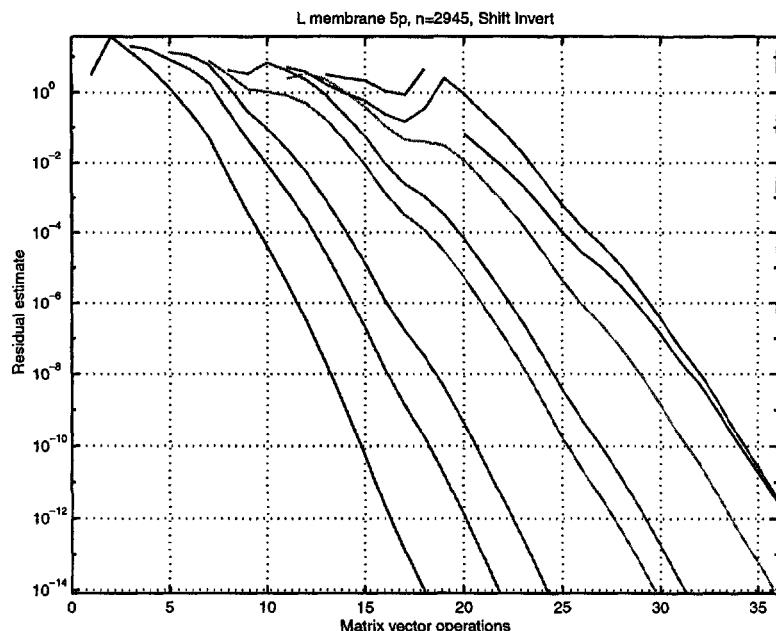


Figure 4.4: Residual estimates, shift-and-invert L-shaped membrane matrix.

4.5 Implicitly Restarted Lanczos Method

R. Lehoucq and D. Sorensen

The Lanczos process for a Hermitian matrix $A = A^*$ has been derived previously in §4.4. Here, we discuss how to apply implicit restart. Our starting point is a k -step Lanczos factorization (4.10):

$$AV_k = V_k T_k + r_k e_k^*,$$

where $V_k \in \mathbb{C}^{n \times k}$ has orthonormal columns, $V_k^* r_k = 0$, and $T_k \in \mathcal{R}^{k \times k}$ is real, symmetric, and tridiagonal with nonnegative subdiagonal elements. The columns of V_k are referred to as the *Lanczos vectors*. For implicit restart it is important that the columns of V_k be made orthogonal to full accuracy.

4.5.1 Implicit Restart

An unfortunate aspect of the Lanczos process is that there is no way to determine in advance how many steps will be needed to determine the eigenvalues of interest within a specified accuracy. It is determined by the distribution of the eigenvalues $\lambda(A)$ and the choice of starting vector v_1 . In many cases, convergence will not occur until k gets very large. Maintaining the numerical orthogonality of V_k quickly becomes intractable for large k at a cost of $O(nk^2)$ floating point operations.

There is another way to maintain orthogonality: one may limit the size of the basis set and use restarting schemes. Restart means replacing the starting vector v_1 with an “improved” starting vector v_1^+ and computing a new Lanczos factorization with the new vector. The structure of the residual vector r_k serves as a guide: In theory, r_k will vanish if v_1 is a linear combination of k eigenvectors of A . Our goal is to make this happen iteratively. In the symmetric case, these eigenvectors are orthogonal and hence form an orthonormal basis for an invariant subspace.

The need for restart was recognized early on by Karush [258] soon after the appearance of the original algorithm of Lanczos [285]. Subsequently, there were developments by Paige [347], Cullum and Donath [89], and Golub and Underwood [197]. All of these schemes are *explicit* in the sense that a new starting vector is produced by some process and an entirely new Lanczos factorization is constructed.

In this section we will describe *implicit restart*. It is a technique to combine the implicitly shifted QR scheme with a k -step Lanczos factorization and obtain a truncated form of the implicitly shifted QR iteration. The numerical difficulties and storage problems normally associated with the Lanczos process are avoided. The algorithm is capable of computing a few (k) eigenvalues with user-specified features such as the algebraically largest or smallest eigenvalues or those of largest magnitude, using storage for only a moderate multiple of k vectors. The computed eigenvectors form a basis for the desired k -dimensional eigenspace and are numerically orthogonal to working precision.

Implicit restart provides a means to extract interesting information from large Krylov subspaces while avoiding the storage and numerical difficulties associated with the standard approach. It does this by continually compressing the interesting information into a fixed-size k -dimensional subspace. This is accomplished through the

implicitly shifted QR mechanism. A Lanczos factorization of length $m = k + p$,

$$AV_m = V_m T_m + r_m e_m^*, \quad (4.20)$$

is compressed to a factorization of length k that retains the eigeninformation of interest. This is accomplished by applying p implicitly shifted QR steps. The first stage of this shift process results in

$$AV_m^+ = V_m^+ T_m^+ + r_m e_m^* Q, \quad (4.21)$$

where $V_m^+ = V_m Q$, $T_m^+ = Q^* T_m Q$, and $Q = Q_1 Q_2 \cdots Q_p$. Each Q_j is the orthogonal matrix associated with the shift μ_j used during the shifted QR algorithm. Because of the Hessenberg structure of the matrices Q_j , it turns out that the first $k - 1$ entries of the vector $e_m^* Q$ are zero. This implies that the leading k columns in equation (4.21) remain in a Lanczos relation. Equating the first k columns on both sides of (4.21) provides an updated k -step Lanczos factorization

$$AV_k^+ = V_k^+ T_k^+ + r_k^+ e_k^*,$$

with an updated residual of the form $r_k^+ = V_m^+ e_{k+1} \beta_k + r_m Q(m, k)$. Using this as a starting point, it is possible to apply p additional steps of the Lanczos process to return to the original m -step form.

A template for the implicitly restarted Lanczos method (IRLM) is given in Algorithm 4.7.

ALGORITHM 4.7: IRLM for HEP

- (1) start with $v_1 = v / \|v\|$ with starting vector v
- (2) compute an m -step Lanczos factorization

$$AV_m = V_m T_m + r_m e_m^*$$
- (3) repeat until convergence ($T_k = D_k$ diagonal)
 - (4) compute $\sigma(T_m)$ and select p shifts $\mu_1, \mu_2, \dots, \mu_p$
 - (5) initialize $Q = I_m$
 - (6) for $j = 1, 2, \dots, p$,
 - (7) QR-factorize $Q_j R_j = T_m - \mu_j I$
 - (8) update $T_m = Q_j^* T_m Q_j$, $Q = Q Q_j$
 - (9) end for
 - (10) $r_k = v_{k+1} \beta_k + r_m \sigma_k$, with $\beta_k = T_m(k+1, k)$ and $\sigma_k = Q(m, k)$
 - (11) $V_k = V_m Q(:, 1:k)$; $T_k = T_m(1:k, 1:k)$
 - (12) beginning with the k -step Lanczos factorization

$$AV_k = V_k T_k + r_k e_k^*,$$

apply p additional steps of the Lanczos process to obtain
a new m -step Lanczos factorization,

$$AV_m = V_m T_m + r_m e_m^*$$
 - (13) end repeat

We will now describe some implementation details, referring to the respective phases in Algorithm 4.7.

- (1) Ideally, for eigenvalue calculations, one should attempt to construct a starting vector v that is dominant in eigenvector directions of interest. If there is a sequence

of closely related eigenvalue problems, the solution of the previous problem may provide a starting vector for the next. In the absence of any other considerations, a random vector is a reasonable choice.

- (2) Algorithm 4.8 below may be used to compute this initial ($m = k + p$)-step Lanczos factorization.
- (4) The shifts μ_j are selected with respect to the user's "wanted set" specification and the current and perhaps past information about the spectrum of T_m ; see §4.5.2 below.
- (6)–(9) Apply p steps of the shifted QR iteration to T_m using the $\mu_1, \mu_2, \dots, \mu_p$ as shifts. This should be done using the implicitly shifted QR variant (i.e., the "bulge chase" mechanism). If exact shifts are used, then $\beta_k = T_m(k+1, k)$ should be zero on completion of the p steps of QR, and the leading submatrix $T_m(1:k, 1:k)$ should have $\theta_1, \theta_2, \dots, \theta_k$ as its eigenvalues.
- (10) When exact shifts are used, the properties mentioned in steps (6)–(9) are usually approximately true in finite precision. However, they might not be achieved due to rounding errors. Therefore, it is important to include both terms in the update of the residual vector $r_k \leftarrow v_{k+1}\beta_k + r_m\sigma_k$. Note that, with exact shifts, the updated V_k and T_k will provide a new k -step Lanczos factorization with Ritz values and vectors that are the best approximations to the user specification that has been produced so far.
- (12) This step requires a slight modification of the Lanczos process discussed previously. It begins with step k of the usual scheme and assumes that all k Lanczos vectors have been retained in a matrix V_k . The details are given in Algorithm 4.8 below.

4.5.2 Shift Selection

There are many ways to select the shifts $\{\mu_j\}$ that are applied by the QR steps. Virtually any explicit polynomial restart scheme could be applied through this implicit mechanism. Considerable success has been obtained with the choice of *exact shifts*. This selection is made by sorting the eigenvalues of T_m into two disjoint sets of k "wanted" and p "unwanted" eigenvalues and using the p unwanted ones as shifts. With this selection, the p shift applications result in T_k^+ having the k wanted eigenvalues as its spectrum. As convergence takes place, the subdiagonals of T_k tend to zero and the most desired eigenvalue approximations appear as eigenvalues on the diagonal of the leading $k \times k$ block of T_m . The basis vectors V_k tend to be orthogonal eigenvectors of A .

Examples of the "wanted set" specification are:

the k algebraically smallest eigenvalues,

the k algebraically largest eigenvalues,

the k eigenvalues with largest magnitude,

the k eigenvalues with smallest magnitude.

When choosing between these alternatives, remember that eigenvalues in the ends of the spectrum converge first, so any choice to avoid these may lead to slow convergence. SI is recommended when seeking interior eigenvalues.

Other interesting choices of shifts $\{\mu_j\}$ include the roots of Chebyshev polynomials [383], harmonic Ritz values [331, 337, 349, 411], the roots of Leja polynomials [23], the roots of least squares polynomials [384], and refined shifts [244]. In particular, the Leja and harmonic Ritz values have been used to estimate the interior eigenvalues of A .

An alternate interpretation stems from the fact that each of these shift cycles results in the implicit application of a polynomial in A of degree p to the starting vector:

$$v_1 \leftarrow \psi(A)v_1 \quad \text{with} \quad \psi(\lambda) = \prod_{j=1}^p (\lambda - \mu_j). \quad (4.22)$$

The roots of this polynomial are the shifts used in the QR process and these may be selected to enhance components of the starting vector in the direction of eigenvectors corresponding to desired eigenvalues and damp the components in unwanted directions. Of course, this is desirable because it forces the starting vector into an invariant subspace associated with the desired eigenvalues. This in turn forces r_k to become small, and hence convergence results. Full details may be found in [419].

There is an alternative way to apply implicit restart, *thick restart*, as described by Wu and Simon [463]. There an eigenvalue factorization of T_m (4.20) is computed, and the part that corresponds to the *wanted* eigenvalues is kept as an arrow matrix and the rest is discarded. Thick restart is mathematically equivalent to exact shift IRLM, if the same choice of wanted and unwanted eigenvalues is made.

4.5.3 Lanczos Method in GEMV Form

At each restart in Algorithm 4.7 we have a k -step Lanczos factorization

$$AV_k = V_k T_k + r_k e_k^*$$

or, for $k = 0$ a starting vector r_0 . Algorithm 4.8 gives a template for applying p additional Lanczos steps to extend this to an m -step Lanczos factorization

$$AV_m = V_m T_m + r_m e_m^*.$$

We will now describe some implementation details, referring to the respective phases in Algorithm 4.8.

- (2) If started from scratch ($k = 0$) take $r_0 = v$ as a starting vector. Ideally, for eigenvalue calculations, one should attempt to construct a v that is dominant in eigenvector directions of interest. In the absence of any other considerations, a random vector is a reasonable choice.
- (3) Normalize r to get the new basis vector v_j . The norm $\beta_{j-1} = \|r\|$ has already been computed at step (7) for the previous j .
- (5) This is the Lanczos three-term recurrence step to orthogonalize r with respect to the two most recent columns of V_j . It is organized in modified Gram–Schmidt form.

ALGORITHM 4.8: *m*-Step Lanczos Factorization

```

(1)      start with  $r = r_k$ , previous residual or starting vector,  $\beta_k = \|r_k\|$ 
(2)      for  $j = k + 1, \dots, m$ 
(3)           $v_j = r/\beta_{j-1}$ 
(4)           $r = Av_j - v_{j-1}\beta_{j-1}$ 
(5)           $\alpha_j = v_j^*r$ 
(6)           $r = r - v_j\alpha_j$ 
(7)          if ( $\|r\| < \rho\sqrt{\alpha_j^2 + \beta_{j-1}^2}$ )
(8)               $s = V_j^*r$ 
(9)               $r = r - V_js$ 
(10)              $\alpha_j = \alpha_j + s_j, \beta_j = \beta_j + s_{j-1}$ 
(11)         end if
(12)     end for

```

- (7) The sequence of statements in this *if* clause assure that the new residual direction r is numerically orthogonal to the previously computed directions, i.e., to all of the columns of V_j . The parameter ρ must be specified ($0 \leq \rho < \infty$). The test asks the question, "Is $r = Av_j$ nearly in the space that already has been constructed?"

The ratio $\|r\|/\sqrt{\alpha_j^2 + \beta_{j-1}^2} = \tan \theta$, where θ is the angle that the vector r makes with the range space of V_j . A larger value of ρ is a more stringent orthogonality requirement. With $\rho = 0$, the algorithm reverts to the standard Lanczos process without reorthogonalization. One step of this correction may not be sufficient and it should be implemented as an iteration with the test for subsequent corrections being $\|r\| < \rho\|s\|$; see the discussion in [96]. If a suitable r has not been generated after a fixed number of attempts (say five), then β_j should be set to zero and a randomly generated vector should be orthogonalized against the basis V_j and put in place of r to continue the factorization. We suggest setting $\rho = 1$.

It is well known that the classical three-term Lanczos scheme will fail to produce orthogonal vectors precisely when a Ritz value (approximate eigenvalue) converges to an eigenvalue of A . To remedy this, we have included an iterative refinement technique which maintains orthogonality to full working precision at a very reasonable cost. The special situation imposed by implicit restart makes this modification essential for obtaining accurate eigenvalues and numerically orthogonal eigenvectors. The implicit restart mechanism will be less effective and may even fail if numerical orthogonality is not maintained.

4.5.4 Convergence Properties

There is a fairly straightforward intuitive explanation of how this repeated updating of the starting vector v_1 through implicit restart might lead to convergence. If v_1 is expressed as a linear combination of eigenvectors $\{x_j\}$ of A , then

$$v_1 = \sum_{j=1}^n x_j \gamma_j \quad \Rightarrow \quad \psi(A)v_1 = \sum_{j=1}^n x_j \psi(\lambda_j) \gamma_j.$$

Applying the same polynomial (i.e., using the same shifts) repeatedly for ℓ iterations will result in the j th original expansion coefficient being attenuated by a factor

$$\left(\frac{\psi(\lambda_j)}{\psi(\lambda_1)} \right)^\ell,$$

where the eigenvalues have been ordered according to decreasing values of $|\psi(\lambda_j)|$. The leading k eigenvalues become dominant in this expansion and the remaining eigenvalues become less and less significant as the iteration proceeds. Hence, the starting vector v_1 is forced into an invariant subspace as desired. The adaptive choice provided with the exact shift mechanism further enhances the isolation of the wanted components in this expansion, and the wanted eigenvalues are approximated better and better as the iteration proceeds.

It is worth noting that if $m = n$, then $r_m = 0$ and this iteration is precisely the same as the implicitly shifted QR iteration. Even for $m < n$, the first k columns of V_m and the leading $k \times k$ tridiagonal submatrix of T_m are mathematically equivalent to the matrices that would appear in the full implicitly shifted QR iteration using the same shifts μ_j . In this sense, the IRLM may be viewed as a truncation of the implicitly shifted QR iteration. The fundamental difference is that the standard implicitly shifted QR iteration selects shifts to drive subdiagonal elements of T_n to zero from the bottom up while the shift selection in the implicitly restarted Lanczos method is made to drive subdiagonal elements of T_m to zero from the top down. Of course, convergence of the implicit restart scheme here is like a “shifted power” method, while the full implicitly shifted QR iteration is like an “inverse iteration” method.

Thus the exact shift strategy can be viewed both as a means to damp unwanted components from the starting vector and also as directly forcing the starting vector to be a linear combination of wanted eigenvectors. See [419] for information on the convergence of IRLM and [22, 421] for other possible shift strategies for Hermitian A . The reader is referred to [293, 334] for studies comparing implicit restart with other schemes.

4.5.5 Computational Costs and Tradeoffs

The implicit scheme costs p rather than the $k + p$ matrix-vector products the explicit scheme would require to apply the same p -degree polynomial and rebuild the k -step Lanczos factorization. When the operation $r = Av$ is relatively expensive, this savings can be considerable. However, there are tradeoffs. It is impossible to predict optimal values for the number of extra vectors p in general.

The convergence behavior depends to a great deal on the distribution of the spectrum of A . Two matrices with precisely the same sparsity pattern could exhibit completely different convergence characteristics for a given choice of k and p . Therefore, it would be misleading to draw any conclusions based solely on operation counts per iteration. Nevertheless, it can be useful to have a notion of the major costs of an iteration to guide initial choices.

In the following, we assume that the cost of a matrix-vector product $r = Av$ is γn . For a sparse matrix, one could think of γ as twice the average number of nonzero entries per row of A . For a dense matrix $\gamma = 2n$, and for a fast Fourier transform (FFT) matrix $\gamma \approx \log_2(n)$. See §10.2. For a fixed value of k and p , the cost (in floating point operations) for one iteration of IRLM is

- matrix-vector products $r = Av$: γpn ;
- basic Lanczos steps: $9pn$;
- orthogonalization corrections: roughly $4(k + p)n$ per correction. A worst-case estimate is $4(kp + p^2)n$ per IRLM iteration (assumes one correction per each new Lanczos vector);
- implicit restart: $2n(k^2 + kp) + O((k + p)^3)$;
- total cost: $\gamma pn + (6k + 9)pn + 4p^2n + 2k^2n + O((k + p)^3)$.

If the matrix-vector product Av is cheap (i.e., γ is quite small), the cost of implicit restart is significant and alternatives should be considered. One of these might be to use a polynomial spectral transformation. This would involve operating with a suitably constricted polynomial function of $\psi(A)$ in place of A . An example would be a Chebyshev polynomial designed to damp the unwanted portion of the spectrum. The action $r = \psi(A)v$ would, of course, be applied as a succession of matrix-vector products involving A .

4.5.6 Deflation and Stopping Rules

Deflation is an important concept in the practical implementation of the QR iteration and therefore equally important to the IRLM. In the context of the QR iteration, deflation amounts to setting a small subdiagonal (and corresponding superdiagonal) element of the tridiagonal matrix T to zero. This is called deflation because it splits the tridiagonal matrix into two smaller subproblems which may be independently refined further.

However, in the context of IRLM, the usual QR deflation techniques are not always appropriate. Additional deflation capabilities specific to implicit restart are needed. It is often desirable to deflate at a coarser accuracy level ϵ_D with $1 > \epsilon_D > \epsilon_M$, where ϵ_M is machine precision. In theory (i.e., in exact arithmetic), when A has multiple eigenvalues it would be impossible for IRLM to compute more than one eigenvector to such a multiple eigenvalue. This is because it is a “single vector” rather than a block method. However, in practice, there is usually little difficulty in computing multiple eigenvalues, because the method deflates itself as convergence takes place, and roundoff usually introduces components in new eigenvector directions in the subsequent starting vectors. Nevertheless, this can be unreliable and miss a multiple eigenvalue. In any case, this approach typically will require a stringent convergence tolerance in order to find all instances of a multiple eigenvalue.

It is far more efficient to deflate (i.e., lock) an approximate eigenvalue once it has converged to a certain level of accuracy and to force subsequent Lanczos vectors to be orthogonal to the converged subspace. With this capability, additional instances of a multiple eigenvalue can be computed to the same specified accuracy without the expense of forcing them to converge to unnecessarily high accuracy.

In the Lanczos process, as with a QR iteration, it is possible for some of the leading k subdiagonals to become small during the course of implicit restart. However, it is usually the case that there are converged Ritz values appearing in the spectrum of T long before small subdiagonal elements appear. This convergence is usually detected through observation of a small last component in an eigenvector y of T . When this

happens, we are able to construct an orthogonal similarity transformation of T that will give an equivalent Lanczos factorization with a slightly perturbed T that does indeed have a zero subdiagonal, and this is the basis of our deflation scheme.

In the context of IRLM, there are two types of deflation required:

Locking: If a Ritz value θ has converged (meaning $\|Ax - x\theta\| < \epsilon_D$) and is thought to be a member of the wanted set of eigenvalues, we wish to declare it converged, decouple the eigenpair (x, θ) , and continue to compute remaining eigenvalues with no further alteration of x or θ . This process is called locking.

Purging: If a Ritz value θ has converged but is not a member of the wanted set, we wish to decouple and remove the eigenpair (x, θ) from the current Krylov subspace spanned by the Lanczos vectors. This process is called purging.

Techniques for this deflation were first developed in [289, 294]. The scheme we present here is based on an improvement developed in [420]. This scheme allows for stable and efficient deflation (or locking) of Ritz values that have converged with a specified relative accuracy of ϵ_D which may be considerably larger than machine precision ϵ_M . This is particularly important when a SI is not available to accelerate convergence. Typically in this setting the number of matrix-vector products will be large, and it will be highly desirable to lock converged Ritz values at low tolerances. This will avoid the expense of the matrix-vector products that would be required to achieve an accuracy that would allow normal QR-type deflation. Also, it is important to be able to purge converged but unwanted Ritz values. As pointed out in [289], the forward instability of the QR bulge chase process discovered by Parlett and Le [359] will prevent implicit restart being used for purging converged unwanted Ritz values.

4.5.7 Orthogonal Deflating Transformation

We shall utilize a special orthogonal transformation to implement the deflation schemes mentioned above. The deflation schemes are related to an eigenvector associated with a Ritz value that is to be deflated (either locked or purged). Given a vector y of unit length, the algorithm shown in Algorithm 4.9 computes an orthogonal matrix Q such that $Qe_1 = y$ (hence $e_1 = Q^*y$). This orthogonal matrix has a very special form and may be written as

$$Q = R + ye_1^*, \quad \text{with } Re_1 = 0, \quad R^*y = 0, \quad (4.23)$$

where R is upper triangular. It may also be written as

$$Q = L + yg^*, \quad \text{with } Le_1 = 0, \quad L^*y = e_1 - g, \quad (4.24)$$

where L is lower triangular and $g^* \equiv e_1^* + \frac{1}{\eta_1}e_1^*R$. Here we assume that $y^T = (\eta_1, \dots, \eta_n)$.

Now, consider the matrix Q^*TQ . The substitutions $Q^* = (L + yg^*)^*$, $Q = (R + yfe_1^*)$ from (4.24) and (4.23) and the facts $Q^*Ty = \theta e_1$ and $1 = y^*y$ will give

$$\begin{aligned} Q^*TQ &= Q^*T(R + ye_1^*) \\ &= (L^* + gy^*)TR + \theta e_1 e_1^* \\ &= L^*TR + gy^*TR + \theta e_1 e_1^*. \end{aligned}$$

Since both L^* and R are upper triangular, it follows that L^*TR is upper Hessenberg with the first row and the first column each being zero due to $Le_1 = Re_1 = 0$. Also, $gy^*TR = g\theta y^*R = 0$. From this we conclude that L^*TR must also be symmetric and hence tridiagonal. Therefore, we see that $T_+ \equiv Q^*TQ$ is of the form

$$T_+ = \begin{bmatrix} \theta & 0 \\ 0 & \widehat{T} \end{bmatrix},$$

where \widehat{T} is symmetric and tridiagonal.

It should be noted that, as computed by Algorithm 4.9, Q will have componentwise relative errors on the order of machine precision ϵ_M with no element growth.

ALGORITHM 4.9: Orthogonal Deflating Transformation for IRLM

- ```

(1) start with the vector y of dimension k and $\|y\| = 1$
(2) $Q = 0$, $Q(:, 1) = y$
(3) $\sigma = y(1)^2$, $\tau_o = |y(1)|$
(4) for $j = 2, \dots, k$
(5) $\sigma = \sigma + y(j)^2$, $\tau = \sqrt{\sigma}$
(6) if $\tau_o \neq 0$
(7) $\gamma = (y(j)/\tau)/\tau_o$
(8) $Q(1 : j - 1, j) = -y(1 : j - 1)\gamma$
(9) $Q(j, j) = \tau_o/\tau$
(10) else
(11) $Q(j - 1, j) = 1$
(12) end if
(13) $\tau_o = \tau$
(14) end for

```

**Locking or Purging a Single Eigenvalue.** The orthogonal transformations developed in the previous section will provide stable and efficient transformations needed to implement locking and purging.

**Locking  $\theta$ .** The first instance to discuss is the locking of a single converged Ritz value. Assume that

$$Ty = y\theta, \quad \|y\| = 1,$$

with  $e_k^*y = \eta$ , where  $|\eta| \leq \epsilon_D\|T\|$ . Here, it is understood that  $\epsilon_M \leq \epsilon_D < 1$  is a specified relative accuracy tolerance between  $\epsilon_M$  and 1.

If  $\theta$  is “wanted,” it is desirable to lock  $\theta$ . However, in order to accomplish this, it will be necessary to arrange a transformation of the current Lanczos factorization to one with a small subdiagonal to isolate  $\theta$ . This may be accomplished by constructing a  $k \times k$  orthogonal matrix  $Q = Q(y)$  using Algorithm 4.9:

$$Qe_1 = y \text{ and } e_k^*Q = (\eta, \tau e_{k-1}^*),$$

with  $\eta^2 + \tau^2 = 1$ .

The end result of these transformations is

$$\begin{aligned} Av_1 &= v_1\theta + r\eta, \quad \text{where } v_1^*r = 0, \\ AV_2 &= V_2T_2 + r\tau e_{k-1}^*, \end{aligned}$$

where  $[v_1, V_2] = VQ$ .

This means that subsequent implicit restart takes place as if

$$AV_2 = V_2T_2 + r\tau e_{k-1}^*$$

with all the subsequent orthogonal transformations associated with implicit restart applied to  $T_2$  and never disturbing the relation  $Av_1 = v_1\theta + r\eta$ . In subsequent Lanczos steps,  $v_1$  participates in the orthogonalization so that the selective orthogonalization recommended by Parlett and Scott [363, 353] is accomplished automatically.

**Purging  $\theta$ .** If  $\theta$  is “unwanted,” we may wish to remove  $\theta$  from the spectrum of the projected matrix  $T$ . However, the implicit restart strategy using exact shifts will sometimes fail to purge a converged unwanted Ritz value [294].

In the symmetric case, the purging process is essentially the same as the locking process just described. However, instead of keeping the deflated Ritz vector and value, they are simply discarded. After this, we are left with a  $(k - 1)$ -step factorization

$$AV_2 = V_2T_2 + r\tau e_{k-1}^*,$$

with  $[v_1, V_2] = VQ$ . No error other than an acceptable level of roundoff will be introduced through purging.

Observe that there is no requirement that  $y$  be an accurate eigenvector for  $T$ . It is only necessary for the residual  $y^*TQ = \theta e_1^*$  to meet a componentwise accuracy condition that we shall discuss in the next subsection. Moreover, there is no need for the last element  $\eta$  of the eigenvector to be small in the case of purging. However, when it is not small, the implicit restart mechanism with exact shifts will suffice to purge  $\theta$ . Finally, this procedure is valid in complex arithmetic with minor notational modifications.

**Stability of  $Q^*TQ$ .** The deflating orthogonal transformation construction shown in Algorithm 4.9 is clearly stable (i.e., a componentwise relatively accurate representation of the transformation one would obtain in exact arithmetic). There is no question that the similarity transformation  $Q^*TQ$  preserves the eigenvalues of  $T$  to full numerical accuracy. However, there is a serious question about how well the tridiagonal form is preserved during locking and/or purging. A modification to the basic algorithm is needed to assure that if  $Ty = y\theta$ , then  $T_+ \equiv Q^*TQ$  is symmetric and numerically tridiagonal (i.e., the entries outside the tridiagonal band are all tiny relative to  $\|T\|$ ).

The fact that  $T_+ = Q^*TQ$  is tridiagonal depends upon the term  $gy^*TR$  vanishing in the expression

$$Q^*TQ = L^*TR + gy^*TR + \theta e_1 e_1^*.$$

However, on closer examination, we see that

$$e_1^* Q = e_1^* L + (e_1^* y) g^* = \eta_1 g^*,$$

where  $\eta_1$  is the first component of  $y$ . Therefore,  $\|g\| = \frac{1}{|\eta_1|}$ . So there may be numerical difficulty when the first component of  $y$  is small. To be specific,  $y^* T = \theta y^*$  and thus  $y^* T R = 0$  in exact arithmetic. However, in finite precision, the computed  $\text{fl}(y^* T) = \theta y^* + z^*$ . The error  $z$  will be on the order of  $\epsilon_M$  relative to  $\|T\|$ , but

$$\|gy^* TR\| = \|g\| \cdot \|z^* R\| = \frac{1}{|\eta_1|} \|z^* R\|,$$

so this term may be quite large. It may be as large as order  $O(1)$  if  $\eta_1 = O(\epsilon_M)$ . This is of serious concern and will occur in practice without the modification we now propose.

The remedy is to introduce a step-by-step acceptable perturbation and rescaling of the vector  $y$  to simultaneously force the conditions

$$Q^* y = e_1 \quad \text{and} \quad y^* T Q = \theta e_1^*$$

to hold with sufficient accuracy in finite precision. To accomplish this, we shall devise a scheme to achieve

$$y^* T q_j = 0 \quad \text{for } j > 1$$

numerically. As shown in [420], this modification is sufficient to establish  $q_i^* T q_j = 0$  numerically, relative to  $\|T\|$  for  $i > j + 1$ .

The basic idea is as follows: If, at the  $j$ th step, the computed quantity  $\text{fl}(y^* T q_j)$  is not sufficiently small, it is adjusted to be small enough by scaling the vector  $y_j$  with a number  $\phi$  and the component  $\eta_{j+1}$  with a number  $\psi$ . With this rescaling just prior to the computation of  $q_{j+1}$ , we have  $y_j \leftarrow y_j \phi$  and  $\widehat{y}_j \leftarrow \widehat{y}_j \psi$ , where  $y^* = [y_1^*, \widehat{y}_1^*, y_2^*, \dots]$ . Certainly,  $\|y\|$  should not be altered with this scaling and this is therefore required. This gives the following system of equations to determine  $\phi$  and  $\psi$ : If  $|y_j^* T_j \widehat{q}_j + \rho_j \beta_j \eta_{j+1}| > \epsilon_M \tau_{j+1}$ ,

$$\begin{aligned} (\tau_j \phi)^2 + (1 - \tau_j^2) \psi^2 &= 1, \\ y_j^* T_j \widehat{q}_j \phi + \rho_j \beta_j \eta_{j+1} \psi &= \pm \epsilon_M \tau_{j+1}. \end{aligned}$$

If  $\rho_j$  is on the order of  $\sqrt{\epsilon_M}$ , the scaling may be absorbed into  $\rho_j$  without alteration of  $y$  and also without affecting the numerical orthogonality of  $Q$ . When  $y$  is modified, it turns out that none of the previously computed  $q_i$ ,  $2 \leq i < j$ , need to be altered. After step  $j$ , the vector  $y_j$  is simply rescaled in subsequent steps, and the formulas defining  $q_i$ ,  $2 \leq i \leq j$ , are invariant with respect to scaling of  $y_j$ . For complete detail, one should consult [420].

Algorithm 4.10 implements this scheme to compute an acceptable  $Q$ .<sup>4</sup> In practice, this transformation may be computed and applied in place to obtain a  $T \leftarrow Q^* T Q$  and  $V \leftarrow V Q$  without storing  $Q$ . However, that implementation is quite subtle and the construction of  $Q$  is obscured by the details required to avoid additional storage. This code departs slightly from the above description since the vector  $y$  is rescaled to have unit norm only after all of the columns 2 to  $m$  have been determined.

---

<sup>4</sup>There is now a much better algorithm for constructing and applying these transformations; see D. Sorensen, Deflation for Implicitly Restarted Arnoldi Methods, CAAM Technical Report, TR 98-12, Rice University, 1998 (revised August 2000).

**ALGORITHM 4.10: Stable Orthogonal Deflating Transformation for IRLM**

```

(1) start with m -vector y with $\|y\| = 1$ and $\eta = e_m^* y$
 and T symmetric tridiagonal of order m with $Ty = \theta y$
(2) $Q = 0$
(3) while $y(i) = 0$, $y(i) = \epsilon_M/m$, $i := i + 1$, end while
(4) $\sigma = y(1)^2$, $\tau_o = |y(1)|$
(5) for $j = 2, \dots, m$
(6) $\sigma = \sigma + y(j)^2$, $\tau = \sqrt{\sigma}$
(7) $\gamma = -(y(j)/\tau)/\tau_o$
(8) $\rho = \tau_o/\tau$
(9) $Q(1 : j - 1, j) = y(1 : j - 1)\gamma$, $Q(j, j) = \rho$
(10) if $(j < m \text{ and } \tau < .05)$
(11) $\tau_p = \sqrt{\sigma + y(j+1)^2}$
(12) $\alpha = y(1:j)^* T(1:j, 1:j) Q(1:j, j)$
(13) $\delta = y(j+1) T(j, j+1) \rho$
(14) $\sigma_o = -1$
(15) if $\text{sign}(\alpha) \cdot \text{sign}(\delta) < 0$, $\sigma_o = 1$, end if
(16) if $|\delta + \alpha| > \epsilon_M \tau_p$
(17) if $\rho < \sqrt{\epsilon_M}/100$
(18) $Q(j, j) = Q(j, j) \sigma_o (\epsilon_M \tau_p + |\alpha|) / |\delta|$
(19) else
(20) if $|\alpha| > |\delta|$
(21) $\phi = \sigma_o (\epsilon_M \tau_p + |\delta|) / |\alpha|$
(22) $y(1:j) = y(1:j) \phi$
(23) $\sigma = \sigma \phi^2$, $\tau = \tau |\phi|$
(24) else
(25) $\psi = \sigma_o (\epsilon_M \tau_p + |\alpha|) / |\delta|$
(26) $y(j+1:m) = y(j+1:m) \psi$
(27) end if
(28) end if
(29) end if
(30) end if
(31) $\tau_0 = \tau$
(32) end for
(33) $Q(:, 1) = y / \|y\|$

```

There are several implementation issues.

- (3) The perturbation  $y(i)$  shown here avoids problems with exact zero initial entries in the eigenvector  $y$ . In theory, this should not happen when  $T$  is unreduced but it may happen numerically if  $\theta$  is weakly represented in the starting vector. There is a cleaner implementation possible that does not modify zero entries. This is the simplest (and crudest) correction.
- (10) As soon as  $\tau_j = \|y_j\|$  is sufficiently large, there is no need for any further corrections. Deleting this if-clause reverts to the unmodified calculation of  $Q$  shown in Algorithm 4.9.

- (16) This shows one of several possibilities for modifying  $y$  to achieve the desired goal of numerically tiny elements outside the tridiagonal band of  $Q^*TQ$ . More sophisticated strategies would absorb as much of the scaling as possible into the diagonal element  $Q(j,j) = \rho$ . Here, the branch that does scale  $\rho$  instead of  $y$  is designed so that  $\text{fl}(1 + (\rho\psi)^2) = \text{fl}(1 + \rho^2) = 1$ .

#### 4.5.8 Implementation of Locking and Purging

Although the basic deflation ideas of locking and purging are conceptually straightforward, there are still a number of implementation details and various strategies one might consider. In the end, some ad-hoc decisions will have to be made. The deflation strategy adopted here is somewhat conservative. However, the performance appears to be quite reasonable in practice. In [420], computational examples demonstrate that very crude tolerances can be specified without missing multiple or clustered eigenvalues.

The implementation details become quite involved and it is difficult to convey the ideas by displaying code. Instead, we shall indicate the main ideas of the strategy with a fairly high-level verbal description.

**ALGORITHM 4.11: Deflation for IRLM**

- (1) Lock a single Ritz value  $\theta$  each time one converges ( $\|Ax - x\theta\| < \epsilon_D$ ) until  $k$  values have been locked.
- (2) Continue to iterate and lock each newly converged Ritz value that is a “better” value than the existing ones. Follow each locking operation with a purge operation to delete the least wanted but locked Ritz value. Thus, there are always  $k$  locked Ritz values and vectors in place.
- (3) Continue step (2) until the next Ritz value to converge is not a “better” value. Replace the  $(k+1)$ st basis vector with a randomly generated one and orthogonalize this against the previous ones and then build a new ( $m = k+p$ )-step Lanczos factorization using Algorithm 4.8. Repeat step (2).
- (4) When step (3) has been executed two consecutive times with no replacement of existing locked Ritz values, the iteration is halted.

Implementation notes:

- (1) Initially, we work with an ( $m = k+p$ )-step Lanczos factorization and apply  $p$  shifts per each implicit restart. Each time a Ritz value is locked, it is advantageous to decrease the effective value of  $p$  by 1. This allows the polynomial filter to have a larger relative magnitude on the Ritz value that is most likely to converge next. (See §4.5.4.) Of course, this must be limited to avoid lowering the degree of the filter so much that it becomes ineffective. If  $k_o$  is the number of locked Ritz values, the IRLM iteration takes place in columns  $k_o + 1, \dots, m$  of  $V$ , working within an  $m - k_o$  length Lanczos factorization. The effective value of  $k$  becomes  $m - k_o - p_o$  and the effective value of  $p$  becomes  $p_o = p - k_o$ . This has two important effects. The rate of convergence as described in §4.5.4 is increased and the amount of work per implicit restart is decreased.

- (2) One might also wish to purge all converged but unwanted Ritz pairs at this stage.
- (3) The purpose of introducing the random start vector here is to greatly increase the likelihood of components in directions of wanted eigenvectors that have not yet been found.
- (4) This ad-hoc stopping strategy is reasonable if there is no opportunity to factor  $A$ . However, there is no ultimate assurance that the  $k$  wanted eigenvalues have all been found (especially in the case of clustered or multiple eigenvalues). If  $A$  (or better yet  $A - \sigma I$ ) can be factored into a symmetric indefinite factorization then convergence may be greatly enhanced through spectral transformation and one will have an inertia count available. That will give a more reliable verification that all  $k$  wanted eigenvalues have been found. See the discussion in §4.4.3.

#### 4.5.9 Software Availability

While we have presented the IRLM as much as possible in template form, we do not recommend implementation from this description for production purposes. High-quality software is freely available in ARPACK. This software is based upon the implicitly restarted Arnoldi method (IRAM) that is presented here in §7.6. When the matrix  $A$  is symmetric (Hermitian) this reduces to the IRLM described in this section.

ARPACK provides various drivers in template form for direct application to  $A$ ,  $SI$ , and for computing a partial SVD of a general rectangular matrix. For complete information, one should consult the *ARPACK Users' Guide* [295].

The thick restart variant of Wu and Simon [463] is implemented in the code TRLAN.

For more information about this software, including how to access it, see the book's homepage ETHOME.

### 4.6 Band Lanczos Method

*R. Freund*

The standard Hermitian Lanczos algorithm uses the Krylov subspaces induced by the matrix  $A$  and a single starting vector  $b$  to produce approximate solutions of the Hermitian eigenproblem  $Ax = \lambda x$ . However, there are situations where the use of a block of  $p$  starting vectors, instead of a single starting vector, is preferable. One such case is that of matrices with multiple or closely clustered eigenvalues. To obtain basis vectors for the eigenspace corresponding to such a cluster of  $p$  eigenvalues, block Krylov subspaces induced by  $A$  and a block of  $p$  starting vectors need to be used.

An important application, where multiple starting vectors are given as part of the problem, is reduced-order modeling of  $m$ -input  $p$ -output linear dynamical systems; see §7.10.4 below. While this application, in general, involves a non-Hermitian square matrix  $A$ , a rectangular “input” matrix  $B$  with  $m$  columns, and a rectangular “output” matrix  $C$  with  $p$  columns, the special case where  $A$  is Hermitian and the input and output matrices  $B$  and  $C$  are identical is of particular importance. For example, this special case arises in the context of interconnect modeling of VLSI circuits; see, e.g., [176]. For this special case, an extension of the Hermitian Lanczos algorithm to multiple starting vectors, namely, the  $p$  columns of  $B = C$ , is needed.

Finally, employing block Krylov subspaces is also beneficial whenever computing matrix-matrix products  $AV$ , where  $V$  is a matrix with  $p$  columns, is cheaper than sequentially computing matrix-vector products  $Av$  for  $p$  vectors. Lanczos methods based on block Krylov subspaces allow computation of all necessary multiplications with  $A$  as matrix-matrix products  $AV$  with blocks  $V$  of size  $p$ , whereas in the standard Hermitian Lanczos algorithm multiplications with  $A$  have to be computed as a sequence of matrix-vector products  $Av$ .

In this section, we describe a band Lanczos method for the Hermitian eigenvalue problem,

$$Ax = \lambda x, \quad (4.25)$$

that is based on block Krylov subspaces induced by the matrix  $A$  and a block of  $p$  starting vectors,

$$b_1, b_2, \dots, b_p. \quad (4.26)$$

The matrix  $A$  and the starting vectors (4.26) induce the block Krylov sequence

$$b_1, b_2, \dots, b_p, Ab_1, Ab_2, \dots, Ab_p, \dots, A^{k-1}b_1, A^{k-1}b_2, \dots, A^{k-1}b_p, \dots \quad (4.27)$$

The goal of the band Lanczos algorithm is to construct orthonormal vectors,

$$v_1, v_2, \dots, v_j, \quad (4.28)$$

that build a basis for the subspace spanned by the first  $j$  linearly independent vectors of the block Krylov sequence (4.27).

### 4.6.1 The Need for Deflation

The use of  $p > 1$  multiple starting vectors causes an additional difficulty that does not arise in the single-vector case  $p = 1$ . The standard Lanczos algorithm for a single starting vector  $b$  terminates after  $j$  iterations if the next Krylov vector,  $A^j b$ , is linearly dependent on the previous Krylov vectors, i.e.,  $A^j b \in \mathcal{K}^j(A, b)$ . In this case, the Lanczos vectors build a basis of the  $A$ -invariant subspace  $\mathcal{K}^j(A, b)$  and all eigenvalues of the Lanczos tridiagonal matrix are also eigenvalues of  $A$ . The subspace  $\mathcal{K}^j(A, b)$  being  $A$ -invariant means that the Krylov sequence has been fully exhausted, and so adding further Krylov vectors would not expand the Krylov subspace. This termination after  $j$  iterations is thus natural.

In the case  $p > 1$ , however, the occurrence of a first linearly dependent vector in (4.27) does not mean that the block Krylov sequence is exhausted. It simply means that the linearly dependent vector and all its following  $A$ -multiples do not contain any new information, and therefore, these vectors should be removed from (4.27). This process of detecting and deleting linearly dependent vectors is called *exact deflation*. For example, suppose the starting vectors (4.26) are such that the vector  $Ab_1$  is a linear combination of  $b_1, b_2, \dots, b_p$ . Then  $Ab_1$  is certainly linearly dependent on the Krylov vectors to the left of  $Ab_1$  in (4.27); in fact, each vector  $A^i b_1$  with  $i \geq 1$  is linearly dependent on vectors to the left of  $A^i b_1$  in (4.27). In this case, all vectors  $A^i b_1$ ,  $i \geq 1$ , need to be deleted from (4.27), resulting in the deflated block Krylov sequence

$$b_1, b_2, \dots, b_p, Ab_2, Ab_3, \dots, Ab_p, \dots, A^{k-1}b_2, A^{k-1}b_3, \dots, A^{k-1}b_p, \dots$$

For  $p > 1$ , in contrast to the case  $p = 1$ , the first occurrence of an exact deflation does not imply that any of the eigenvalues of the Lanczos matrix produced by the

band Lanczos method is also an eigenvalue of  $A$ . However, after  $p$  exact deflations have occurred, then, just as in the case  $p = 1$ , the Lanczos vectors span an  $A$ -invariant subspace and all the eigenvalues of the Lanczos matrix are also eigenvalues of  $A$ .

Of course, in finite precision arithmetic, it is impossible to distinguish between exactly linearly dependent and almost linearly dependent vectors. Therefore, in practice, almost linearly dependent vectors also have to be detected and deleted. In what follows, we will refer to the process of detecting and deleting linearly dependent and almost linearly dependent vectors as *deflation*.

#### 4.6.2 Basic Properties

After the first  $j$  iterations, the band Lanczos algorithm has generated the first  $j$  Lanczos vectors (4.28). These vectors are constructed to be orthonormal:

$$V_j^* V_j = I_j, \quad \text{where } V_j = [ v_1 \ v_2 \ \cdots \ v_j ]. \quad (4.29)$$

Here,  $I_j$  denotes the  $j \times j$  identity matrix. In addition to (4.28), the algorithm has constructed the vectors

$$\hat{v}_{j+1}, \hat{v}_{j+2}, \dots, \hat{v}_{j+p_c}, \quad (4.30)$$

which are the candidates for the next  $p_c$  Lanczos vectors,  $v_{j+1}, v_{j+2}, \dots, v_{j+p_c}$ . Here,  $p_c$  is the number of starting vectors,  $p$ , minus the number of deflations that have occurred during the first  $j$  iterations. The vectors (4.30) are constructed so that they satisfy the orthogonality relations

$$V_j^* \hat{v}_k = 0 \quad \text{for all } k = j + 1, j + 2, \dots, j + p_c. \quad (4.31)$$

The band Lanczos algorithm has a very simple built-in deflation procedure based on the vectors (4.30). In fact, an exact deflation at iteration  $j+1$  is equivalent to  $\hat{v}_{j+1} = 0$ . Therefore, in the algorithm, one checks if  $\|\hat{v}_{j+1}\|_2$  is smaller than some suitable deflation tolerance. If yes, the vector  $\hat{v}_{j+1}$  is deflated and  $p_c$  is reduced by 1. Otherwise,  $\hat{v}_{j+1}$  is normalized to become the next Lanczos vector  $v_{j+1}$ .

The recurrences that are used in the algorithm to generate the vectors (4.28) and (4.30) can be summarized compactly as follows:

$$AV_j = V_j T_j + [ 0 \ \cdots \ 0 \ \hat{v}_{j+1} \ \hat{v}_{j+2} \ \cdots \ \hat{v}_{j+p_c} ] + \hat{V}_j^{(dl)}. \quad (4.32)$$

Here,  $T_j$  is a  $j \times j$  matrix whose entries are chosen so that the orthogonality conditions (4.29) and (4.31) are satisfied. The matrix  $\hat{V}_j^{(dl)}$  in (4.32) contains mostly zero columns, together with the unnormalized vectors that have been deflated during the first  $j$  iterations. Recall that  $p - p_c$  is the number of deflated vectors.

It turns out that orthogonality only has to be explicitly enforced among  $2p_c + 1$  consecutive Lanczos vectors and, once deflation has occurred, against  $p - p_c$  fixed earlier vectors. As a result, the matrix  $T_j$  is “essentially” banded with bandwidth  $2p_c + 1$ , where the bandwidth is reduced by 2 every time a deflation occurs. In addition, each inexact deflation causes  $T_j$  to have nonzero elements in a fixed row outside and to the right of the banded part. More precisely, the row index of each such row caused by deflation is given by  $k - p_c(k)$ , where  $k$  is the number of the iteration at which the

deflation has occurred and  $p_c(k)$  is the corresponding value of  $p_c$  at that iteration. The matrix  $T_j$  can thus be written as

$$T_j = T_j^{(b)} + T_j^{(d)}, \quad (4.33)$$

where  $T_j^{(b)}$  is a banded matrix and  $T_j^{(d)}$  contains the horizontal “spikes” outside the band of  $T_j$  due to deflation. In particular, if no inexact deflation has occurred, then  $T_j^{(d)}$  is the zero matrix. Finally, we note that the banded part  $T_j^{(b)}$  of  $T_j$  is a Hermitian matrix.

For example, consider the case of  $p = 5$  starting vectors and assume that during the first  $j = 15$  iterations, deflations have occurred at steps  $k = 8$ ,  $k = 11$ , and  $k = 13$ . These three deflations correspond to deleting the vectors  $Ab_3$ ,  $A^2b_2$ , and  $A^3b_1$ , as well as the following  $A$ -multiples of these three vectors, from the block Krylov sequence (4.27). In this case, the matrix  $T_{15} = T_{15}^{(b)} + T_{15}^{(d)}$  has the following sparsity structure:

$$T_{15} = \left[ \begin{array}{ccccccccccccc} * & * & * & * & * & * & & & & & & & & & \\ * & * & * & * & * & * & * & & & & & & & & \\ * & * & * & * & * & * & * & d & d & d & d & d & d & d & d \\ * & * & * & * & * & * & * & * & * & * & * & * & * & * & \\ * & * & * & * & * & * & * & * & * & * & * & * & * & * & \\ * & * & * & * & * & * & * & * & * & * & * & * & * & * & \\ * & * & * & * & * & * & * & * & * & * & * & * & * & * & \\ * & * & * & * & * & * & * & * & * & * & * & * & d & d & d \\ * & * & * & * & * & * & * & * & * & * & * & * & * & * & \\ * & * & * & * & * & * & * & * & * & * & * & * & * & * & \\ * & * & * & * & * & * & * & * & * & * & * & * & * & * & \\ * & * & * & * & * & * & * & * & * & * & * & * & * & * & \\ * & * & * & * & * & * & * & * & * & * & * & * & * & * & \\ * & * & * & * & * & * & * & * & * & * & * & * & * & * & \\ * & * & * & * & * & * & * & * & * & * & * & * & * & * & \\ * & * & * & * & * & * & * & * & * & * & * & * & * & * & \\ \end{array} \right]. \quad (4.34)$$

Here, the \*'s denote potentially nonzero entries within the banded part,  $T_{15}^{(b)}$ , and the d's denote potentially nonzero entries of  $T_{15}^{(d)}$  due to the deflations at iterations  $k = 8$ ,  $k = 11$ , and  $k = 13$ . Note that the three deflations have reduced the initial bandwidth  $2p+1=11$  to  $2p_c+1=5$  at iteration  $j = 15$ .

After  $j$  iterations of the band Lanczos algorithm, approximate eigensolutions for the Hermitian eigenvalue problem (4.25) are obtained by computing eigensolutions of  $T_j^{(\text{pr})}$ ,

$$T_j^{(\text{pr})} z_i^{(j)} = \theta_i^{(j)} z_i^{(j)}, \quad i = 1, 2, \dots, j.$$

Here,  $T_j^{(\text{pr})}$  is the projection of  $A$  onto the space spanned by the Lanczos basis matrix  $V_j$ , i.e.,

$$T_j^{(\text{pr})} = V_j^* A V_j. \quad (4.35)$$

Each value  $\theta_i^{(j)}$  and its Ritz vector,  $x_i^{(j)} = V_j z_i^{(j)}$ , yield an approximate eigenpair of  $A$ . Of course, the matrix  $T_j^{(\text{pr})}$  is not computed via its definition (4.35). Instead, we use the formula

$$T_j^{(\text{pr})} = T_j + \left( T_j^{(d)} \right)^*. \quad (4.36)$$

By (4.36), we only have to conjugate and transpose the part of  $T_j$  outside its banded part and add it to  $T_j$  in order to obtain  $T_j^{(\text{pr})}$ . To show that (4.36) indeed holds true, note that by multiplying (4.32) from the left by  $V_j^*$  and by using the orthogonality relations (4.29) and (4.31), we get

$$T_j^{(\text{pr})} = V_j^* A V_j = T_j + S_j, \quad \text{where } S_j = V_j^* \widehat{V}_j^{(\text{dl})}. \quad (4.37)$$

Since the matrices  $T_j^{(\text{pr})}$  and  $T_j^{(\text{b})}$  are both Hermitian, it follows from (4.33) that  $S_j = (T_j^{(\text{d})})^*$  in (4.37). Thus (4.37) reduces to (4.36).

For example, for  $T_{15}$  in (4.34) the associated matrix  $T_{15}^{(\text{pr})} = T_{15} + (T_{15}^{(\text{d})})^*$  is of the form

$$T_{15}^{(\text{pr})} = \left[ \begin{array}{cccccccccccccccccc} * & * & * & * & * & * \\ * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & d & d & d & d & d & d & d & d & d \\ * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * & * & * \\ \bar{d} & * & * & * & * & * & * & * & * & * & * & * \\ \bar{d} & * & * & * & * & * & * & * & * & * & * \\ \bar{d} & * & * & * & * & * & * & * & * & * & * & * \\ \bar{d} & * & * & * & * & * & * & * & * & * & * & * \\ \bar{d} & * & * & * & * & * & * & * & * & * & * & * \\ \bar{d} & * & * & * & * & * & * & * & * & * & * & * \\ \bar{d} & \bar{d} & * & * & * & * & * & * & * & * & * & * \\ \bar{d} & \bar{d} & \bar{d} & * & * & * & * & * & * & * & * & * \\ \bar{d} & \bar{d} & \bar{d} & \bar{d} & * & * & * & * & * & * & * & * \end{array} \right]. \quad (4.38)$$

Here, the  $\bar{d}$ 's below the banded part were obtained by conjugating and transposing the corresponding entries above the banded part in (4.34). We remark that, in (4.38), the entries  $d$  and  $\bar{d}$  outside the banded part are usually small. More precisely, if the deflation criterion (4.42) below is used, then the absolute values of all  $d$ 's and  $\bar{d}$ 's is bounded by the deflation tolerance  $\text{dtol}$ . Even though these entries are small, setting them to zero would introduce unnecessary errors. Indeed, the projection property (4.35) for  $T_j^{(\text{pr})}$  holds true only if all entries  $d$  and  $\bar{d}$  outside the banded part are included in  $T_j^{(\text{pr})}$ .

Finally, we note that the band Lanczos algorithm terminates as soon as  $p_c = 0$  is reached. This means that  $p$  deflations have occurred, and thus the block Krylov sequence is exhausted. At termination due to  $p_c = 0$ , the relation (4.32) for the Lanczos vectors reduces to

$$AV_j = V_j T_j + \widehat{V}_j^{(\text{dl})}. \quad (4.39)$$

Using (4.37), we can rewrite (4.39) as follows:

$$AV_j - V_j T_j^{(\text{pr})} = (I - V_j V_j^*) \widehat{V}_j^{(\text{dl})}. \quad (4.40)$$

Now let  $\theta_i^{(j)}$  and  $z_i^{(j)}$  be any of the eigenpairs of  $T_j^{(\text{pr})}$ , and assume that  $z_i^{(j)}$  is normalized so that  $\|z_i^{(j)}\|_2 = 1$ . Recall that the pair  $\theta_i^{(j)}$  and  $x_i^{(j)} = V_j z_i^{(j)}$  is used as an approximate eigensolution of  $A$ . By multiplying (4.40) from the right by  $z_i^{(j)}$  and taking norms, it

follows that the residual of the approximate eigensolution  $\theta_i^{(j)}$  and  $x_i^{(j)}$  can be bounded as follows:

$$\left\| Ax_i^{(j)} - \theta_i^{(j)} x_i^{(j)} \right\|_2 = \left\| (I - V_j V_j^*) \widehat{V}_j^{(\text{dl})} z_i^{(j)} \right\|_2 \leq \left\| \widehat{V}_j^{(\text{dl})} \right\|_2. \quad (4.41)$$

In particular, if only exact deflation is performed, then  $\widehat{V}_j^{(\text{dl})} = 0$  and (4.41) shows that each eigenvalue  $\theta_i^{(j)}$  of  $T_j^{(\text{pr})}$  is indeed an eigenvalue of  $A$ . For general deflation,  $\widehat{V}_j^{(\text{dl})} \neq 0$ , however,  $\left\| \widehat{V}_j^{(\text{dl})} \right\|_2$  is of the order of the deflation tolerance. More precisely, if the deflation check (4.42) below is used, then

$$\left\| \widehat{V}_j^{(\text{dl})} \right\|_2 \leq \sqrt{p} \text{ dtol},$$

where  $\text{dtol}$  denotes the deflation tolerance.

### 4.6.3 Algorithm

A complete statement of the band Lanczos algorithm for Hermitian matrices  $A$  is as follows.

**ALGORITHM 4.12: Band Lanczos Method for HEP**

- (1) set  $\widehat{v}_k = b_k$  for  $k = 1, 2, \dots, p$
- (2) set  $p_c = p$  and  $\mathcal{I} = \emptyset$
- (3) **for**  $j = 1, 2, \dots$ , until convergence or  $p_c = 0$  **do**
- compute  $\|\widehat{v}_j\|_2$
- decide if  $\widehat{v}_j$  should be deflated. If yes, do the following:
- (a) if  $j - p_c > 0$ , set  $\mathcal{I} = \mathcal{I} \cup \{j - p_c\}$
- (b) set  $p_c = p_c - 1$ . If  $p_c = 0$ , set  $j = j - 1$  and **stop**
- (c) **for**  $k = j, j + 1, \dots, j + p_c - 1$ , set  $\widehat{v}_k = \widehat{v}_{k+1}$
- (d) return to step (3)
- set  $t_{j,j-p_c} = \|\widehat{v}_j\|_2$  and normalize  $v_j = \widehat{v}_j / t_{j,j-p_c}$
- (6) **for**  $k = j + 1, j + 2, \dots, j + p_c - 1$ , set
- $t_{j,k-p_c} = v_j^* \widehat{v}_k$  and  $\widehat{v}_k = \widehat{v}_k - v_j t_{j,k-p_c}$
- compute  $\widehat{v}_{j+p_c} = Av_j$ .
- set  $k_0 = \max\{1, j - p_c\}$ . For  $k = k_0, k_0 + 1, \dots, j - 1$ , set
- $t_{kj} = \overline{t_{jk}}$  and  $\widehat{v}_{j+p_c} = \widehat{v}_{j+p_c} - v_k t_{kj}$
- (9) **for**  $k \in \mathcal{I} \cup \{j\}$  (in ascending order), set
- $t_{kj} = v_k^* \widehat{v}_{j+p_c}$  and  $\widehat{v}_{j+p_c} = \widehat{v}_{j+p_c} - v_k t_{kj}$
- (10) **for**  $k \in \mathcal{I}$ , set  $s_{jk} = \overline{t_{kj}}$ .
- (11) set  $T_j^{(\text{pr})} = T_j + S_j = [t_{ik}]_{i,k=1,2,\dots,j} + [s_{ik}]_{i,k=1,2,\dots,j}$
- (12) test for convergence
- (13) **end for**

Next, we discuss some of the steps of Algorithm 4.12 in more detail.

- (4) Generally, the decision on whether  $\widehat{v}_j$  needs to be deflated should be based on checking whether

$$\|\widehat{v}_j\|_2 \leq \text{dtol}, \quad (4.42)$$

where  $\text{dtol}$  is a suitably chosen deflation tolerance. The vector  $\hat{v}_j$  is then deflated if (4.42) is satisfied. In this case, the value  $j - p_c$ , if positive, is added to the index set  $\mathcal{I}$ , which contains the indices of the nonzero rows in the part  $T_j^{(d)}$  of  $T_j$ , and the current block size is updated to  $p_c = p_c - 1$ . If  $p_c = 0$ , the block Krylov sequence (4.27) is exhausted, and the algorithm terminates naturally. If  $p_c > 0$ , the vector  $\hat{v}_j$  is deleted, the index  $k+1$  of each of the remaining candidate vectors  $\hat{v}_{k+1}$  is reset to  $k$ , and finally, the algorithm returns to step (3). If (4.42) is not satisfied, no deflation is necessary and the algorithm proceeds with step (5).

Usually, in (4.42), one will choose a small tolerance  $\text{dtol}$ , such as the square root of machine precision. However,  $\text{dtol}$  does not have to be small; Algorithm 4.12 and its properties remain correct for *any* choice of  $\text{dtol}$ . Note that the algorithm performs only exact deflation if one sets  $\text{dtol} = 0$  in (4.42).

- (5) The vector  $\hat{v}_j$  has passed the deflation check and is now normalized to become the next Lanczos vector  $v_j$ . The normalization is such that

$$\|v_j\|_2 = 1 \quad \text{for all } j.$$

- (6) The remaining candidate vectors,  $\hat{v}_{j+1}, \hat{v}_{j+2}, \dots, \hat{v}_{j+p_c-1}$ , are explicitly orthogonalized against the latest Lanczos vector  $v_j$ . Note that in the first  $p$  steps some  $t_{j,k-p_c}$  will have nonpositive column index. They serve to make the starting basis orthonormal, but need not be stored in the  $T_j$  matrix.
- (7) The block Krylov sequence is advanced by computing a new candidate vector,  $\hat{v}_{j+p_c}$ , as the  $A$ -multiple of the latest Lanczos vector  $v_j$ .
- (8) The vector  $\hat{v}_{j+p_c}$  is orthogonalized against the previous Lanczos vectors  $v_{k_0}, v_{k_0+1}, \dots, v_{j-1}$ , where  $k_0 = \max\{1, j - p_c\}$ . Here, we exploit the fact that the matrix  $T_j^{(\text{pr})}$  is Hermitian, and instead of explicitly computing  $t_{kj}$ , we set  $t_{kj} = \overline{t_{jk}}$ . Note that the  $t_{jk}$ 's were computed explicitly in step (6).
- (9) The vector  $\hat{v}_{j+p_c}$  is explicitly orthogonalized against the Lanczos vectors  $v_k$ ,  $k \in \mathcal{I}$ , and against  $v_j$ . Note that modified Gram–Schmidt is used to do this orthogonalization.
- (11) All the nonzero elements in the  $j$ th row and the  $j$ th column have now been computed, and they are added to the matrix  $T_{j-1}^{(\text{pr})}$  of the previous iteration  $j-1$ , to yield the current matrix  $T_j^{(\text{pr})}$ . Here, we use the convention that entries  $t_{ik}$  and  $s_{ik}$  that are not explicitly defined in Algorithm 4.12 are set to zero.
- (12) To test for convergence, the eigenvalues  $\theta_i^{(j)}$ ,  $i = 1, 2, \dots, j$ , of the Hermitian  $j \times j$  matrix  $T_j^{(\text{pr})}$  are computed, and the algorithm is stopped if some of the  $\theta_i^{(j)}$ 's are good enough approximations to the desired eigenvalues of  $A$ .

#### 4.6.4 Variants

A band Lanczos method for Hermitian matrices was first proposed in [375]. The Hermitian band Lanczos Algorithm 4.12 described here, however, is somewhat different in that it uses both the first  $j$  Lanczos vectors (4.28) and the candidate vectors (4.30)

for the following  $p_c$  Lanczos vectors. This makes it possible to build only the  $j \times j$  matrix  $T_j^{(\text{pr})}$ . The algorithm in [375] constructs the first  $j + p_c$  Lanczos vectors, and as a result, it needs to build a somewhat larger  $(j + p_c) \times j$  matrix, instead of  $T_j^{(\text{pr})}$ . The version of the band Lanczos method described here was first derived in [177] as a special case of the general nonsymmetric band Lanczos method proposed in [166].

The version of the band Lanczos method stated as Algorithm 4.12 performs all multiplications with  $A$  as matrix-vector products with single vectors. However, at any stage of the algorithm, it is also possible to precompute the next  $p_c$  matrix-vector products, which will be needed in the next  $p_c$  iterations, as a single matrix-matrix product  $AV$  with a block  $V$  of  $p_c$  vectors. The procedure is as follows. Instead of performing step (7) in Algorithm 4.12, we compute the matrix-matrix product

$$A \begin{bmatrix} v_j & \hat{v}_{j+1} & \hat{v}_{j+2} & \cdots & \hat{v}_{j+p_c-1} \end{bmatrix}.$$

This gives us the vector  $\hat{v}_{j+p_c} = Av_j$ , which is used in the remaining steps of iteration  $j$ , as well as the vectors

$$A\hat{v}_{j+1}, A\hat{v}_{j+2}, \dots, A\hat{v}_{j+p_c-1}. \quad (4.43)$$

In the following  $p_c - 1$  iterations, we will need the vectors

$$Av_{j+1}, Av_{j+2}, \dots, Av_{j+p'_c-1}. \quad (4.44)$$

Here,  $p'_c$  is defined as  $p_c$  minus the number of deflations that will have occurred during these following  $p_c - 1$  iterations. In particular,  $p'_c = p_c$  if no deflations will occur. The matrix-vector products (4.44) we need are not quite the ones we computed in (4.43). However, the vectors (4.43) and (4.44) are connected by the relation

$$\begin{aligned} & \begin{bmatrix} A\hat{v}_{j+1} & A\hat{v}_{j+2} & \cdots & A\hat{v}_{j+p_c-1} \end{bmatrix} \\ &= \begin{bmatrix} Av_{j+1} & Av_{j+2} & \cdots & Av_{j+p'_c-1} \end{bmatrix} T_{j+1:j+p'_c-1, j-p_c+1:j-1}. \end{aligned} \quad (4.45)$$

Here,  $T_{j+1:j+p'_c-1, j-p_c+1:j-1}$  is the submatrix of  $T_{j+p'_c-1}$  that consists of the entries in rows  $j+1, \dots, j+p'_c-1$  and columns  $j-p_c+1, \dots, j-1$  of  $T_{j+p'_c-1}$ . If  $p'_c < p_c$ , some of the  $\hat{v}_k$ 's in (4.45) will lead to deflated vectors. Let  $\hat{v}_{j_1}, \hat{v}_{j_2}, \dots, \hat{v}_{j_{p'_c-1}}$  be the vectors that are not deflated. Furthermore, let  $\tilde{T}$  be the matrix obtained from  $T_{j+1:j+p'_c-1, j-p_c+1:j-1}$  by deleting the columns that correspond to the deflated  $\hat{v}_k$  vectors in (4.45). With these notations, by (4.45), we have

$$\begin{bmatrix} A\hat{v}_{j_1} & A\hat{v}_{j_2} & \cdots & A\hat{v}_{j_{p'_c-1}} \end{bmatrix} = \begin{bmatrix} Av_{j+1} & Av_{j+2} & \cdots & Av_{j+p'_c-1} \end{bmatrix} \tilde{T}. \quad (4.46)$$

The matrix  $\tilde{T}$  is nonsingular and upper triangular. Therefore, using (4.46), we can obtain each of the vectors  $Av_{j+k}$ ,  $k = 1, 2, \dots, p'_c - 1$ , in (4.44) as a suitable linear combination of  $k$  of the precomputed vectors (4.43).

For Hermitian positive definite matrices  $A$ , yet another variant of Algorithm 4.12, based on coupled recurrences, was proposed in [35]. The motivation for this variant was the observation that for ill-conditioned Hermitian positive definite matrices  $A$ , the smallest eigenvalue obtained from the Lanczos matrix  $T_j^{(\text{pr})}$  may become negative; see [34, 35] for examples. In exact arithmetic, this would be impossible, since the positive definiteness of  $A$  implies that  $T_j^{(\text{pr})} = V_j^* AV_j$  is also positive definite. However, in finite precision arithmetic, roundoff may cause the matrix  $T_j^{(\text{pr})}$  generated by

Algorithm 4.12 to be slightly indefinite. This can be avoided easily by using coupled recurrences to generate a Cholesky factor of  $T_j^{(\text{pr})}$ , instead of  $T_j^{(\text{pr})}$  itself. More precisely, the recurrences summarized in (4.32) are replaced by coupled recurrences of the form

$$\begin{aligned} AP_j &= V_j L_j D_j + [ \begin{array}{cccccc} 0 & \cdots & 0 & \widehat{v}_{j+1} & \widehat{v}_{j+2} & \cdots & \widehat{v}_{j+p_c} \end{array}] + \widehat{V}_j^{(\text{dl})}, \\ V_j &= P_j U_j. \end{aligned} \quad (4.47)$$

Here,  $P_j$  is a second basis matrix whose columns span the same subspace as  $V_j$ , and it is constructed to be  $A$ -orthogonal:

$$P_j^* A P_j = D_j,$$

where  $D_j$  is a positive definite diagonal matrix. Furthermore, in (4.47), the matrix  $L_j$  is unit lower triangular, and the matrix  $U_j$  is unit upper triangular. After  $j$  iterations, the approximate eigenpairs of  $A$  are then obtained by computing the eigensolutions of the Hermitian positive definite matrix

$$V_j^* A V_j = U_j^* D_j U_j,$$

which is given in terms of  $D_j$  and  $U_j$ .

Finally, we note that the block Lanczos method is the more traditional way of extending the standard Lanczos algorithm for a single starting vector to multiple starting vectors. When the block Lanczos method is combined with deflation (as described, e.g., in [89]), then the resulting algorithm is mathematically equivalent to the band Lanczos method described here. However, the band Lanczos method has two distinct advantages over the block Lanczos method. First, deflation is extremely simple, since it requires only to check the norm of a single vector. In contrast, in the block Lanczos method, one needs to check if a whole block of  $p_c$  vectors has full rank, and if not, find the linearly dependent columns. Typically, this requires the computation of a QR factorization of each block. Second, the band Lanczos method actually performs slightly fewer explicit orthogonalizations, resulting in a Lanczos matrix  $T_j^{(\text{pr})}$  that is slightly sparser than the one produced by the block Lanczos method.

## 4.7 Jacobi–Davidson Methods

*G. Sleijpen and H. van der Vorst*

The Lanczos method (see §4.4) is quite effective in computing eigenvalues in the ends of the spectrum of  $A$  if these eigenvalues are well separated from the remaining spectrum, or if it is applied to a shifted and inverted matrix operator  $(A - \sigma I)^{-1}$ , for some reasonable shift  $\sigma$  close to the interesting eigenvalues.

If none of these conditions is fulfilled, for instance, if the computation of a vector  $(A - \sigma I)^{-1}y$  for given  $y$  is not feasible with a direct solver, then variants of the Jacobi–Davidson method [411] offer an attractive alternative.

### 4.7.1 Basic Theory

The Jacobi–Davidson method is based on a combination of two basic principles. The first one is to apply a Galerkin approach for the eigenproblem  $Ax = \lambda x$ , with respect to

some given subspace spanned by an orthonormal basis  $v_1, \dots, v_m$ . The usage of other than Krylov subspaces was suggested by Davidson [99], who also suggested specific choices, different from the ones that we will take, for the construction of orthonormal basis vectors  $v_j$ . The Galerkin condition is

$$AV_m s - \theta V_m s \perp \{v_1, \dots, v_m\},$$

which leads to the reduced system

$$V_m^* AV_m s - \theta s = 0, \quad (4.48)$$

where  $V_m$  denotes the matrix with columns  $v_1$  to  $v_m$ . This equation has  $m$  solutions  $(\theta_j^{(m)}, s_j^{(m)})$ . The  $m$  pairs  $(\theta_j^{(m)}, u_j^{(m)} \equiv V_m s_j^{(m)})$  are called the Ritz values and Ritz vectors of  $A$  with respect to the subspace spanned by the columns of  $V_m$ . These Ritz pairs are approximations for eigenpairs of  $A$ , and our goal is to get better approximations by a well-chosen expansion of the subspace.

At this point the other principle behind the Jacobi–Davidson approach comes into play. The idea goes back to Jacobi [241]. Suppose that we have an eigenvector approximation  $u_j^{(m)}$  for an eigenvector  $x$  corresponding to a given eigenvalue  $\lambda$ . Then Jacobi suggested (in the original paper for strongly diagonally dominant matrices) computing the orthogonal correction  $t$  for  $u_j^{(m)}$  so that

$$A(u_j^{(m)} + t) = \lambda(u_j^{(m)} + t).$$

Since  $t \perp u_j^{(m)}$ , we can restrict ourselves to the subspace orthogonal to  $u_j^{(m)}$ . The operator  $A$  restricted to that subspace is given by

$$(I - u_j^{(m)} u_j^{(m)*}) A (I - u_j^{(m)} u_j^{(m)*}),$$

and we find that  $t$  satisfies the equation

$$(I - u_j^{(m)} u_j^{(m)*}) (A - \lambda I) (I - u_j^{(m)} u_j^{(m)*}) t = -(A - \theta_j^{(m)} I) u_j^{(m)}.$$

In practical situations we do not know  $\lambda$ , and the obvious solution to this is to replace it by its approximation  $\theta_j^{(m)}$ , which leads to the *Jacobi–Davidson correction equation* for an update  $t_j^{(m)} \perp u_j^{(m)}$ :

$$\begin{aligned} & (I - u_j^{(m)} u_j^{(m)*}) (A - \theta_j^{(m)} I) (I - u_j^{(m)} u_j^{(m)*}) t_j^{(m)} \\ &= -r_j^{(m)} \equiv -(A - \theta_j^{(m)} I) u_j^{(m)}. \end{aligned} \quad (4.49)$$

This correction equation is solved only approximately and its approximate solution  $\tilde{t}^{(m)}$  is taken for the expansion of the subspace. This is a fundamental difference with the Krylov subspace methods; instead of selecting a subspace as powers of an operator acting on a given starting vector, we select some subspace without Krylov structure and we project the given matrix onto this subspace.

From (4.48) we conclude that  $r_j^{(m)} \perp \{v_1, \dots, v_m\}$ , and in particular that  $r_j^{(m)} \perp u_j^{(m)}$ , so that the Jacobi–Davidson correction equation represents a consistent linear system.

It can be shown that the exact solution of (4.49) leads to cubic convergence of the largest  $\theta_j^{(m)}$  towards  $\lambda_{\max}(A)$  for increasing  $m$  (similar statements can be made for the convergence towards other eigenvalues of  $A$ , provided that the Ritz values are selected appropriately in each step).

In [411] it is suggested to solve equation (4.49) only approximately, for instance, by some steps of minimal residual (MINRES) [350], with an appropriate preconditioner  $K$  for  $A - \theta_j^{(m)}I$ , if available, but in fact any approximation technique for  $t_j^{(m)}$  is formally allowed, provided that the projectors  $(I - u_j^{(m)}u_j^{(m)*})$  are taken into account. In our templates we will present ways to approximate  $t_j^{(m)}$  with Krylov subspace methods.

We will now discuss how to use preconditioning for an iterative solver like generalized minimal residual (GMRES) or conjugate gradient squared (CGS), applied to equation (4.49). Suppose that we have a left preconditioner  $K$  available for the operator  $A - \theta_j^{(m)}I$ , for which in some sense  $K^{-1}(A - \theta_j^{(m)}I) \approx I$ . Since  $\theta_j^{(m)}$  varies with the iteration count  $m$ , so may the preconditioner, although it is often practical to work with the same  $K$  for different values of  $\theta$ . Of course, the preconditioner  $K$  has to be restricted to the subspace orthogonal to  $u_j^{(m)}$  as well, which means that we have to work with, efficiently,

$$\tilde{K} \equiv (I - u_j^{(m)}u_j^{(m)*}) K (I - u_j^{(m)}u_j^{(m)*}).$$

This may seem unnecessarily complicated, and at first sight it also seems to involve a lot of overhead because of the projections  $(I - u_j^{(m)}u_j^{(m)*})$  surrounding the matrix-vector operations. But it all amounts to a handful of rather simple operations, as we will show now.

We assume that we use a Krylov solver with initial guess  $t_0 = 0$  and with left preconditioning for the approximate solution of the correction equation (4.49). Since the starting vector is in the subspace orthogonal to  $u_j^{(m)}$ , all iteration vectors for the Krylov solver will be in that space. In that subspace we have to compute the vector  $z \equiv \tilde{K}^{-1}\tilde{A}v$  for a vector  $v$  supplied by the Krylov solver, and

$$\tilde{A} \equiv (I - u_j^{(m)}u_j^{(m)*})(A - \theta_j^{(m)}I)(I - u_j^{(m)}u_j^{(m)*}).$$

We will do this in two steps. First we compute

$$\tilde{A}v = (I - u_j^{(m)}u_j^{(m)*})(A - \theta_j^{(m)}I)(I - u_j^{(m)}u_j^{(m)*})v = (I - u_j^{(m)}u_j^{(m)*})y$$

with  $y \equiv (A - \theta_j^{(m)}I)v$  since  $u_j^{(m)*}v = 0$ . Then, with left-preconditioning, we solve  $z \perp u_j^{(m)}$  from

$$\tilde{K}z = (I - u_j^{(m)}u_j^{(m)*})y.$$

Since  $z \perp u_j^{(m)}$ , it follows that  $z$  satisfies  $Kz = y - \alpha u_j^{(m)}$  or  $z = K^{-1}y - \alpha K^{-1}u_j^{(m)}$ . The condition  $z \perp u_j^{(m)}$  leads to

$$\alpha = \frac{u_j^{(m)*}K^{-1}y}{u_j^{(m)*}K^{-1}u_j^{(m)}}.$$

The vector  $\widehat{y} \equiv K^{-1}y$  is solved from  $K\widehat{y} = y$  and, likewise,  $\widehat{u} \equiv K^{-1}u_j^{(m)}$  is solved from  $K\widehat{u} = u_j^{(m)}$ . The last equation has to be solved only once in an iteration process for equation (4.49), so that effectively  $i_S + 1$  operations with the preconditioner are required for  $i_S$  iterations of the linear solver. Also, a matrix-vector multiplication with the left-preconditioned operator in an iteration of the Krylov solver requires only one inner product and one vector update instead of the four actions of the projector operator  $(I - u_j^{(m)}u_j^{(m)*})$ . This has been worked out in the solution template, given in Algorithm 4.15. Along similar lines one can obtain an efficient template, although slightly more expensive than the left-preconditioned case, for the right-preconditioned operator. This template is described in Algorithm 4.16. For more details on preconditioning of the correction equation, see [412].

If we form an approximation for  $t_j^{(m)}$  in (4.49) as  $\tilde{t}^{(m)} \equiv K^{-1}r - \alpha K^{-1}u_j^{(m)}$ , with  $\alpha$  such that  $\tilde{t}^{(m)} \perp u_j^{(m)}$  and without acceleration by an iterative solver, we obtain a process which was suggested by Olsen, Jørgensen, and Simons [344].

### 4.7.2 Basic Algorithm

The basic form of the Jacobi–Davidson algorithm is given in Algorithm 4.13. Later, we will describe more sophisticated variants with restart and other strategies.

In each iteration of this algorithm an approximated eigenpair  $(\theta, u)$  for the eigenpair of the Hermitian matrix  $A$ , corresponding to the largest eigenvalue of  $A$ , is computed. The iteration process is terminated as soon as the norm of the residual  $Au - \theta u$  is below a given threshold  $\epsilon$ .

**ALGORITHM 4.13: Jacobi–Davidson Method for  $\lambda_{\max}(A)$  for HEP**

- (1) start with  $t = v_0$ , starting guess
- (2) **for**  $m = 1, 2, \dots$
- (3)     **for**  $i = 1, \dots, m - 1$
- (4)          $t = t - (v_i^* t)v_i$
- (5)     **end for**
- (6)      $v_m = t/\|t\|_2$ ,  $v_m^A = Av_m$
- (7)     **for**  $i = 1, \dots, m$
- (8)          $M_{i,m} = v_i^* v_m^A$
- (9)     **end for**
- (10)    compute the largest eigenpair  $(\theta, s)$  of  $M$  ( $\|s\|_2 = 1$ )
- (11)     $u = Vs$
- (12)     $u^A = V^A s$
- (13)     $r = u^A - \theta u$
- (14)    **if** ( $\|r\|_2 \leq \epsilon$ ),  $\tilde{\lambda} = \theta$ ,  $\tilde{x} = u$ , **then stop**
- (15)    solve (approximately) a  $t \perp u$  from  

$$(I - uu^*)(A - \theta I)(I - uu^*)t = -r$$
- (16) **end for**

To apply this algorithm we need to specify a starting vector  $v_0$  and a tolerance  $\epsilon$ . On completion an approximation for the largest eigenvalue  $\lambda = \lambda_{\max}(A)$  and its corresponding eigenvector  $x = x_{\max}$  is delivered. The computed eigenpair  $(\tilde{\lambda}, \tilde{x})$  satisfies  $\|A\tilde{x} - \tilde{\lambda}\tilde{x}\| < \epsilon$ .

We will now describe some implementation details, referring to the respective phases in Algorithm 4.13.

- (1) This is the initialization phase of the process. The search subspace is expanded in each iteration by a vector  $t$ , and we start this process with a given vector  $t = v_0$ . Ideally, this vector should have a significant component in the direction of the wanted eigenvector. Unless one has some idea of the wanted eigenvector, it may be a good idea to start with a random vector. This gives some confidence that the wanted eigenvector has a nonzero component in the starting vector, which is necessary for detection of the eigenvector.
  - (3)–(5) This represents the modified Gram–Schmidt process for the orthogonalization of the new vector  $t$  with respect to the set  $v_1, \dots, v_{m-1}$ . If  $m = 1$ , this is an empty loop. Let  $t_{in}$  represent the vector before the start of the orthogonalization, and  $t_{out}$  the vector that results at completion of phase (3)–(5). It is advisable (see [96]) to repeat the Gram–Schmidt process one time if  $\|t_{out}\|_2/\|t_{in}\|_2 \leq \kappa$ , where  $\kappa$  is a modest constant, say  $\kappa = .25$ . This guarantees that the loss of orthogonality is restricted to  $1/\kappa$  times machine precision, in a relative sense. The template for this modified Gram–Schmidt orthogonalization with iterative refinement is given in Algorithm 4.14.
  - (7)–(9) In this phase, computation of the  $m$ th column of the upper triangular part of the matrix  $M \equiv V^*AV$  occurs. The matrix  $V$  denotes the  $n$  by  $m$  matrix with columns  $v_j$ ,  $V^A$  likewise.
  - (10) Computing the largest eigenpair of the  $m \times m$  Hermitian matrix  $M$ , with elements  $M_{i,j}$  in its upper triangular part, can be done with the appropriate routines from LAPACK (see §4.2).
  - (12) The vector  $u^A$  may either be updated as described here or recomputed as  $u^A = Au$ , depending on which is cheapest. The choice is between an  $m$ -fold update and another multiplication with  $A$ ; if  $A$  has fewer than  $m$  nonzero elements on average per row, the computation via  $Au$  is preferable. If  $u^A$  is computed as  $Au$ , it is not necessary to store the vectors  $v_j^A$ .
  - (14) The algorithm is terminated if  $\|Au - \theta u\|_2 \leq \epsilon$ . In that case  $A$  has an eigenvalue  $\lambda$  for which  $|\lambda - \theta| \leq \epsilon$ . For the corresponding normalized eigenvector there is a similar bound on the angle, provided that  $\lambda$  is simple and well separated from the other eigenvalues of  $A$ . That case leads also to a sharper bound for  $\lambda$ ; see (4.5) or §4.8.
- Convergence to a  $\lambda \neq \lambda_{\max}(A)$  may take place, but is in general unlikely. It happens, for instance, if  $v_0 \perp x_{\max}$ , or if the selected  $\theta$  is very close to an eigenvalue  $\lambda \neq \lambda_{\max}(A)$ . This may happen for any iterative solver, in particular if  $\epsilon$  is taken not small enough (say, larger than the square root of machine precision).
- (15) The approximate solution for the expansion vector  $t$  can be computed with a Krylov solver, for instance, MINRES, or with SYMMLQ. With left- or right-preconditioning one has to select a Krylov solver for unsymmetric systems (like GMRES, CGS, or Bi-CGSTAB), since the preconditioned operator is in general

not symmetric. A template for the approximate solution, with a left-preconditioned Krylov subspace method of choice, is given in Algorithm 4.15. The right preconditioned case, which is slightly more expensive, is covered by the template in Algorithm 4.16. For iterative Krylov subspace solvers see [41]. The approximate solution has to be orthogonal to  $u$ , but that is automatically the case with Krylov solvers if one starts with an initial guess orthogonal to  $u$ , for instance,  $t_0 = 0$ . In most cases it is not necessary to solve the correction equation to high precision; a relative precision of  $2^{-m}$  in the  $m$ th iteration seems to suffice. It is advisable to put a limit to the number of iteration steps for the iterative solver.

Davidson [99] suggested taking  $t = (\text{diag}(A) - \theta I)^{-1}r$ , but in this case  $t$  is not orthogonal with respect to  $u$ . Moreover, for diagonal matrices this choice leads to stagnation, which is an illustration of the problems in this approach.

In order to restrict storage, the algorithm can be terminated at some appropriate value  $m = m_{\max}$  and restarted with  $v_0$  as the latest value of  $u$ . We will describe a variant of the Jacobi–Davidson algorithm with a more sophisticated restart strategy in §4.7.3.

Note that most of the computationally intensive operations, i.e., those operations the cost of which is proportional to  $n$ , can easily be parallelized. Also, the multiple vector updates can be performed by the appropriate Level 2 BLAS routines (see §10.2).

In the coming subsections we will describe more sophisticated variants of the Jacobi–Davidson algorithm. In §4.7.3 we will introduce a variant that allows for restarts, which is convenient if one wants to keep the dimensions of the involved subspaces limited. This variant is also suitable for a restart after an eigenpair has been discovered, in order to locate the next eigenpair. The technique is based on deflation. The resulting algorithm is designed for the computation of a few of the largest or smallest eigenvalues of a given matrix.

In §4.7.4 we will describe a variant of the Jacobi–Davidson method that is suitable for the computation of interior eigenvalues of  $A$ .

**ALGORITHM 4.14: Modified Gram–Schmidt Orthogonalization with Refinement**

- (1) select a value for  $\kappa$  less than 1, say  $\kappa = .25$
- (2)  $\tau_{in} = \|t\|_2$
- (3) **for**  $i = 1, \dots, m - 1$
- (4)      $t = t - (v_i^* t)v_i$
- (5) **end for**
- (6) **if**  $\|t\|_2/\tau_{in} \leq \kappa$
- (7)     **for**  $i = 1, \dots, m - 1$
- (8)          $t = t - (v_i^* t)v_i$
- (9)     **end for**
- (10) **end if**

**ALGORITHM 4.15: Approximate Solution of the Jacobi–Davidson HEP Correction Equation with Left-Preconditioning**

“Solve” with left preconditioner  $\tilde{K} \equiv (I - uu^*)K(I - uu^*)$ ,  
for  $\tilde{A} \equiv (I - uu^*)(A - \theta I)(I - uu^*)$ :

- (1) solve  $\hat{u}$  from  $K\hat{u} = u$ ,  $\mu \equiv u^*\hat{u}$
- (2) compute  $\tilde{r} \equiv \tilde{K}^{-1}r$  as:
- (3) (b') solve  $\hat{r}$  from  $K\hat{r} = r$
- (4) (c')  $\tilde{r} = \hat{r} - \frac{u^*\hat{r}}{\mu} \hat{u}$
- (5) apply a Krylov subspace method with  $t_0 = 0$ , operator  $\tilde{K}^{-1}\tilde{A}$ ,  
and right-hand side  $-\tilde{r}$ ; given  $v$ ,  $z = \tilde{K}^{-1}\tilde{A}v$  is computed as:
- (6) (a)  $y = (A - \theta I)v$
- (7) (b) solve  $\hat{y}$  from  $K\hat{y} = y$
- (8) (c)  $z = \hat{y} - \frac{u^*\hat{y}}{\mu} \hat{u}$

**ALGORITHM 4.16: Approximate Solution of the Jacobi–Davidson HEP Correction Equation with Right-Preconditioning**

“Solve” with right preconditioner  $\tilde{K} \equiv (I - uu^*)K(I - uu^*)$ ,  
for  $\tilde{A} \equiv (I - uu^*)(A - \theta I)(I - uu^*)$ :

- (1) solve  $\hat{u}$  from  $K\hat{u} = u$ ,  $\mu \equiv u^*\hat{u}$
- (2) apply Krylov subspace method with start  $t_0 = 0$ , with operator  $\tilde{A}\tilde{K}^{-1}$ ,  
and right-hand side  $-r$ ,  $z = \tilde{A}\tilde{K}^{-1}v$  for given  $v$  is computed as:
- (3) (a) solve  $\hat{v}$  from  $K\hat{v} = v$
- (4) (b)  $y = \hat{v} - \frac{u^*\hat{v}}{\mu} \hat{u}$
- (5) (c)  $z = (I - uu^*)(A - \theta I)y$
- (6) the approximate solution  $t$  should be back-transformed as:
- (7) (a'') solve  $\hat{t}$  from  $K\hat{t} = t$
- (8) (b'')  $t = \hat{t} - \frac{u^*\hat{t}}{\mu} \hat{u}$

**Storage and Computational Costs.** We have collected the dominant costs of the simple Jacobi–Davidson approach, in terms of storage and floating point operations, in two tables. The costs are given for iteration  $m$  of the algorithm:

| Item                         | Storage                    |
|------------------------------|----------------------------|
| Search space                 | $2m n$ -vectors            |
| Residual                     | $2 n$ -vector              |
| Approx. eigenvector          | $1 n$ -vector              |
| Projected system             | .5 matrix of order $m$     |
| Eigenvectors of proj. system | $1$ matrix of order $m$    |
| Correction equation          | Depends on selected solver |

| Action                       | Work                                                     |
|------------------------------|----------------------------------------------------------|
| Search basis                 | $m + 1$ dot products, $m$ updates in iteration $m$       |
| Projected system             | $m$ dots                                                 |
| Eigensystem projected system | $O(m^3)$                                                 |
| Residual                     | 1 matrix-vector product, 1 update<br>or $m$ -fold update |
| Approx. eigenvector          | $m$ -fold update                                         |
| Correction equation          | Depends on choice of solver                              |

### 4.7.3 Restart and Deflation

**Restart Strategy.** The increasing storage or the computational overhead, for increasing dimension  $m$  of the subspace, may make it necessary to restart. An obvious way to restart is to take the most recent approximation for the desired eigenvector. However, this may not be the most efficient strategy for restarting. With any restart by a single vector we discard possibly valuable information that is contained in the remaining part of the subspace. Unless we have an invariant subspace, all vectors in the subspace contain information for the wanted eigenvector. After restart with one single vector we see the effect of the lost information by a slowdown in the speed of convergence. Therefore it is often better to restart with a set of vectors representing a subspace that contains more information for the wanted eigenpair, and a good strategy is to restart with the subspace spanned by the Ritz vectors of a small number of the Ritz values closest to a specified target value.

**Deflation.** When a Ritz value is close enough to an eigenvalue, the remaining part of the current subspace will already have rich components in nearby eigenpairs, since we have selected in all steps the Ritz vectors for Ritz values close to the desired eigenvalue. We can use this information as the basis for a subspace for the computation of a next eigenvector. In order to avoid the old eigenvector reentering the computational process, we make the new search vectors in the Jacobi–Davidson algorithm explicitly orthogonal to the computed eigenvectors. This technique is called explicit *deflation*. We will discuss this in slightly more detail.

Let  $\tilde{x}_1, \dots, \tilde{x}_{k-1}$  denote the accepted eigenvector approximations and let us assume that these vectors are orthonormal. The matrix  $\tilde{X}_{k-1}$  has the vectors  $\tilde{x}_j$  as its columns. In order to find the next eigenvector  $\tilde{x}_k$ , we apply the Jacobi–Davidson algorithm to the deflated matrix<sup>5</sup>

$$(I - \tilde{X}_{k-1} \tilde{X}_{k-1}^*) A (I - \tilde{X}_{k-1} \tilde{X}_{k-1}^*),$$

and this leads to a correction equation of the form

$$P_m (I - \tilde{X}_{k-1} \tilde{X}_{k-1}^*) (A - \theta_j^{(m)} I) (I - \tilde{X}_{k-1} \tilde{X}_{k-1}^*) P_m t_j^{(m)} = -r_j^{(m)}, \quad (4.50)$$

with  $P_m \equiv (I - u_j^{(m)} u_j^{(m)*})$ , that has to be solved for the correction  $t_j^{(m)}$  to each new eigenvector approximation  $u_j^{(m)}$ , with corresponding Ritz value  $\theta_j^{(m)}$ . In [172] it is shown, by numerical evidence, that the explicit deflation against the vectors represented

<sup>5</sup>Deflation with approximate eigenvectors may introduce an error of order  $\epsilon^2$  on the eigenvalues, provided that the computed eigenvalues are well separated from the remaining ones [353, Section 5.1].

by  $\tilde{X}_{k-1}$  is highly recommended for the correction equation, but it is not necessary to include this deflation in the computation of the projected matrix (the projection of  $A$  onto the subspace spanned by the successive approximations  $v_j$  in the search for the  $k$ th eigenvector). The projected matrix can be computed as  $V_m^* A V_m$ , without significant loss of accuracy.

**Preconditioning.** Preconditioning for an iterative solver like GMRES or CGS, applied with equation (4.50), is only slightly more complicated than in the single-vector case (cf. §4.7.1). Suppose that we have a left preconditioner  $K$  available for the operator  $A - \theta_j^{(m)} I$ . Let  $\tilde{Q}$  denote the matrix  $\tilde{X}_{k-1}$  expanded with  $u_j^{(m)}$  as its  $k$ th column. In this case, the preconditioner  $K$  has to be restricted to the subspace orthogonal to  $\tilde{Q}$  as well, which means that we have to work effectively with

$$\tilde{K} \equiv (I - \tilde{Q}\tilde{Q}^*)K(I - \tilde{Q}\tilde{Q}^*).$$

Similar to the single-vector case, this can be realized in a surprisingly efficient way.

Assume that we use a Krylov solver with initial guess  $t_0 = 0$  and with left-preconditioning for the approximate solution of the correction equation (4.50). Since the starting vector is in the subspace orthogonal to  $\tilde{Q}$ , all iteration vectors for the Krylov solver will be in that space. In that subspace we have to compute the vector  $z \equiv \tilde{K}^{-1}\tilde{A}v$  for a vector  $v$  supplied by the Krylov solver, and

$$\tilde{A} \equiv (I - \tilde{Q}\tilde{Q}^*)(A - \theta_j^{(m)} I)(I - \tilde{Q}\tilde{Q}^*).$$

This is done in two steps. First we compute

$$\tilde{A}v = (I - \tilde{Q}\tilde{Q}^*)(A - \theta_j^{(m)} I)(I - \tilde{Q}\tilde{Q}^*)v = (I - \tilde{Q}\tilde{Q}^*)y$$

with  $y \equiv (A - \theta_j^{(m)} I)v$  since  $\tilde{Q}^*v = 0$ . Then, with left-preconditioning we solve  $z \perp \tilde{Q}$  from

$$\tilde{K}z = (I - \tilde{Q}\tilde{Q}^*)y.$$

Since  $\tilde{Q}^*z = 0$ , it follows that  $z$  satisfies  $Kz = y - \tilde{Q}\vec{\alpha}$  or  $z = K^{-1}y - K^{-1}\tilde{Q}\vec{\alpha}$ . The condition  $\tilde{Q}^*z = 0$  leads to

$$\vec{\alpha} = (\tilde{Q}^*K^{-1}\tilde{Q})^{-1}\tilde{Q}^*K^{-1}y.$$

The vector  $\hat{y} \equiv K^{-1}y$  is solved from  $K\hat{y} = y$  and, likewise,  $\hat{Q} \equiv K^{-1}\tilde{Q}$  is solved from  $K\hat{Q} = \tilde{Q}$ . Note that the last set of equations has to be solved only once in an iteration process for equation (4.50), so that effectively  $i_S + k$  operations with the preconditioner are required for  $i_S$  iterations of the linear solver. Note also that a matrix-vector multiplication with the left-preconditioned operator, in an iteration of the Krylov solver, requires only one operation with  $\tilde{Q}^*$  and  $K^{-1}\tilde{Q}$ , instead of the four actions of the projector operator  $(I - \tilde{Q}\tilde{Q}^*)$ . This has been worked out in the solution template, given in Algorithm 4.18. Note that obvious savings can be realized if the operator  $K$  is kept the same for a number of successive eigenvalue computations (for details, see [412]).

**An Algorithm Template.** The complete algorithm for the Jacobi–Davidson method that includes restart with a number of Ritz vectors and deflation for the computation of a number of eigenpairs is called JDQR [172], since it can be interpreted as an iterative approach for the QR algorithm. The template for this algorithm is given in Algorithm 4.17.

**ALGORITHM 4.17: Jacobi–Davidson Method for  $k_{\max}$  Exterior Eigenvalues for HEP**

```

(1) start with v_0 starting vector and τ target value.
(2) $t = v_0$, $k = 0$, $m = 0$; $\tilde{X} = []$,
(3) while $k < k_{\max}$
(4) for $i = 1, \dots, m$
(5) $t = t - (v_i^* t)v_i$
(6) end for
(7) $m = m + 1$; $v_m = t / \|t\|_2$, $v_m^A = Av_m$
(8) for $i = 1, \dots, m$
(9) $M_{i,m} = v_i^* v_m^A$
(10) end for
(11) compute eigendecomposition $M = S\Theta S^*$ with the sorted pairs:
 $|\theta_i - \tau| \geq |\theta_{i-1} - \tau|$
(12) $u = Vs_1$, $u^A = V^As_1$, $r = u^A - \theta_1 u$
(13) while $\|r\|_2 \leq \epsilon$
(14) $\tilde{\lambda}_{k+1} = \theta_1$, $\tilde{X} = [\tilde{X}, u]$, $k = k + 1$
(15) if $k = k_{\max}$ then stop
(16) $m = m - 1$, $M = 0$
(17) for $i = 1, \dots, m$
(18) $v_i = Vs_{i+1}$, $v_i^A = V^As_{i+1}$,
(19) $M_{i,i} = \theta_{i+1}$, $s_i = e_i$, $\theta_i = \theta_{i+1}$
(20) end for
(21) $u = v_1$, $r = v_1^A - \theta_1 u$
(22) end while
(23) if $m \geq m_{\max}$ then
(24) $M = 0$
(25) for $i = 2, \dots, m_{\min}$
(26) $v_i = Vs_i$, $v_i^A = V^As_i$, $M_{i,i} = \theta_i$
(27) end for
(28) $v_1 = u$, $v_1^A = u^A$, $M_{1,1} = \theta_1$, $m = m_{\min}$
(29) end if
(30) $\theta = \theta_1$, $\tilde{Q} = [\tilde{X}, u]$
(31) solve $t \perp \tilde{Q}$ (approximately) from:
 $(I - \tilde{Q}\tilde{Q}^*)(A - \theta I)(I - \tilde{Q}\tilde{Q}^*)t = -r$
(32) end while

```

To apply this algorithm we need to specify a starting vector  $v_0$ , a tolerance  $\epsilon$ , a target value  $\tau$ , and a number  $k_{\max}$  that specifies how many eigenpairs near  $\tau$  should be computed. The value of  $m_{\max}$  denotes the maximum dimension of the search subspace. If it is exceeded, a restart takes place with a subspace of specified dimension  $m_{\min}$ .

On completion typically the  $k_{\max}$  largest eigenvalues are delivered when  $\tau$  is chosen larger than  $\lambda_{\max}(A)$ ; the  $k_{\max}$  smallest eigenvalues are delivered if  $\tau$  is chosen smaller than  $\lambda_{\min}(A)$ . The computed eigenpairs  $(\tilde{\lambda}_j, \tilde{x}_j)$ ,  $\|\tilde{x}_j\|_2 = 1$ , satisfy  $\|A\tilde{x}_j - \tilde{\lambda}_j\tilde{x}_j\|_2 \leq j\epsilon$ , where  $\tilde{x}_j$  denotes the  $j$ th column of  $\tilde{X}$ .

In principle, this algorithm computes the  $k_{\max}$  eigenvalues closest to a specified target value  $\tau$ . This is only reliable if the  $k_{\max}$  largest or  $k_{\max}$  smallest eigenvalues are wanted. For interior sets of eigenvalues we will describe safer techniques in §4.7.4. We will now comment on some parts of the algorithm in view of our discussions in previous subsections.

(2) Initialization phase. The search subspace is initialized with  $t = v_0$ .

(4)–(6) The new expansion vector for the search subspace is made orthogonal with respect to the current search subspace by means of modified Gram–Schmidt. This can be replaced, for improved numerical stability, by the template given in Algorithm 4.14.

If  $m = 0$ , this is an empty loop.

(8)–(10) We compute only the upper triangular part of the Hermitian matrix  $M \equiv V^*AV$  (of order  $m$ ).

(11) The eigenproblem for the  $m$  by  $m$  matrix  $M$  can be solved by a standard eigensolver for dense Hermitian eigenproblems from LAPACK. We have chosen to compute the standard Ritz values, which makes the algorithm suitable for computing the largest or smallest  $k_{\max}$  eigenvalues of  $A$ . If one wishes to compute  $k_{\max}$  eigenvalues somewhere in the interior of the spectrum, the usage of harmonic Ritz values is advocated; see §4.7.4.

The matrix  $V$  denotes the  $n$  by  $m$  matrix with columns  $v_j$ ,  $V^A \equiv AV$  likewise;  $S$  is the  $m$  by  $m$  matrix with columns  $s_j$ , and  $\Theta = \text{diag}(\theta_1, \dots, \theta_m)$ .

(13) The stopping criterion is to accept an eigenvector approximation as soon as the norm of the residual (for the normalized eigenvector approximation) is below  $\epsilon$ . This means that we accept inaccuracies the order of  $\epsilon^2$  in the computed eigenvalues and inaccuracies (in angle) in the eigenvectors in the order of  $\epsilon$ , provided that the associated eigenvalue is simple and well separated from the other eigenvalues; see (4.4).

Occasionally one of the wanted eigenvectors of  $A$  may be undetected, for instance, if  $v_0$  has no component in the corresponding eigenvector direction. For a random start vector this is rather unlikely. (See also note (14) for Algorithm 4.13.)

(16) After acceptance of a Ritz pair, we continue the search for a next eigenpair, with the remaining Ritz vectors as a basis for the initial search space. These vectors are computed in (17)–(20).

(23) We restart as soon as the dimension of the search space for the current eigenvector exceeds  $m_{\max}$ . The process is restarted with the subspace spanned by the  $m_{\min}$  Ritz vectors corresponding to the Ritz values closest to the target value  $\tau$  (and they are computed lines (25)–(27)).

(30)–(31) We have collected the locked (computed) eigenvectors in  $\tilde{X}$ , and the matrix  $\tilde{Q}$  is  $\tilde{X}$  expanded with the current eigenvector approximation  $u$ . This is done in order to obtain a more compact formulation; the correction equation is equivalent to the one in (4.50). The new correction  $t$  has to be orthogonal to the columns of  $\tilde{X}$  as well as to  $u$ .

Of course, the correction equation can be approximately solved by any suitable process, for instance, by a preconditioned Krylov subspace method. Because of the occurrence of  $\tilde{Q}$  one has to be careful with the use of preconditioners for the matrix  $A - \theta I$ . The inclusion of preconditioners can be done following the same principles as for the single-vector Jacobi–Davidson algorithm; see Algorithm 4.18 for a template. Make sure that the starting vector  $t_0$  for an iterative solver satisfies the orthogonality constraints  $\tilde{Q}^* t_0 = 0$ . Note that significant savings per step can be made in Algorithm 4.18 if  $K$  is kept the same for a (few) Jacobi–Davidson iterations. In that case columns of  $\tilde{Q}$  can be saved from previous steps. Also the matrix  $\mathcal{M}$  in Algorithm 4.18 can be updated from previous steps, as well as its  $\mathcal{LU}$  decomposition.

**ALGORITHM 4.18: Approximate Solution of the Deflated Jacobi–Davidson HEP Correction Equation**

“Solve” with left preconditioner  $\tilde{K} \equiv (I - \tilde{Q}\tilde{Q}^*)K(I - \tilde{Q}\tilde{Q}^*)$ ,  
for  $\tilde{A} \equiv (I - \tilde{Q}\tilde{Q}^*)(A - \theta I)(I - \tilde{Q}\tilde{Q}^*)$ :

- (1) solve  $\hat{Q}$  from  $K\hat{Q} = \tilde{Q}$
- (2) compute  $\mathcal{M} = \tilde{Q}^*\hat{Q}$
- (3) decompose  $\mathcal{M} = \mathcal{LU}$
- (4) compute  $\tilde{r} \equiv \tilde{K}^{-1}r$  as:
  - (5) (b') solve  $\hat{r}$  from  $K\hat{r} = r$
  - (6) (c')  $\vec{\gamma} = \tilde{Q}^*\hat{r}$
  - (7) solve  $\vec{\beta}$  from  $\mathcal{L}\vec{\beta} = \vec{\gamma}$
  - (8) solve  $\vec{\alpha}$  from  $\mathcal{U}\vec{\alpha} = \vec{\beta}$
  - (9) (d')  $\tilde{r} = \hat{r} - \tilde{Q}\vec{\alpha}$
- (10) apply a Krylov subspace method with start  $t_0 = 0$ , operator  $\tilde{K}^{-1}\tilde{A}$ ,  
and right-hand side  $-\tilde{r}$ ;  $z = \tilde{K}^{-1}\tilde{A}v$  for given  $v$  is computed as:
  - (11) (a)  $y = (A - \theta I)v$
  - (12) (b) solve  $\hat{y}$  from  $K\hat{y} = y$
  - (13) (c)  $\vec{\gamma} = \tilde{Q}^*\hat{y}$
  - (14) solve  $\vec{\beta}$  from  $\mathcal{L}\vec{\beta} = \vec{\gamma}$
  - (15) solve  $\vec{\alpha}$  from  $\mathcal{U}\vec{\alpha} = \vec{\beta}$
  - (16) (d)  $z = \hat{y} - \tilde{Q}\vec{\alpha}$

#### 4.7.4 Computing Interior Eigenvalues

If one is searching for the eigenpair with the smallest or largest eigenvalue only, then the obvious restart approach works quite well, but often it does not do very well if one is interested in an interior eigenvalue. The problem is that the Ritz values converge

monotonically towards exterior eigenvalues, and a Ritz value that is close to a target value in the interior of the spectrum may be well on its way to some other exterior eigenvalue. It may even be the case that the corresponding Ritz vector has only a small component in the direction of the desired eigenvector. It will be clear that such a Ritz vector represents a poor candidate for restart and the question is, What is a better vector for restart? One answer is given by the so-called *harmonic Ritz* vectors, discussed in §3.2; see also [331, 349, 411].

As we have seen, the Jacobi–Davidson methods generate basis vectors  $v_1, \dots, v_m$  for a subspace  $\mathcal{V}_m$ . For the projection of  $A$  onto this subspace we compute the vectors  $w_j \equiv Av_j$ . The harmonic Ritz values are inverses of the Ritz values of  $A^{-1}$ , with respect to the subspace spanned by the  $w_j$ . They can be computed without inverting  $A$ , since a harmonic Ritz pair  $(\tilde{\theta}_j^{(m)}, \tilde{u}_j^{(m)})$  satisfies

$$A\tilde{u}_j^{(m)} - \tilde{\theta}_j^{(m)}\tilde{u}_j^{(m)} \perp \mathcal{W}_m \equiv \text{span}(w_1, \dots, w_m) \quad (4.51)$$

for  $\tilde{u}_j^{(m)} \in V_m$  and  $\tilde{u}_j^{(m)} \neq 0$ . This implies that the harmonic Ritz values are the eigenvalues of the pencil  $(W_m^*AV_m, W_m^*V_m)$ , or, since  $W_m = AV_m$ :

$$W_m^*W_m s_j^{(m)} - \tilde{\theta}_j^{(m)} W_m^*V_m s_j^{(m)} = 0.$$

For stability reasons we orthonormalize the columns of  $W_m$  and transform the columns of  $V_m$  accordingly. This also further simplifies the equation: we see that the harmonic Ritz values are the inverses of the eigenvalues of the symmetric matrix  $W_m^*V_m$ .

In [349] it is shown that for Hermitian  $A$  the harmonic Ritz values converge monotonically towards the smallest nonzero eigenvalues in absolute value. Note that the harmonic Ritz values are unable to identify a zero eigenvalue of  $A$ , since that would correspond to an infinite eigenvalue of  $A^{-1}$ . Likewise, the harmonic Ritz values for the shifted matrix  $A - \tau I$  converge monotonically towards eigenvalues  $\lambda \neq \tau$  closest to the target value  $\tau$ . Fortunately, the search subspace  $\mathcal{V}_m$  for the shifted matrix and the unshifted matrix coincide, which facilitates the computation of harmonic Ritz pairs for any shift. The harmonic Ritz vector for the shifted matrix, corresponding to the harmonic Ritz value closest to  $\tau$ , can be interpreted as maximizing a Rayleigh quotient for  $(A - \tau I)^{-1}$ . It represents asymptotically the best information that is available for the wanted eigenvalue, and hence it represents asymptotically the best candidate as a starting vector after restart, provided that  $\tau \neq \lambda$ .

For harmonic Ritz values, the correction equation has to take into account the orthogonality with respect to  $A\tilde{u}_j^{(m)}$ , and this leads to skew projections. We can use orthogonal projections in the following way. If  $\tilde{u} = \tilde{u}_j^{(m)}$  is the selected approximation of an eigenvector, the Rayleigh quotient  $\theta = \tilde{u}^*A\tilde{u}/\tilde{u}^*\tilde{u}$  leads to the residual with smallest norm; that is, with  $r \equiv A\tilde{u} - \theta\tilde{u}$ , we have that  $\|r\|_2 \leq \|A\tilde{u} - \mu\tilde{u}\|_2$  for any scalar  $\mu$ , including the harmonic Ritz value  $\mu = \tilde{\theta}_j^{(m)}$ . Moreover, the residual  $r$  for the Rayleigh quotient is orthogonal to  $\tilde{u}$ . This makes  $r$  “compatible” with the operator  $(I - uu^*)(A - \theta I)(I - uu^*)$  in the correction equation. Here  $u \equiv \tilde{u}/\|\tilde{u}\|_2$ .

An algorithm for the Jacobi–Davidson method based on harmonic Ritz values and vectors, combined with restart and deflation, is given in Algorithm 4.19. The algorithm can be used for the computation of a number of successive eigenvalues immediately to the right of the target value  $\tau$ .

**ALGORITHM 4.19: Jacobi–Davidson Method for  $k_{\max}$  Interior Eigenvalues at the Right Side Nearest to  $\tau$  for HEP**

```

(1) $t = v_0, k = 0, m = 0, \tilde{X} = []$
(2) while $k < k_{\max}$
(3) $w = (A - \tau I)t$
(4) for $i = 1, \dots, m$
(5) $\gamma = w_i^* w, w = w - \gamma w_i, t = t - \gamma v_i$
(6) end for
(7) $m = m + 1, w_m = w/\|w\|_2, v_m = t/\|w\|_2$
(8) for $i = 1, \dots, m$
(9) $M_{i,m} = w_i^* v_m$
(10) end for
(11) compute eigendecomposition $M = S\tilde{\Theta}S^*$ with sorted eigenvalues:
 $\tilde{\theta}_1 \geq \tilde{\theta}_2 \geq \dots$
(12) $\tilde{u} = Vs_1, \mu = \|\tilde{u}\|_2, u = \tilde{u}/\mu, \vartheta = \tilde{\theta}_1/\mu^2$
(13) $\tilde{w} = Ws_1, r = \tilde{w}/\mu - \vartheta u$
(14) while $\|r\|_2 \leq \epsilon$
(15) $k = k + 1, \tilde{X} = [\tilde{X}, u], \tilde{\lambda}_k = \vartheta + \tau$
(16) if $k = k_{\max}$ then stop
(17) $m = m - 1, M = 0$
(18) for $i = 1, \dots, m$
(19) $v_i = Vs_{i+1}, w_i = Ws_{i+1}$
(20) $M_{i,i} = \tilde{\theta}_{i+1}, s_i = e_i, \tilde{\theta}_{i+1} = \tilde{\theta}_i$
(21) end for
(22) $\mu = \|v_1\|_2, \vartheta = \tilde{\theta}_2/\mu^2, u = v_1/\mu, r = w_1/\mu - \vartheta u$
(23) end while
(24) if $m \geq m_{\max}$ then
(25) $M = 0$
(26) for $i = 2, \dots, m_{\min}$
(27) $v_i = Vs_i, w_i = Ws_i, M_{i,i} = \tilde{\theta}_i$
(28) end for
(29) $w_1 = \tilde{w}, v_1 = \tilde{u}, M_{1,1} = \tilde{\theta}_1, m = m_{\min}$
(30) end if
(31) $\theta = \vartheta + \tau, \tilde{Q} = [\tilde{X}, u]$
(32) solve $t \perp \tilde{Q}$ (approximately) from:
 $(I - \tilde{Q}\tilde{Q}^*)(A - \theta I)(I - \tilde{Q}\tilde{Q}^*)t = -r$
(33) end while

```

To apply this algorithm we need to specify a starting vector  $v_0$ , a tolerance  $\epsilon$ , a target value  $\tau$ , and a number  $k_{\max}$  that specifies how many eigenpairs near  $\tau$  should be computed. The value of  $m_{\max}$  denotes the maximum dimension of the search subspace. If it is exceeded, a restart takes place with a subspace of specified dimension  $m_{\min}$ .

On completion, the  $k_{\max}$  eigenvalues at the right side nearest to  $\tau$  are delivered. The computed eigenpairs  $(\tilde{\lambda}_j, \tilde{x}_j)$ ,  $\|\tilde{x}_j\|_2 = 1$ , satisfy  $\|A\tilde{x}_j - \tilde{\lambda}_j\tilde{x}_j\|_2 \leq j\epsilon$ , where  $\tilde{x}_j$  denotes the  $j$ th column of  $\tilde{X}$ .

For exterior eigenvalues a simpler algorithm has been described in §4.7.3. We will now comment on some parts of the algorithm in view of our discussions in previous subsections.

(1) Initialization phase.

(3)–(7) The vector  $w = (A - \tau I)t$  is made orthogonal with respect to the current test subspace  $\mathcal{W}_m$  by means of modified Gram–Schmidt. This can be replaced, for improved numerical stability, by an adoption (for the vector  $t$ ) of the template given in Algorithm 4.14.

(8)–(10) The values  $M_{i,j}$  represent elements of the square  $m$  by  $m$  matrix  $M \equiv W^*V$ , where  $V$  denotes the  $n$  by  $m$  matrix with columns  $v_j$ , and likewise  $W$ . Because  $M$  is Hermitian, only the upper triangular part of this matrix is computed.

(11)–(13) At this point the eigenpairs for the problem  $Ms = \tilde{\theta}s$  should be computed. This can be done with a suitable routine for Hermitian dense matrices from LAPACK. Note that the harmonic Ritz values are the inverses of the eigenvalues of  $M$ . We have to compute the Rayleigh quotient  $\theta$  for  $Vs_1$ , and next normalize  $Vs_1$ , in order to compute a proper residual  $r \perp Vs_1$ . We have used that  $s_1^*V^*(A - \tau I)Vs_1 = s_1^*M^*s_1 = \tilde{\theta}_1$ .

The vectors  $s_j$  are the columns of  $m$  by  $m$  matrix  $S$  and  $\tilde{\Theta} = \text{diag}(\tilde{\theta}_1, \dots, \tilde{\theta}_m)$ .

(14) The stopping criterion is to accept an eigenvector approximation as soon as the norm of the residual (for the normalized eigenvector approximation) is below  $\epsilon$ . This means that we accept inaccuracies in the order of  $\epsilon^2$  in the computed eigenvalues, and inaccuracies (in angle) in the eigenvectors of  $\mathcal{O}(\epsilon)$ , provided that the associated eigenvalue is simple and well separated from the others; see (4.4).

Detection of all wanted eigenvalues cannot be guaranteed; see note (14) for Algorithm 4.13 and note (13) for Algorithm 4.17.

(17) This is a restart after acceptance of an approximate eigenpair. The restart is slightly more complicated since two subspaces are involved. Recomputation of the spanning vectors from these subspaces is done in (18)–(21).

(24) At this point we have a restart when the dimension of the subspace exceeds  $m_{\max}$ . After a restart, the Jacobi–Davidson iterations are resumed with a subspace of dimension  $m_{\min}$ . The selection of this subspace is based on the harmonic Ritz values nearest to the target  $\tau$ .

(31)–(32) The deflation with computed eigenvectors is represented by the factors with  $\tilde{X}$ . The matrix  $\tilde{X}$  has the computed eigenvectors as its columns. If a left preconditioner  $K$  is available for the operator  $A - \theta I$ , then with a Krylov solver similar reductions are realizable, as in the situation for exterior eigenvalues. A template for the efficient handling of the left-preconditioned operator is given in Algorithm 4.18.

#### 4.7.5 Software Availability

There are MATLAB and FORTRAN implementations of the algorithms described in this section. For information about this software, including how to access it, see the book's homepage ETHOME.

### 4.7.6 Numerical Example

We include a numerical example for testing purposes, so that potential users of the Jacobi–Davidson algorithms can verify and compare their results.

The symmetric matrix  $A$  is of dimension  $n = 1000$ . The diagonal entries are  $a_{i,i} = i$ , the codiagonal entries are  $a_{i,i-1} = a_{i,i+1} = 0.5$ , and furthermore,  $a_{1000,1} = a_{1,1000} = 0.5$ . All other entries are zero. This example has been taken from [88] and is discussed, in the context of the Jacobi–Davidson algorithm, in [411, p. 410].

We use Algorithm 4.17 for the computation of the  $k_{\max} = 10$  largest eigenvalues. The input parameters have been chosen as follows. The starting vector  $v_0 = (0.01, \dots, 0.01, 1)^T$ . The tolerance is  $\epsilon = 10^{-8}$ . The subspace dimension parameters are  $m_{\min} = 10$ ,  $m_{\max} = 15$ , and the target value  $\tau = 1001$ .

We show graphically the norm of the residual vector as a function of the iteration number in Figure 4.5. Every time the norm is less than  $\epsilon$ , we have determined an eigenvalue within this precision, and the iteration is continued with deflation for the next eigenvalue. The four pictures represent, lexicographically, the following different situations:

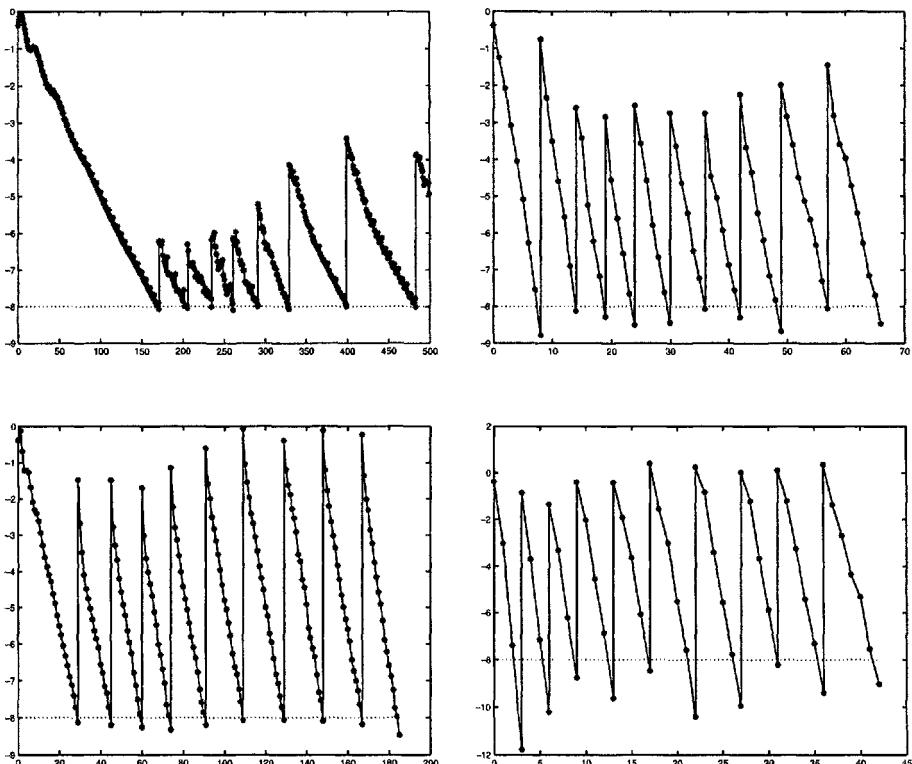


Figure 4.5: Jacobi–Davidson for exterior eigenvalues with several strategies for solving the correction equation.

1. Top left: This shows what happens when  $t$ , in item (32) of Algorithm 4.17, is simply taken as  $t = -r$ . In exact arithmetic, this should deliver the same Ritz values as the Arnoldi algorithm (assuming for Arnoldi a similar restart strategy as in Algorithm 4.17).
2. Top right: Here we have applied a simple preconditioner  $K = \text{diag}(A)$ , as in Algorithm 4.18, without further subspace acceleration; that is, we stop after step (d'). This is equivalent to the method currently in use among quantum chemists [344].
3. Bottom left: This gives the iteration results, for the case where the correction equation has been solved with 5 steps of MINRES, without preconditioning (Algorithm 4.17 with  $K = I$ ).
4. Bottom right: Here we have used preconditioning as in Algorithm 4.17, with  $K = \text{diag}(A)$  and 5 steps of GMRES (note that it would have been more efficient to use MINRES, but this requires two-sided preconditioning, for which we did not supply the algorithm).

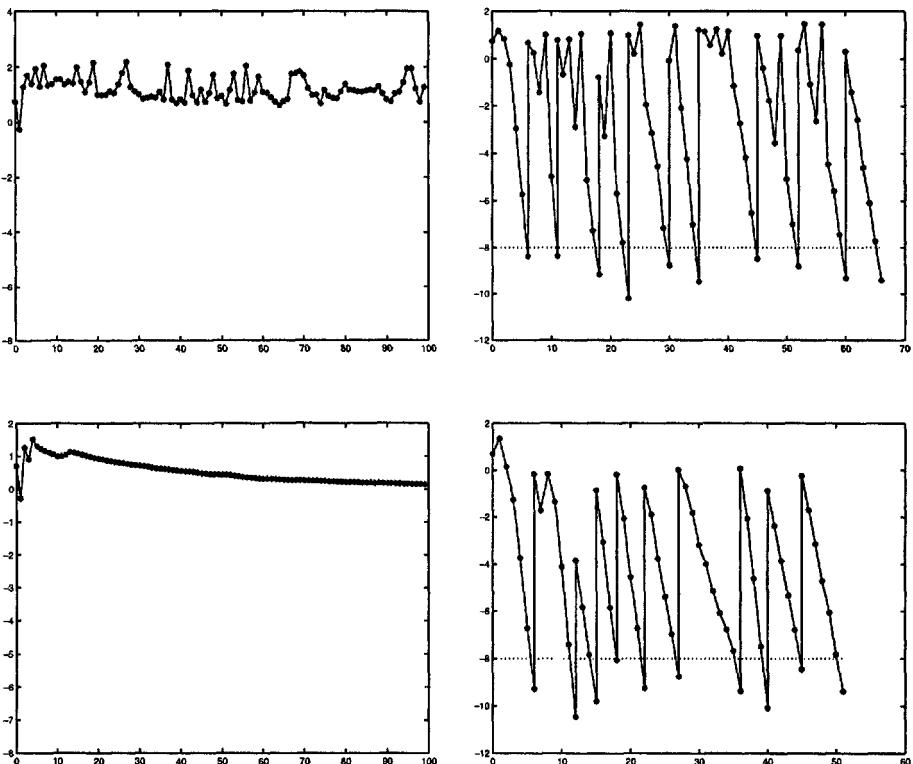


Figure 4.6: Jacobi–Davidson for exterior eigenvalues (top) and interior eigenvalues (bottom). The correction equations have been solved with 5 steps of plain GMRES (left) and with 5 steps of preconditioned GMRES (right).

In Figure 4.6, we give the convergence history for interior eigenvalues, as obtained with Algorithm 4.17 (top parts) and with Algorithm 4.19 (bottom parts), with the following input specifications:  $v_0 = (0.01, \dots, 0.01, 1)^T$ ,  $\tau = 900.5$ ,  $\epsilon = 10^{-8}$ ,  $k_{\max} = 10$ ,  $m_{\min} = 5$ , and  $m_{\max} = 10$ . Again, every time the curve gets below  $\epsilon$ , this indicates convergence of an approximated eigenvalue to within that tolerance. For all figures, we used 5 steps of GMRES to solve the correction equation in (32). For the left figures, we did not use preconditioning. For the right figures, we preconditioned GMRES with  $K = \text{diag}(A) - \tau I$ , as in Algorithm 4.18.

## 4.8 Stability and Accuracy Assessments

*Z. Bai and R. Li*

This section reviews some basic results that are readily applicable to assess the accuracy of computed eigenvalues and eigenvectors. We only assume the availability of residual vectors, which are usually available upon the exit of a successful computation. If not, they can be computed at marginal cost afterwards.

For the treatment of error estimation of computed eigenvalues and eigenvectors of dense Hermitian eigenproblems, see Chapter 4 of the *LAPACK Users' Guide* [12].

**Residual Vector.** Let  $\tilde{\lambda}$  denote a computed eigenvalue, and let  $\tilde{x}$  be its corresponding computed eigenvector. For simplicity, we normalize the computed eigenvector so that  $\|\tilde{x}\|_2 = 1$ . The corresponding *residual vector* or *residual error* is defined by

$$r = A\tilde{x} - \tilde{\lambda}\tilde{x}.$$

Ideally, we would like to have  $r = 0$ , but in practice  $r \approx 0$ . It is conceivable that a small residual error implies good accuracy in the computed  $\tilde{\lambda}$  and  $\tilde{x}$ . We are interested in knowing how accurate the computed  $\tilde{\lambda}$  and  $\tilde{x}$  are, given  $r$ .

**Transfer Residual Error to Backward Error.** There are Hermitian matrices  $E$  such that  $\tilde{\lambda}$  and  $\tilde{x}$  are an exact eigenvalue and its corresponding eigenvector of  $A + E$ , i.e.,

$$(A + E)\tilde{x} = \tilde{\lambda}\tilde{x}.$$

One such  $E$  is

$$E = -r\tilde{x}^* - \tilde{x}r^* + (\tilde{x}^* A\tilde{x} - \tilde{\lambda})\tilde{x}\tilde{x}^*. \quad (4.52)$$

We are interested in such matrices  $E$  with smallest possible norms. It turns out the best possible  $E = E_2$  for the spectral norm  $\|\cdot\|_2$  and the best possible  $E = E_F$  for Frobenius norm  $\|\cdot\|_F$  satisfy

$$\|E_2\|_2 = \|r\|_2 \quad \text{and} \quad \|E_F\|_F = \sqrt{2\|r\|_2^2 - (\tilde{x}^* A\tilde{x} - \tilde{\lambda})^2}; \quad (4.53)$$

see, e.g., [256, 431]. In fact,  $E_F$  is given explicitly by (4.52). So if  $\|r\|_2$  is small, the computed  $\tilde{\lambda}$  and  $\tilde{x}$  are an exact eigenpair of *nearby* matrices. Error analysis of this kind is called *backward error analysis*, and matrices  $E$  are *backward errors*.

We say an algorithm that delivers an approximate eigenpair  $(\tilde{\lambda}, \tilde{x})$  is  $\tau$ -backward stable for the pair with respect to the norm  $\|\cdot\|$  if it is an exact eigenpair for  $A+E$  with  $\|E\| \leq \tau$ . With this definition in mind, statements can be made about the numerical stability of the algorithm which computes the eigenpair  $(\tilde{\lambda}, \tilde{x})$ . By convention, an algorithm is called *backward stable* if  $\tau = O(\epsilon_M \|A\|)$ , where  $\epsilon_M$  is the machine precision.

**Error Bounds for Computed Eigenvalues.** For the Hermitian eigenproblem, the  $i$ th largest eigenvalue of  $A$  differs from the  $i$ th largest eigenvalue of  $A+E$  by at most  $\|E\|_2$ . Therefore a small backward error implies a small (*forward*) error in the computed eigenvalue, i.e.,

$$|\tilde{\lambda} - \lambda| \leq \|E\|_2 = \|r\|_2 \quad (4.54)$$

for some eigenvalue  $\lambda$  of  $A$ .

With more information, a better error bound can be obtained. Let us assume that  $(\tilde{\lambda}, \tilde{x})$  is an approximation of the eigenpair  $(\lambda, x)$  of  $A$ . The “best”  $\tilde{\lambda}$  corresponding to  $\tilde{x}$  is the Rayleigh quotient  $\tilde{\lambda} = \tilde{x}^* A \tilde{x}$ , so we assume that  $\tilde{\lambda}$  has this value. Suppose that  $\tilde{\lambda}$  is closer to  $\lambda$  than any other eigenvalues of  $A$ , and let  $\delta$  be the *gap* between  $\tilde{\lambda}$  and any other eigenvalue:  $\delta = \min_{\lambda_j \neq \lambda} |\tilde{\lambda} - \lambda_j|$ . Then we have

$$|\tilde{\lambda} - \lambda| \leq \frac{\|r\|_2^2}{\delta}. \quad (4.55)$$

This improves (4.54) if the gap  $\delta$  is reasonably big. In practice we can always pick the better one.

Note that (4.55) needs information on  $\delta$ , besides the residual error  $r$ . Usually such information is available after a successful computation by, e.g., a Lanczos algorithm with SI, which usually delivers eigenvalues in the neighborhood of a shift and consequently yields good information on  $\delta$ . This comment also applies to the bound in (4.56) below.

**Error Bound for Computed Eigenvectors.** Keeping the assignments to  $\tilde{\lambda}$ ,  $\lambda$ , and  $\delta$  as above, and letting  $x$  be the eigenvector of  $A$  corresponding to  $\lambda$ , we have the following error bound of the computed eigenvector  $\tilde{x}$  [101, 425]:

$$\sin \theta(x, \tilde{x}) \leq \frac{\|r\|_2}{\delta}. \quad (4.56)$$

**Remarks on Clustered Eigenvalues.** In the case when the eigenvalue  $\lambda$  has one or more other eigenvalues of  $A$  close by, in other words, when  $\lambda$  belongs to clustered eigenvalues, as guaranteed by (4.54), the computed  $\tilde{\lambda}$  is still accurate as long as  $\|r\|_2$  is tiny, but the computed eigenvector  $\tilde{x}$  may be inaccurate because of the appearance of the gap  $\delta$  in the denominator of (4.56). It turns out that each individual eigenvector associated with the clustered eigenvalues is very sensitive to perturbations, but the eigenspace spanned by all the eigenvectors associated with the clustered eigenvalues is not. Thus, for the clustered eigenvalues, we should instead compute the entire eigenspace. A theory along the lines given above can be established, starting with a residual matrix

$$R = A\tilde{X} - \tilde{X}\tilde{\Lambda}.$$

where  $\tilde{\Lambda}$  is diagonal with diagonal entries consisting of approximations to all the eigenvalues in the cluster, and the columns of  $\tilde{X}$  are the corresponding approximate eigenvectors. Assume that  $\tilde{X}$  has orthonormal columns and that  $\tilde{\Lambda}$  is closer to  $\Lambda$ , the diagonal matrix whose diagonal entries consist of all the eigenvalues in the cluster. Let  $X$  be the eigenvector matrix associated with  $\Lambda$ , and let  $\delta$  be the smallest difference between any approximate eigenvalue in the diagonal of  $\tilde{\Lambda}$  and those eigenvalues of  $A$  not presented in the diagonal of  $\Lambda$ . Then [101]

$$\|\sin \Theta(X, \tilde{X})\|_F \leq \frac{\|R\|_F}{\delta},$$

where  $\Theta(X, \tilde{X})$  is diagonal with diagonal entries being the arccosines of the singular values of  $X^* \tilde{X}$ . Because of the way it is defined, this gap is expected to be big and thus  $\|\sin \Theta(X, \tilde{X})\|_F$  will be small as long as  $\|R\|_F$  is.

**Remarks on Eigenvalue Computations to High Relative Accuracy.** Computation of eigenvalues to high accuracy has been attracting lots of attention over the last 10 years or so. Tremendous progress has been made in both theoretical understanding and numerical algorithms. But to give a detailed account is outside of the scope of this book. Interested readers are referred to the literature. On the algorithmic side there are the Demmel–Kahan QR method for bidiagonal singular value computations [123] and (two-sided) Jacobi methods for the eigenvalue problems of positive definite matrices. For the singular value computations [124, 317, 406], there are the bisection method for scaled diagonally dominant matrices [40], and for matrices with acyclic graphs [117, 255], new implementations of the qd method [168, 360] and Demmel’s algorithms for structured matrices [115]. More recently, [118] showed how to compute SVDs to high relative accuracy for matrices that can be factored accurately as  $B = X\Gamma Y^*$ , where  $\Gamma$  is diagonal and  $X$  and  $Y$  are any well-conditioned matrices. On the theoretical side, analogous results to many celebrated theorems for absolute perturbations  $A \rightarrow \tilde{A} = A + \Delta A$  are obtained for perturbations that are multiplicative,  $A \rightarrow \tilde{A} = D^* A E$  ( $E = D$  when  $A$  is Hermitian) [157, 300, 301, 302, 303, 297].

*This page intentionally left blank*

# Chapter 5

## Generalized Hermitian Eigenvalue Problems

### 5.1 Introduction

A generalized Hermitian eigenvalue problem (GHEP) is given by

$$Ax = \lambda Bx, \quad (5.1)$$

where  $A$  and  $B$  are Hermitian,  $A^* = A$ , and  $B^* = B$ . We call the pair  $\{A, B\}$  of matrices in (5.1) a *matrix pencil*. In this chapter we make the additional assumption that  $A$  or  $B$  or  $\alpha A + \beta B$  for some scalars  $\alpha$  and  $\beta$  is positive definite, in which case we talk about a *Hermitian definite pencil*. This assumption is true for a wide class of practically important cases, and the theory is very closely related to the standard Hermitian eigenproblem, as expounded in Chapter 4. If no positive definite combination exists, we could as well regard  $\{A, B\}$  as a general pencil and use the theory and algorithms described in Chapter 8.

The nonstandard case, where  $\alpha A + \beta B$  is positive definite, may be reduced to the standard case when  $B$  is positive definite, by noting that the pencil

$$(A - \theta(\alpha A + \beta B))x = 0 \quad (5.2)$$

has eigenvalues  $\theta_i = \lambda_i / (\beta + \alpha \lambda_i)$  and the same eigenvectors as the original pencil (5.1). One may apply any algorithm applicable for positive definite  $B$  to this modified pencil and recover the  $\lambda_i$  from the  $\theta_i$ .

The GHEP (5.1) has  $n$  real eigenvalues  $\lambda_i$ , which we may order increasingly so that  $\lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$ . Several eigenvalues may coincide, as in the standard case, except that some eigenvalues may be infinite. If the matrices  $A$  and  $B$  are positive definite,  $\lambda_1 > 0$ , but if  $A$  is positive semidefinite we can only say that  $\lambda_1 \geq 0$ .

When the pencil  $\{A, B\}$  is Hermitian definite, it is possible to find  $n$  mutually  $B$ -orthogonal eigenvectors,  $x_i$ ,  $i = 1, \dots, n$ , so that

$$A = X\Lambda X^*, \quad B = XX^*, \quad (5.3)$$

where  $\Lambda = \text{diag}(\lambda_i)$  and  $X = [x_1, x_2, \dots, x_n]$ . The eigenvectors  $x_i$  are not unique, but there is a unique *reducing subspace* for each different eigenvalue. For a Hermitian definite pencil, the reducing subspace is of the same dimension as the multiplicity of the eigenvalue. It is important to keep in mind that when a couple of eigenvalues coincide, their eigenvectors lose their individuality: there is no way of saying that one set of vectors comprises the eigenvectors of a multiple eigenvalue.

There are some cases where the matrix  $B$  is singular (positive semidefinite). We still have a Hermitian definite pencil provided that  $Ax \neq 0$  for all vectors  $x$  in the null space of  $B$ . The pencil (5.1) then has a  $p$ -fold infinite eigenvalue with the null space of  $B$  as its reducing subspace. In most practical cases, we are not interested in computing these infinite eigenvalues, since they correspond to constraints, symmetries or rigid body modes of the physical application that gave rise to the eigenvalue computation.

Eigenvalues  $\{\lambda_i\}$  of  $A - \lambda B$  may be well-conditioned or ill-conditioned. If  $B = I$  (or close to it), the eigenvalues are well-conditioned (close to it) as for the Hermitian eigenproblem. But if  $|x_i^* B x_i|$  is small, where  $x_i$  is a unit eigenvector of  $\lambda_i$ ,  $\lambda_i$  can be very ill-conditioned. We give a more detailed account of error assessment for the computed eigenvalues and eigenvectors in §5.7.

**Overview of Available Algorithms.** In many cases we can transform the pencil (5.1) into a standard HEP discussed in Chapter 4, or in some cases into a standard NHEP discussed in Chapter 7, and use any of the algorithms described in these two chapters. We give a short review of such transformations in §5.2.

The main body of this chapter is devoted to algorithms that take a special form for generalized Hermitian pencils (5.1):

*Power method and inverse iteration*, §5.4, is the basic iterative method. It starts with an appropriate starting vector and makes one matrix-vector multiplication and one linear system solve in each iteration. Needing both these operations, it is less appealing than in the standard case, but it is included here because of its simplicity and to clarify its relations to more sophisticated schemes described later.

*Lanczos method*, §5.5, builds up a  $B$  orthogonal basis of a Krylov sequence of vectors in which the matrix operator is represented by a tridiagonal matrix  $T$ , whose eigenvalues yield Ritz approximations to several of the eigenvalues of the original matrix pencil (5.1). We will describe two variants in §5.5, one direct iteration that needs multiplications with  $A$  and the solution of systems with  $B$ , and one shift-and-invert iteration that needs solution of systems with  $(A - \sigma B)$  and multiplication with  $B$ .

*Jacobi–Davidson method*, §5.6, updates a sequence of subspaces, operating with a preconditioned shifted matrix. It uses  $B$  orthogonality and does not need the solution of linear systems. This makes it applicable in several cases when the matrices are too large to allow for solving linear systems, as needed in the other algorithms described in this chapter.

**Summary of Choices.** Let us now turn to Table 5.1, in which we have listed these algorithms and added some information that may be helpful to decide which algorithm to use in a specific situation.

Table 5.1: Summary of algorithms for GHEPs

|         | Appl | Orth       | IE  | CE   | M    | # vec    | Fact                   |
|---------|------|------------|-----|------|------|----------|------------------------|
| Power   | Dir  |            | Yes | Slow | No   | 2        | $B = LL^*$             |
|         | SI   |            | -   | Yes  | Yes  | 3        | $A - \sigma B = LDL^*$ |
| Lanczos | Dir  | local      | Yes | No   | No   | 6        | $B = LL^*$             |
|         | Dir  | sel Borth  | Yes | Slow | No   | Many     | $B = LL^*$             |
|         | SI   | full Borth | -   | Yes  | Yes  | Moderate | $A - \sigma B = LDL^*$ |
| Jac-Dav | Prec | full Borth | Yes | Yes  | Slow | Few      | ILU of $A - \theta B$  |

**Application.** For each algorithm we have distinguished between different ways of running the algorithm:

“Dir” is direct application, where we multiply with  $A$  and solve with  $B$ .

“SI” is shift and invert, which solves systems  $(A - \sigma B)x = b$  for  $x$  and multiplies with  $B$ . This gives the ability to compute a wider choice of eigenvalues in fewer iterations.

“Prec” means application with a preconditioner, for instance, a sparse approximate factorization. This needs less space than shift and invert, but most often it also needs a larger number of matrix-vector multiplies.

**Orthogonalization.** We have also indicated which type of orthogonalization is needed for the bases of subspaces that most of the algorithms use to find eigenvector approximations.  $B$ -orthogonalization either needs extra operations with  $B$  or storage of an auxiliary sequence of vectors. See the algorithm descriptions for details for each algorithm.

**Eigenvalues Sought.** We give three typical cases:

“IE” stands for one or a few isolated eigenvalues at the end of the spectrum.

“CE” stands for one or several eigenvalues at the end of the spectrum, possibly clustered, a typical example being computing the lowest eigenmodes of a vibrating mechanical system, modeled by finite element methods.

“M” means some eigenvalues in the middle of the spectrum, most often in a specified interval, such as seeking all eigenmodes of a mechanical or electrical system excited by an external load of a prescribed frequency.

**Storage and Work.** In the final columns of Table 5.1, we give an estimate of storage needed and extra work for factorizations.

“#vec” gives how many vectors one needs to store. The meaning of 2 is obvious; “Moderate” means a multiple of the number  $m$  of eigenvalues sought, say  $3m$  to  $5m$ . “Few” is smaller than moderate, say  $m + 10$ , and “Many” is larger.

“Fact” indicates whether we need extra matrix storage. “ $LL^*$ ” means a sparse Cholesky factorization, “ $LDL^*$ ,” a sparse symmetric indefinite Gaussian elimination, and “ILU” is an incomplete factorization. It is supposed to be more compact and need less arithmetic work than the other two.

We note that a task such as counting the number of eigenvalues of  $A - \lambda B$  that are smaller than a given real number  $\alpha$  or are in a given interval  $[\alpha, \beta]$  does not require computing the eigenvalues, and so can be much cheaper. The key tool is the matrix inertia as presented in §4.1 (p. 49). It can be extended easily to the case of  $A - \lambda B$  assuming  $B$  positive definite. In summary, let

$$A - \alpha B = L_\alpha D_\alpha L_\alpha^T, \quad A - \beta B = L_\beta D_\beta L_\beta^T$$

be the  $LDL^T$  factorizations of matrices  $A - \alpha B$  and  $A - \beta B$ , respectively, where we assume that  $D_\alpha$  and  $D_\beta$  are nonsingular diagonal matrices. We refer to §10.3 for the information on software availability for the  $LDL^T$  factorization. Then the number of eigenvalues of  $A - \lambda B$  in  $[\alpha, \beta]$  equals  $\nu(D_\beta) - \nu(D_\alpha)$ , where  $\nu(D)$  denotes the number of negative diagonal elements.

## 5.2 Transformation to Standard Problem

We can reformulate the GHEP (5.1) into a standard eigenvalue problem if we can factor either  $B$  or  $A$  or a linear combination of these matrices. See §10.3 for a brief survey of matrix factorizations.

In the former case we make the decomposition

$$B = LL^*, \tag{5.4}$$

e.g., using Cholesky decomposition, and apply a standard Hermitian algorithm to

$$Cy = \lambda y, \tag{5.5}$$

where  $C = L^{-1}AL^{-*}$ . The eigenvectors  $x$  of the original pencil (5.1) can be recovered by a back substitution,

$$x = L^{-*}y.$$

This strategy is advisable whenever the matrix  $B$  is reasonably well conditioned with respect to inversion or when it has a simpler structure than  $A$ , as when  $B$  is diagonal.<sup>1</sup>

The matrix  $C$  is Hermitian, and we may apply any of the iteration algorithms described in Chapter 4. Each time the matrix  $C$  is applied to a vector, the computations are made in the order indicated by the parentheses in the expression

$$y = Cx = L^{-1}(A(L^{-*}x)),$$

that is, a back substitution, followed by a matrix-vector multiply and a forward substitution.

We may also apply a shift-and-invert spectral transformation (SI) from Chapter 4, noting that

$$(C - \sigma I)^{-1} = (L^{-1}AL^{-*} - \sigma I)^{-1} = L^*(A - \sigma B)^{-1}L,$$

---

<sup>1</sup>When  $B$  is ill-conditioned, incorporating pivoting into the Cholesky decomposition can improve the numerical stability of the process [472].

and using a symmetric indefinite factorization, in general,

$$A - \sigma B = R^* D R \quad (5.6)$$

with block-diagonal  $D$  to compute

$$y = (C - \sigma I)^{-1}x = L^*(R^{-1}(D^{-1}(R^{-*}(Lx))))$$

in the order indicated by the parentheses.

As in the standard case, this shift-and-invert variant most often converges in fewer steps, compensating for the cost of factorizing the shifted matrix  $A - \sigma B$  and applying the extra operations on triangular matrices.

It looks like a shift-and-invert algorithm needs two matrix factorizations, one of  $B$  (5.4) and one of  $A - \sigma B$  (5.6), but we may avoid factoring  $B$  if we use any of the algorithms developed specially for generalized eigenproblems (5.1), which are described later in this chapter.

Another way to avoid factorizing  $B$  is to apply any algorithm for non-Hermitian matrices, described in Chapter 7, to the non-Hermitian matrix

$$C = (A - \sigma B)^{-1}B.$$

The eigenvalues  $\lambda$  of the original pencil (5.1) can be computed as

$$\lambda = \sigma + \frac{1}{\theta},$$

where  $\theta$  are the eigenvalues of  $C$ . It has the same eigenvectors as the original pencil (5.1).

The main advantage of this second approach is that we may have a standard non-Hermitian code, such as the implicitly restarted Arnoldi method (see §7.6), easily available. The nonsymmetric matrix  $C$  will have a reasonably well-conditioned eigenproblem, provided that  $B$  is well-conditioned with respect to inversion, but there is still a risk that we get a solution that violates some of the properties of a GHEP. For example, we may get eigenvalues with small but nonzero imaginary parts.

### 5.3 Direct Methods

In this section, we briefly discuss methods for computing eigenvalues and eigenvectors of dense matrices. With the factorization (5.4) of  $B$ , the GHEP (5.1) is reduced to the standard Hermitian eigenproblem (5.5). Then one may use the direct methods discussed in §4.2.

Specifically, in LAPACK [12], the following driver routines are provided for solving the GHEP (5.1) with  $B$  positive definite:

- a simple driver `xSYGV` computes all the eigenvalues and (optionally) eigenvectors. The underlying algorithm is the QR algorithm; see §4.2.
- an expert driver `xSYGVX` computes all or a selected subset of the eigenvalues and (optionally) eigenvectors. If few enough eigenvalues or eigenvectors are desired, the expert driver is faster than the simple driver. This driver routine uses the QR algorithm or bisection method and inverse iteration, whichever is more efficient.

- a divide-and-conquer driver xSYGVD solves the same problem as the simple driver. It is much faster than the simple driver for large matrices, but uses more workspace. The name divide-and-conquer refers to the underlying divide-and-conquer algorithm; see §4.2.

Numerical analysis of the methods shows that if  $B$  is ill-conditioned with respect to inversion, i.e., the condition number  $\kappa_2(B) = \|B\|_2 \|B^{-1}\|_2$  is large, the methods may be numerically unstable and/or have large errors in computed eigenvalues and eigenvectors. As yet there is no implementation of any direct method directly applicable to  $A$  and  $B$  while persevering with the symmetry of  $A$  and  $B$ . An alternative approach would be to apply the QZ algorithm (see §8.2), but this will lose the symmetry.

## 5.4 Single- and Multiple-Vector Iterations

*M. Gu*

**Power Method.** The direct iteration methods for the HEP can be generalized to the GHEP (5.1).

First we assume that a Cholesky factorization of  $B = L L^*$  is easily obtainable. We then reformulate (5.1) as a standard Hermitian eigenvalue problem with the matrix  $C$  as described in the previous subsection (5.5), possibly after making  $B$  positive definite as noted in the introduction (5.2). The power method of §4.3 now takes the following form.

**ALGORITHM 5.1: Power Method for GHEP**

```
(1) compute $y := L v$ and $z := y/\|y\|_2$ for initial guess v
(2) for $k = 2, 3, \dots$
(3) compute $y := C z$ and $\theta := z^T y$
(4) if $\|y - \theta z\|_2 \leq \epsilon_M$ then
(5) compute $y := L^{-*} z$, $z := y/\|y\|_2$ and stop.
(6) else
(7) $z := y/\|y\|_2$.
(8) end if
(9) end for
```

In step (3), the computation  $y := C z$  can be executed into three steps as

$$y_1 = L^{-*} z, \quad y_2 = A y_1, \quad \text{and} \quad y = L^{-1} y_2.$$

Hence, no explicit knowledge of the matrix  $C$  is needed for the power method. The power method converges under conditions similar to those for the standard HEP.

**Inverse Iteration.** If we can factor  $A - \sigma B$  for a shift  $\sigma$ , we can generalize the inverse iteration to compute eigenvalues of the problem (5.1) near  $\sigma$  as shown in Algorithm 5.2.

**ALGORITHM 5.2: Inverse Iteration for GHEP**

- (1) start with initial guess  $y$
- (2) for  $k = 1, 2, \dots$
- (3) multiply  $w = By$
- (4)  $\eta = (w^*y)^{1/2}$ ,  $v = y/\eta$ ,  $w = w/\eta$
- (5) operate  $y = (A - \sigma B)^{-1}w$
- (6)  $\theta = w^*y$
- (7) if  $\|(1/\theta)y - v\|_2 \leq \epsilon_M$ , stop
- (8) end for
- (9) accept  $\lambda = \sigma + 1/\theta$  and  $x = y/\theta$

Here are some comments on this algorithm. In step (3) we multiply by  $B$ , while in step (5) we solve a system with the shifted matrix  $A - \sigma B$ . In a practical case, we perform an initial sparse Gaussian elimination and use its  $L$  and  $U$  factors while operating. Step (4) makes sure that the vector  $v$  is of unit  $B$  norm. The quantity  $\theta$  is a Rayleigh quotient,

$$\theta = w^*y = w^*(A - \sigma B)^{-1}w = v^*B(A - \sigma B)^{-1}Bv. \quad (5.7)$$

In step (7) we test a residual vector,

$$r = (1/\theta)y - v = (A - \sigma B)^{-1}(\tilde{\lambda}B - A)v,$$

where  $\tilde{\lambda} = \sigma + 1/\theta$  is the current approximate eigenvalue. The inverse iteration converges under conditions similar to those for the standard HEP.

**Rayleigh Quotient Iteration.** Similar to inverse iteration, the Rayleigh quotient iteration (RQI) method of §4.3 can also be generalized to solve the problem (5.1).

**ALGORITHM 5.3: RQI for GHEP**

- (1) start with vector  $v$  with  $v^*Bv = 1$ ,  $w = Bv$ , and value  $\rho_1$
- (2) for  $k = 1, 2, \dots$
- (3) operate  $y = (A - \rho_k B)^{-1}w$
- (4) if singular, stop
- (5)  $\theta = w^*y$
- (6) multiply  $w = By$
- (7)  $\eta = (w^*y)^{1/2}$ ,  $v = y/\eta$ ,  $w = w/\eta$
- (8)  $\rho_{k+1} = \rho_k + \theta/\eta^2$
- (9) if  $|\theta| \geq \epsilon_M^{-1/2}$ , stop
- (10) end for
- (11) accept  $\lambda = \rho_k$  for most recent  $k$  and  $x = v$

The only difference between Algorithms 5.2 and 5.3 is in step (8), where the shift is updated. This makes it necessary to perform a sparse factorization in step (3) in each iteration. The reward for this is a cubic rate of convergence.

## 5.5 Lanczos Methods

### A. Ruhe

There are several variants of the Lanczos algorithm for the GHEP (5.1). Theoretically they correspond to a reformulation of (5.1) as a standard problem  $Cy = \theta y$ , with a matrix  $C$  chosen as either  $C = B^{-1}A$ , which corresponds to a *direct* iteration, or  $C = BA^{-1}$ , which corresponds to *inverse* iteration. We will actually study the slightly more general formulation  $C = B(A - \sigma B)^{-1}$ , *shift-and-invert* iteration. This is the variant that is preferred in most practical cases because it gives fast convergence to eigenvalues close to the target value  $\sigma$ , provided that we can solve linear systems with the shifted matrix.

The cause for using shift-and-invert iterations is stronger in this generalized case (5.1) than in the standard (4.1), since also a direct iteration needs solution of linear systems in each step, now with  $B$  as a matrix. Even if  $B$  most often is better conditioned than  $A$ , e.g., when  $B$  stands for a mass matrix in a vibration problem, it is only when  $B$  has a much simpler structure, like being diagonal, that direct iterations need substantially less work in each step than shift-and-invert iterations.

In all the variants, a basis of the Krylov subspace,

$$\mathcal{K}^j(C, x) = \text{span}\{x, Cx, C^2x, \dots, C^{j-1}x\},$$

is built up one column at a time, alternately solving systems and doing matrix-vector multiplications.

**Algorithm.** In the direct iteration variant, we multiply with  $A$  and solve systems with  $B$ ; this corresponds to  $C = B^{-1}A$ . It computes a basis  $V_j$ , where the matrix pencil  $\{A, B\}$  is represented by a real symmetric tridiagonal matrix,

$$T_j = \begin{bmatrix} \alpha_1 & \beta_1 & & \\ \beta_1 & \alpha_2 & \ddots & \\ \ddots & \ddots & \ddots & \beta_{j-1} \\ & & \beta_{j-1} & \alpha_j \end{bmatrix}, \quad (5.8)$$

satisfying the basic recursion,

$$AV_j = BV_j T_j + re_j^*. \quad (5.9)$$

Compare this to the standard Hermitian case (4.10). Now the basis  $V_j$  is  $B$ -orthogonal,

$$V_j^* BV_j = I_j, \quad (5.10)$$

and the matrix  $T$  is congruent to a section of  $A$ ,

$$V_j^* AV_j = T_j. \quad (5.11)$$

We simplify the description of the  $B$ -orthogonalization by introducing an auxiliary basis,

$$W_j = BV_j, \quad (5.12)$$

which is  $B^{-1}$ -orthogonal,

$$W_j^* B^{-1} W_j = I_j, \quad (5.13)$$

and for which  $W^* V = I$ .

Precisely as in the standard case, compute an eigensolution of  $T_j$ ,

$$T_j s_i^{(j)} = s_i^{(j)} \theta_i^{(j)},$$

and get a Ritz value  $\theta_i^{(j)}$  and a Ritz vector,

$$x_i^{(j)} = V_j s_i^{(j)}, \quad (5.14)$$

approximating an eigenpair of the pencil (5.1). Its residual,

$$\begin{aligned} r_i^{(j)} &= Ax_i^{(j)} - Bx_i^{(j)}\theta_i^{(j)} = AV_j s_i^{(j)} - BV_j s_i^{(j)}\theta_i^{(j)} \\ &= (AV_j - BV_j T_j) s_i^{(j)} = re_j^* s_i^{(j)} = Bv_{j+1} \beta_j s_{j,i}^{(j)}, \end{aligned}$$

is  $B$ -orthogonal to the Krylov space spanned by  $V_j$ .

We may estimate the norm of the residual as we did in the standard Hermitian case, (4.13), but now this is better done using the  $B^{-1}$ -norm getting

$$\|r_i^{(j)}\|_{B^{-1}} = |\beta_j s_{j,i}^{(j)}|, \quad (5.15)$$

using the fact that  $(Bv_{j+1})^* B^{-1} (Bv_{j+1}) = v_{j+1}^* B v_{j+1}$ . It is natural to use the  $B^{-1}$ -norm when measuring convergence; see [353, Chap. 15].

As in the standard case we need to monitor the subdiagonal elements  $\beta_j$  of  $T$ , and the last elements  $s_{j,i}^{(j)}$  of its eigenvectors. As soon as this product is small, we may flag an eigenvalue as converged, without actually performing the matrix-vector multiplication (5.14). We save this operation until the step  $j$  when the estimate (5.15) indicates convergence.

We get the following algorithm.

**ALGORITHM 5.4: Lanczos Method for GHEP**

- (1) start with  $q = x$ , starting vector, compute  $r = Bq$  and  $\beta_0 = |q^* r|^{1/2}$
- (2) for  $j = 1, 2, \dots$ , until convergence,
- (3)      $w_j = r/\beta_{j-1}$ ,  $v_j = q/\beta_{j-1}$
- (4)     operate  $r = Av_j$
- (5)      $r = r - w_{j-1}\beta_{j-1}$
- (6)      $\alpha_j = v_j^* r$
- (7)      $r = r - w_j \alpha_j$
- (8)     reorthogonalize if necessary
- (9)     solve system  $Bq = r$  for  $q$
- (10)     $\beta_j = |q^* r|^{1/2}$
- (11)    compute approximate eigenvalues  $T_j = S\Theta^{(j)}S^*$
- (12)    test for convergence
- (13) end for
- (14) compute approximate eigenvectors  $X = V_j S$

Let us comment on this algorithm step by step:

- (1) If a good guess for the wanted eigenvector is available, use it as the starting  $q$ . In other cases choose a random direction, for instance, one consisting of normally distributed random numbers. Notice that  $\alpha_1 = q^* A q / (q^* B q)$  is the Rayleigh quotient of the starting vector and that  $\beta_1$  measures the  $B^{-1}$ -norm of its residual (5.15).
- (4) This is where the large matrix  $A$  comes in. Any routine that performs a matrix-vector multiplication can be used.
- (8) It is sufficient to reorthogonalize just one of the bases  $V$  or  $W$ , not both of them. The choices are the same as in the standard case:

1. *Full*: Now we want to make the  $v$  basis vectors  $B$ -orthogonal, computing

$$r = r - B(V_j(V_j^* r))$$

and repeat until the vector  $r$  is orthogonal to the basis  $V_j$ . We have to apply one matrix-vector multiplication by  $B$  for each reorthogonalization, and we have to use the classical variant of the Gram–Schmidt process.

We may avoid these extra multiplications with  $B$  if we save both bases  $V_j$  and  $W_j$  and subtract multiples of the columns of  $W_j$ ,

$$r = r - W_j(W_j^* r),$$

until orthogonality is obtained, almost always just once. Now we may use a modified Gram–Schmidt orthogonalization.

2. *Selective*: Reorthogonalize only when necessary, one has to monitor the orthogonality as described in §4.4.4 for the standard case. Note that now the symmetric matrix  $V_j^* W_j = V_j^* B V_j$  is involved and some of the vectors  $v_k$  have to be replaced by the corresponding vectors  $w_k$ .
3. *Local*: Used for huge matrices, when it is difficult to store the whole basis  $V_j$ . Advisable only when one or two extreme eigenvalues are sought. We make sure that  $w_{j+1}$  is orthogonal to  $v_{j-1}$  and  $v_j$  by subtracting  $r = r - w_{j-1}(v_{j-1}^* r); r = r - w_j(v_j^* r)$  once in this step.

- (9) Here we need to solve a system with the positive definite matrix  $B$ . This was not needed in the standard case (4.1).

- (11) For each step  $j$ , or at appropriate intervals, compute the eigenvalues  $\theta_i^{(j)}$  and eigenvectors  $s_i^{(j)}$  of the symmetric tridiagonal matrix  $T_j$  (5.8). Same procedure as for the standard case.

- (12) The algorithm is stopped when a sufficiently large basis  $V_j$  has been found, so that eigenvalues  $\theta_i^{(j)}$  of the tridiagonal matrix  $T_j$  (5.8) give good approximations to all the eigenvalues of the pencil (5.1) sought.

The estimate (5.15) for the residual may be too optimistic if the basis  $V_j$  is not fully  $B$ -orthogonal. Then the Ritz vector  $x_i^{(j)}$  (5.14) may have its norm smaller than 1, and we have to replace the estimate by,

$$\|r_i^{(j)}\| = |\beta_j s_{j,i}^{(j)}| / \|V_j s_i^{(j)}\|_B.$$

- (14) The eigenvectors of the original matrix pencil (5.1) are computed only when the test in step (5.4) has indicated that they have converged. Then the basis  $V_j$  is used in a matrix-vector multiplication to get the eigenvector (5.14),

$$x_i^{(j)} = V_j s_i^{(j)},$$

for each  $i$  that is flagged as converged.

**Lanczos Algorithm with SI.** In many practical contexts a shift-and-invert variant is natural. It corresponds to the choice

$$C = B(A - \sigma B)^{-1}. \quad (5.16)$$

It gives eigenvalues close to a chosen shift point  $\sigma$ , and we get convergence after a much smaller number of steps  $j$ . Even if the systems with the shifted matrix  $(A - \sigma B)$  are more cumbersome to solve than those with  $B$  needed in the direct variant, the smaller number of steps  $j$  needed will most often compensate for this.

The basic recursion (5.9) is replaced by

$$B(A - \sigma B)^{-1}V_j = V_j T_j + r e_j^*, \quad (5.17)$$

now with a  $B^{-1}$ -orthogonal (5.10) basis  $V_j$  and a  $B$  orthogonal auxiliary basis  $W_j$  for which  $V_j = BW_j$ . The two bases are biorthogonal,  $V_j^* W_j = I_j$ , compared to the nonsymmetric two-sided Lanczos as described in §7.8.

The matrix  $T$  is symmetric,

$$T_j = V_j^* (A - \sigma B)^{-1} V_j. \quad (5.18)$$

An eigensolution of  $T_j$ ,

$$T_j s_i^{(j)} = s_i^{(j)} \theta_i^{(j)},$$

is used to get an approximate eigenvalue

$$\lambda_i^{(j)} = \sigma + \frac{1}{\theta_i^{(j)}} \quad (5.19)$$

and an approximate eigenvector

$$x_i^{(j)} = W_j s_i^{(j)}. \quad (5.20)$$

Its residual is

$$\begin{aligned} r_i^{(j)} &= Ax_i^{(j)} - Bx_i^{(j)}\lambda_i^{(j)} = (A - \sigma B)x_i^{(j)} - \frac{1}{\theta_i^{(j)}}Bx_i^{(j)} \\ &= \frac{1}{\theta_i^{(j)}} \left( (A - \sigma B)W_j s_i^{(j)} \theta_i^{(j)} - V_j s_i^{(j)} \right) \\ &= -\frac{1}{\theta_i^{(j)}}(A - \sigma B)w_{j+1}\beta_j s_{j,i}^{(j)}, \end{aligned}$$

and its norm is small whenever the quantity

$$\beta_j s_{j,i}^{(j)} / \theta_i^{(j)} \quad (5.21)$$

is small. This approximation is not a standard but a harmonic Ritz value of the original eigenproblem (5.1); see the discussion in §3.2.

As in the standard case, we can monitor  $\beta_j s_{j,i}^{(j)} / \theta_i^{(j)}$  and flag the eigenvalue  $\lambda_i$  as converged when it gets small, without actually performing the matrix-vector multiplication (5.20). We save this operation until the step  $j$  when the estimate indicates convergence.

We get the following algorithm.

**ALGORITHM 5.5: Shift-and-Invert Lanczos Method for GHEP**

- (1) start with  $r = x$ , starting vector. Compute  $q = Br$  and  $\beta_0 = |q^* r|^{1/2}$
- (2) for  $j = 1, 2, \dots$ , until convergence,
- (3)      $w_j = r / \beta_{j-1}$ ,  $v_j = q / \beta_{j-1}$
- (4)     operate  $r = (A - \sigma B)^{-1} v_j$
- (5)      $r = r - w_{j-1} \beta_{j-1}$
- (6)      $\alpha_j = v_j^* r$
- (7)      $r = r - w_j \alpha_j$
- (8)     reorthogonalize if necessary
- (9)     multiply  $q = Br$
- (10)     $\beta_j = |q^* r|^{1/2}$
- (11)    compute approximate eigenvalues  $T_j = S \Theta^{(j)} S^*$
- (12)    test for convergence
- (13) end for
- (14) compute approximate eigenvectors  $X = W_j S$

We note that only a few steps differ from the previous direct iteration algorithm. In actual implementations one can use the same program for both computations. Let us comment on those steps that differ:

- (8) Now we expect convergence for a rather small number of vectors  $j$ , and it is advised to use full reorthogonalization, even if selective reorthogonalization gives results that are accurate enough. Apply the Gram–Schmidt orthogonalization process,

$$r = r - W_j(W_j^*(Br)),$$

until  $r$  is  $B$ -orthogonal to the basis  $W_j$ . We need one extra matrix-vector multiplication with  $B$  for each reorthogonalization, and we have to use the classical Gram–Schmidt orthogonalization process.

We may save and use the auxiliary basis  $V_j$  to save extra  $B$  operations,

$$r = r - W_j(V_j^* r).$$

- (14) Now the basis  $W_j$  is used to get the eigenvector (5.20),

$$x_i^{(j)} = W_j s_i^{(j)},$$

for each  $i$  that is flagged as converged.

When we run a shift-and-invert operator (5.16), we factor

$$LDL^* = P^T(A - \sigma B)P \quad (5.22)$$

for an appropriate sparsity preserving permutation  $P$  once in the beginning, using sparse Gaussian elimination.

During the actual iteration, we use the factors  $L$  and  $U$ , computing

$$r = P(L^{-*}(D^{-1}(L^{-1}(P^T v_j)))) \quad (5.23)$$

in step (4) of Algorithm 5.5.

If there are eigenvalues  $\lambda$  on both sides of the shift  $\sigma$ , the matrix  $A - \sigma B$  is indefinite, and we cannot use simple Cholesky decomposition, but have to make a symmetric indefinite factorization, e.g., MA47 of Duff and Reid [141]. See §10.3. This type of algorithm makes  $D$  block-diagonal with one by one and two by two blocks. It is easy to determine the inertia of such a matrix, and this gives information on exactly how many eigenvalues there are on each side of the shift  $\sigma$ . We may determine the number of eigenvalues in an interval by performing two factorizations with  $\sigma$  in either end point. This is used to make sure that all eigenvalues in the interval have been computed in the software based on [162] and [206].

We note that Algorithm 5.5 does not need any factorization of the matrix  $B$  and can be applied also when  $B$  is singular. This is not an uncommon case in practical situations, but special precautions have to be taken to prevent components of the null space of  $B$  from entering the computation; see [161] or [340].

**Convergence Properties.** The convergence is governed by the same orthogonal polynomial theory as in the standard case; see, e.g., [353].

This theory says that we get convergence to those eigenvalues that are represented in the starting vector and faster convergence to those in the ends of the spectrum. The better separated these are from the rest of the eigenvalues, the faster they will converge.

In practical cases, we are often interested just in the lowest eigenvalues, and then it is good that those are among the first to converge in the direct iteration Algorithm 5.4. On the other hand, the relative separation of the lowest eigenvalues is often poor—remember that the separation is relative to the whole spread of the spectrum, not the distance to the origin.

In these cases, and when we want eigenvalues in a specified range, say  $\alpha \leq \lambda \leq \beta$ , it is of great advantage to use the shift-and-invert Algorithm 5.5 for an appropriately chosen shift  $\sigma$ , for instance, in the interval  $\alpha \leq \sigma \leq \beta$ .

In the generalized case, shift-and-invert strategies are even better motivated than in the standard case, since we have to solve a system in any case, either in step 4 or in step 9. The shift-and-invert operator  $C$ , (5.16), most often has very much better separated eigenvalues, needing just  $j = 50$  instead of several hundreds to get 10 eigenvalues.

**Multiple Eigenvalues.** As stated in the previous paragraph, we would get convergence only to eigenvectors that were represented in the starting vector. This is important when the matrix pencil (5.1) has multiple eigenvalues. In that case, we will get only one vector from the corresponding multidimensional invariant subspace.

There are two different ways to get more than one linearly independent eigenvector to a multiple eigenvalue, precisely as in the standard case. The first is to restart and run a projected operator, where all the converged eigendirections have been projected away; this corresponds to orthogonalizing the vector  $r$  in step 8 of Algorithms 5.4 or 5.5 to the matrix  $B$  multiplied by all converged eigenvectors. This procedure is repeated as long as new vectors converge; see, e.g., [318].

We may also run a generalized variant of block or band Lanczos, starting with several, say  $p$  starting directions, forming a block  $V_1$ , letting  $A$  operate on all of  $V_j$  in step  $j$ , to compute a new  $B$ -orthogonal block  $V_{j+1}$ . The matrix  $T$  will be a block tridiagonal, or more properly band matrix. See a detailed description in [206].

**Software Availability.** There are a few software packages available for the Lanczos methods described in this section. The package, LANZ, written by Jones and Patrick [249], is applicable to a definite pencil (5.1) with  $B$  positive semidefinite and includes SI (5.16) and uses selective (partial)  $B$ -reorthogonalization. The LANSO package discussed in §4.4 can also be used for a definite pencil with SI and selective reorthogonalization of a  $B$  orthogonal basis. The code described in [206] is a block Lanczos algorithm with SI that uses inertia to make sure that all eigenvalues in a specified interval have been found. This code is currently not in the public domain. Implicitly restarted Lanczos method described in §4.5 is available in ARPACK [295]. ARPACK has driver routines for the generalized Hermitian eigenproblem (5.1).

For more information about this software, including how to access it, see the book's homepage ETHOME.

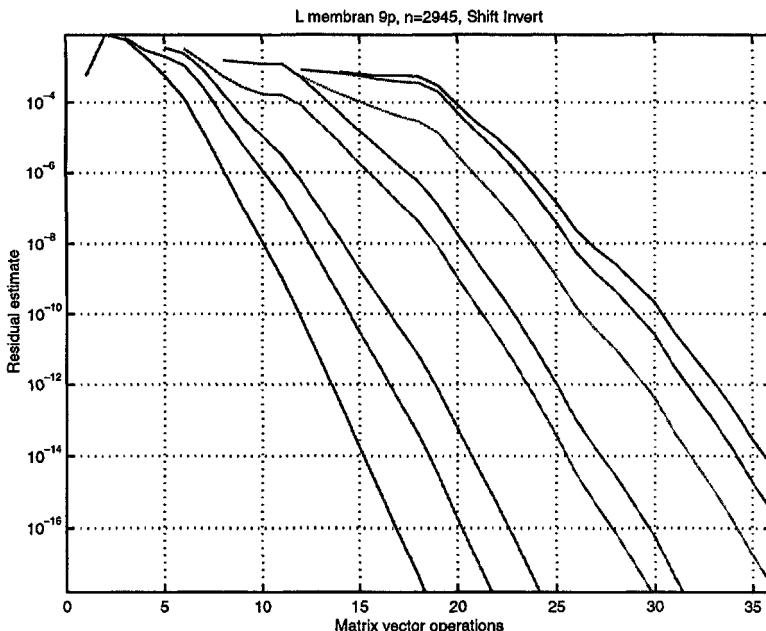


Figure 5.1: Residual estimates, Lanczos with shift-and-invert, L-membrane 9-point finite difference approximation.

**Numerical Example.** We report results on the generalized Hermitian shift-and-invert Lanczos algorithm for one of the examples in §4.4.6, the L-shaped membrane, but now with a 9-point finite difference approximation.

We use the same grid spacing  $h = 1/64$  and get two sparse symmetric positive definite matrices  $A$  and  $B$  of order  $n = 2945$ . They have a very regular sparse band structure, now with at most 9 nonzero elements filled in each row. We used the shift  $\sigma = 0$  to seek the 8 smallest eigenvalues. The residual estimates (5.21) at each step  $j$  are plotted in Figure 5.1.

When we compare the results on this generalized problem to those for the standard Hermitian eigenproblem of the same order in Figure 4.4 (p. 66) in §4.4.6, we have to bear in mind that this 9-point approximation is a more accurate one to the partial differential equation than the 5-point approximation used there.

## 5.6 Jacobi–Davidson Methods

*G. Sleijpen and H. van der Vorst*

We consider the application of the Jacobi–Davidson approach to the GHEP (5.1). We can, similarly as for the Lanczos method treated in the previous section (see also [444]), apply the Jacobi–Davidson method to (5.1), with a  $B$ -orthogonal basis  $v_1, v_2, \dots, v_m$  for the search subspace; that is,

$$V_m^* B V_m = I_m,$$

if we let  $V_m$  denote the matrix with columns  $v_j$ .

The Ritz–Galerkin condition for vectors  $u \equiv V_m s$  in this subspace leads to

$$(V_m^* A V_m)s - \theta(V_m^* B V_m)s = 0, \quad (5.24)$$

or, because of the  $B$ -orthogonality of  $V_m$ :

$$(V_m^* A V_m)s - \theta s = 0.$$

This leads to Ritz vectors  $u_j^{(m)} \equiv V_m s_j$  and Ritz values  $\theta_j^{(m)}$ . We will assume that these Ritz vectors are normalized with respect to the  $B$ -inner product.

The correction equation for the eigenvector component  $t_j^{(m)} \perp Bu_j^{(m)}$  for the generalized eigenproblem can be written as

$$\begin{aligned} & \left( I - Bu_j^{(m)} u_j^{(m)*} \right) (A - \theta_j^{(m)} B) \left( I - u_j^{(m)} u_j^{(m)*} B \right) t_j^{(m)} \\ &= -r_j^{(m)} \equiv -(A - \theta_j^{(m)} B) u_j^{(m)}. \end{aligned} \quad (5.25)$$

If linear systems with the matrix  $A - \theta_j^{(m)} B$  are efficiently solvable, then we can compute the exact solution  $t_j^{(m)}$ . In other cases we can, as is usual for the Jacobi–Davidson methods, compute approximate solutions to  $t_j^{(m)}$  with a Krylov solver applied to (5.25). Note that the operator

$$\left( I - Bu_j^{(m)} u_j^{(m)*} \right) (A - \theta_j^{(m)} B) \left( I - u_j^{(m)} u_j^{(m)*} B \right)$$

maps the space  $(Bu_j^{(m)})^\perp$  onto the space  $(u_j^{(m)})^\perp$ , so that preconditioning is always required if we use a Krylov solver in order to get a mapping between  $(Bu_j^{(m)})^\perp$  and  $(Bu_j^{(m)})^\perp$  (see item (31) of Algorithm 5.6).

As for the standard Hermitian case, the resulting scheme can be combined with restart and deflation. If we want to work with orthogonal operators in the deflation, then we have to work with  $B$ -orthogonal matrices that reduce the given generalized system to Schur form:

$$AQ_k = Z_k D_k,$$

**ALGORITHM 5.6: Jacobi–Davidson Method for  $k_{\max}$  Exterior Eigenvalues for GHEP**

```

(1) $t = v_0, k = 0, m = 0; Q = [], Z = []$
(2) while $k < k_{\max}$
(3) for $i = 1, \dots, m$
(4) $t = t - (v_i^B t^*) v_i$
(5) end for
(6) $m = m + 1; \tilde{t} = Bt, \text{norm} = \sqrt{t^* \tilde{t}},$
(7) $v_m = t/\text{norm}, v_m^A = Av_m, v_m^B = \tilde{t}/\text{norm}$
(8) for $i = 1, \dots, m$
(9) $M_{i,m} = v_i^* v_m^A$
(10) end for
(11) compute all eigenpairs (θ_i, s_i) of M ($\|s_i\|_2 = 1$)
 with sorted eigenpairs: $|\theta_i - \tau| \geq |\theta_{i-1} - \tau|$
(12) $u = Vs_1, p = V^B s_1, u^A = V^A s_1, r = u^A - \theta_1 p$
(13) while $\|r\|_2 / \|u\|_2 \leq \epsilon$
(14) $Q = [Q, u], Z = [Z, p], \tilde{\lambda}_{k+1} = \theta_1, k = k + 1$
(15) if $(k = k_{\max}), \tilde{X} = Q$, then stop
(16) $m = m - 1, M = 0$
(17) for $i = 1, \dots, m$
(18) $v_i = Vs_{i+1}, v_i^A = V^A s_{i+1}, v_i^B = V^B s_{i+1}$
(19) $M_{i,i} = \theta_{i+1}, s_i = e_i, \theta_i = \theta_{i+1}$
(20) end for
(21) $u = v_1, p = v_1^B, r = v_1^A - \theta_1 p$
(22) end while
(23) if $m > m_{\max}$ then
(24) $M = 0$
(25) for $i = 2, \dots, m_{\min}$
(26) $v_i = Vs_i, v_i^A = V^A s_i, v_i^B = V^B s_i, M_{i,i} = \theta_i$
(27) end for
(28) $v_1 = u, v_1^B = p, M_{1,1} = \theta_1, m = m_{\min}$
(29) end if
(30) $\theta = \theta_1, \tilde{Q} = [Q, u], \tilde{Z} = [Z, p]$
(31) solve $t \perp \tilde{Z}$ (approximately) from:
 $(I - \tilde{Z}\tilde{Q}^*)(A - \theta B)(I - \tilde{Q}\tilde{Z}^*)t = -r$
(32) end while

```

in which  $Z_k = BQ_k$  and  $Q_k$  is  $B$ -orthogonal. The matrix  $D_k$  is a diagonal matrix with the  $k$  computed eigenvalues on its diagonal; the columns of  $Q_k$  are eigenvectors of  $A$ . This leads to skew projections for the deflation with the first  $k$  eigenvectors:

$$(I - Z_k Q_k^*) (A - \lambda B) (I - Q_k Z_k^*) .$$

It is easy to verify that the deflated operator  $B$  is still symmetric positive definite with respect to the space  $(Bu_j^{(m)})^\perp$ . We can simply use the  $B$ -inner product in that space, since  $B$  and the deflated  $B$  coincide over that space.

If  $B$  is not well-conditioned, that means that if  $x^*By$  leads to a highly distorted inner product, then we suggest using the  $QZ$  approach with Jacobi–Davidson (see §8.4). The  $QZ$  approach does not exploit symmetry of the involved matrices.

Algorithm 5.6 represents a Jacobi–Davidson template with restart and deflation for exterior eigenvalues. A template for a left-preconditioned Krylov solver is given in Algorithm 5.7.

To apply this algorithm we need to specify a tolerance  $\epsilon$ , a target value  $\tau$ , and a number  $k_{\max}$  that specifies how many eigenpairs near  $\tau$  should be computed. The value of  $m_{\max}$  denotes the maximum dimension of the search subspace. If it is exceeded then a restart takes place with a subspace of specified dimension  $m_{\min}$ . We also need to give a starting vector  $v_0$ .

On completion the  $k_{\max}$  largest eigenvalues are delivered when  $\tau$  is chosen larger than  $\lambda_{\max}(A)$ ; the  $k_{\max}$  smallest eigenvalues are delivered if  $\tau$  is chosen smaller than  $\lambda_{\min}$ . The computed eigenpairs  $(\tilde{\lambda}_j, \tilde{x}_j)$ ,  $\|\tilde{x}_j\|_B = 1$ , satisfy  $\|A\tilde{x}_j - \tilde{\lambda}_j B\tilde{x}_j\|_2 \leq j\epsilon$ , where  $\tilde{x}_j$  denotes the  $j$ th column of  $\tilde{X}$ . The eigenvectors are  $B$ -orthogonal:  $\tilde{x}_i^* B \tilde{x}_j = 0$  for  $i \neq j$ .

Let us now discuss the different steps of Algorithm 5.6.

(1) Initialization phase.

(3)–(5) The new vector  $t$  is made  $B$  orthogonal with respect to the current search subspace by means of modified Gram–Schmidt. This can be replaced, for improved numerical stability, by a template as in Algorithm 4.14, in which all inner products and norms should be interpreted as  $B$ -inner products,  $B$ -norms, respectively.

If  $m = 0$  then this is an empty loop.

(7) We expand the  $n$  by  $m$  matrices  $V$ ,  $V^A \equiv AV$ , and  $V^B \equiv BV$ ,  $V$  denotes the matrix with the current basis vectors  $v_i$  for the search subspace as its columns; likewise  $V^A$  and  $V^B$ .

(8)–(10) The  $m$ th column of the symmetric matrix  $M \equiv V^*AV = V^*V^A$  (of order  $m$ ) is computed.

(11) The eigenproblem for the  $m$  by  $m$  matrix  $M$  can be solved with a suitable routine for Hermitian dense matrices from LAPACK. We have chosen to compute the standard Ritz values, which makes the algorithm suitable for computing  $k_{\max}$  exterior eigenvalues of  $A - \lambda B$  close to a specified  $\tau$ . If eigenvalues in the interior

part of the spectrum have to be computed, then the computation of *harmonic Petrov values* is advocated; see §8.4.

- (13) The stopping criterion is to accept an eigenvector approximation as soon as the norm of the residual (for the normalized vector approximation) is below  $\epsilon$ . This means that we accept inaccuracies in the order of  $\epsilon^2$  in the computed eigenvalues and inaccuracies (in angle) in the eigenvectors of  $O(\epsilon)$  (provided that the concerned eigenvalue is simple and well separated from the others and  $B$  is not ill-conditioned; see (5.33) and (5.34) in §5.7).

Detection of all wanted eigenvalues cannot be guaranteed; see item (13) of Algorithm 4.17 (p. 97).

- (17)–(20) After acceptance of a Ritz pair, we continue the search for a next pair, with the remaining Ritz vectors as a basis for the initial search space.

- (23)–(29) We restart as soon as the dimension of the search space for the current eigenvector exceeds  $m_{\max}$ . The process is restarted with the subspace spanned by the  $m_{\min}$  Ritz vectors corresponding to the Ritz values closest to the target value  $\tau$ . The construction of that subspace is done in (25)–(27).

- (31) We have collected the locked (computed) eigenvectors in  $Q$ , and the matrix  $\tilde{Q}$  is  $Q$  expanded with the current eigenvector approximation  $u$ . This is done in order to obtain a more compact formulation; the correction equation in step (31) of Algorithm 5.6 is equivalent to the one in equation (5.25). The new correction  $t$  has to be orthogonal to the columns of  $Z = BQ$  as well as to  $p = Bu$ .

Of course, the correction equation can be solved by any suitable process, for instance, a preconditioned Krylov subspace method that is designed to solve unsymmetric systems. However, because of the skew projections, we always need a preconditioner (which may be the identity operator if nothing else is available) that is deflated by the same skew projections so that we obtain a mapping between  $\tilde{Z}^\perp$  and itself. Because of the occurrence of  $\tilde{Q}$  and  $\tilde{Z}$ , one has to be careful with the usage of preconditioners for the matrix  $A - \theta B$ . The inclusion of preconditioners can be done as in Algorithm 5.7. Make sure that the starting vector  $t_0$  for an iterative solver satisfies the orthogonality constraints  $\tilde{Z}^* t_0 = 0$ . Note that significant savings per step can be made in Algorithm 5.7 if  $K$  is kept the same for a (few) Jacobi–Davidson iterations. In that case, columns of  $\tilde{Z}$  can be saved from previous steps. Also the matrix  $\mathcal{M}$  can be updated from previous steps, as well as its  $\mathcal{LU}$  decomposition.

It is not necessary to solve the correction equation very accurately. A strategy, often used for inexact Newton methods [113], works well here also: increase the accuracy with the Jacobi–Davidson iteration step; for instance, solve the correction equation with a residual reduction of  $2^{-\ell}$  in the  $\ell$ th Jacobi–Davidson iteration ( $\ell$  is reset to 0 when an eigenvector is detected).

For a full theoretical background of this method see [172]. For details on the deflation technique with eigenvectors see also §4.7.3.

**ALGORITHM 5.7: Approximate Solution of the Deflated Jacobi–Davidson GHEP Correction Equation**

“Solve” with left preconditioner  $\tilde{K} \equiv (I - \tilde{Z}\tilde{Q}^*)K(I - \tilde{Q}\tilde{Z}^*)$ ,  
 for  $\tilde{A} \equiv (I - \tilde{Z}\tilde{Q}^*)(A - \theta B)(I - \tilde{Q}\tilde{Z}^*)$ :

- (1) solve  $\tilde{Z}$  from  $K\tilde{Z} = \tilde{Z}$
- (2) compute  $\mathcal{M} = \tilde{Z}^*\tilde{Z}$
- (3) decompose  $\mathcal{M} = \mathcal{L}\mathcal{U}$
- (4) compute  $\tilde{r} \equiv \tilde{K}_j^{-1}r$  as:
  - (b') solve  $\tilde{r}$  from  $K\tilde{r} = r$
  - (c')  $\vec{\gamma} = \tilde{Z}^*\tilde{r}$
  - (7) solve  $\vec{\beta}$  from  $\mathcal{L}\vec{\beta} = \vec{\gamma}$
  - (8) solve  $\vec{\alpha}$  from  $\mathcal{U}\vec{\alpha} = \vec{\beta}$
  - (9) (d')  $\tilde{r} = \hat{r} - \tilde{Z}\vec{\alpha}$
- (10) Apply Krylov subspace method with start  $t_0 = 0$ , with operator  $\tilde{K}_j^{-1}\tilde{A}$ ,  
 and right-hand side  $-\tilde{r}$ ,  $z = \tilde{K}_j^{-1}\tilde{A}v$  for arbitrary  $v$  is computed as:
  - (a)  $y = (A - \theta B)v$
  - (12) solve  $\hat{y}$  from  $K\hat{y} = y$
  - (13)  $\vec{\gamma} = \tilde{Z}^*y$
  - (14) solve  $\vec{\beta}$  from  $\mathcal{L}\vec{\beta} = \vec{\gamma}$
  - (15) solve  $\vec{\alpha}$  from  $\mathcal{U}\vec{\alpha} = \vec{\beta}$
  - (16)  $z = \hat{y} - \tilde{Z}\vec{\alpha}$

## 5.7 Stability and Accuracy Assessments

*Z. Bai and R. Li*

The generalized eigenvalue problem for a Hermitian matrix pair  $\{A, B\}$  with one of  $A$  and  $B$ , or some linear combination of  $A$  and  $B$ , being positive definite takes a unique position among all generalized eigenvalue problems for matrix pairs because it resembles in many ways the standard Hermitian eigenvalue problem discussed in Chapter 4. Matrix pairs as such are called *Hermitian definite pairs*. We shall consider separately

1.  $B$  is definite and well-conditioned, meaning that  $\kappa(B) \equiv \|B\|_2\|B^{-1}\|_2$  is not too large.<sup>2</sup>
2. Some combination of  $A$  and  $B$  is definite and well-conditioned.

In this section, we only review some basic results that are readily applicable to assess how accurate computed eigenvalues and eigenvectors may be. We assume the availability of residual vectors which are usually available upon the exit of a successful computation. If not, they can be computed at marginal cost afterwards.

<sup>2</sup>Exactly how large is too large is an ambiguous notion and should generally be dealt with on a case-by-case basis. Usually one may regard any number over 1000 as too large for condition numbers.

For the treatment of error estimation of computed eigenvalues and eigenvectors of dense generalized Hermitian eigenproblems, see Chapter 4 of the *LAPACK Users' Guide* [12].

### 5.7.1 Positive Definite $B$

We reduce it to an equivalent standard HEP. It is done as follows. Choose a decomposition for  $B$ :

$$B = GG^*. \quad (5.26)$$

Then the generalized eigenvalue problem for  $A - \lambda B$  is equivalent to the standard HEP for  $G^{-1}AG^{-*}$ . Both share the same eigenvalues since

$$Ax = \lambda Bx \Leftrightarrow G^{-1}AG^{-*}(G^*x) = \lambda(G^*x),$$

which also says that if  $x$  is an eigenvector for the pair,  $G^*x$  is an eigenvector for the matrix  $G^{-1}AG^{-*}$ , and on the other hand if  $y$  is an eigenvector for  $G^{-1}AG^{-*}$ ,  $G^{-*}y$  is an eigenvector for the pair. Common choices for  $G$  are:

1.  $G = B^{1/2}$ , the unique positive definite square root of  $B$ . In this case,  $G^* = G$ . This choice is good enough for theoretical investigations.
2.  $G$  is the Cholesky factor; optionally with pivoting, i.e.,  $G$  is lower triangular with positive diagonal entries. This choice is preferred for numerical computations.
3. Analogously  $G$  is upper triangular with positive diagonal entries. It shares the same advantage of the second choice.

In what follows, sometimes it is more convenient to use the inner product  $(\cdot, \cdot)_M$  induced by a positive definite matrix  $M$ , the corresponding vector norm  $\|\cdot\|_M$ , and the two-vector angle function (more precisely, angle between the subspaces spanned by two vectors)  $\theta_M(\cdot, \cdot)$ . In our case,  $M = B$  or  $B^{-1}$ . They are defined as follows.

$$\begin{aligned} (x, y)_M &\equiv y^* M x, \\ \|x\|_M &\equiv \sqrt{(x, x)_M} \equiv \sqrt{x^* M x}, \\ \cos \theta_M(x, y) &\equiv \frac{|(x, y)_M|}{\|x\|_M \|y\|_M}. \end{aligned}$$

When  $M = I$ , all three reduce to the usual definitions. It is rather easy to see that

$$\|M^{-1}\|_2^{-1/2} \|x\|_2 \leq \|x\|_M \leq \|M\|_2^{1/2} \|x\|_2. \quad (5.27)$$

With some extra work, we can relate  $\theta_M$  to the usual angle function, e.g., for  $M = I$ , as follows.

$$(2\kappa(M))^{-1/2} \sin \theta(x, y) \leq \sin \theta_M(x, y) \leq (2\kappa(M))^{1/2} \sin \theta(x, y). \quad (5.28)$$

**Residual Vector.** Let  $\tilde{\lambda}$  be a computed eigenvalue, and let  $\tilde{x}$  be its corresponding computed eigenvector. For the simplicity of the presentation, we normalize the computed eigenvector so that  $\|\tilde{x}\|_2 = 1$ . The corresponding *residual vector* or *residual error* is defined by

$$r = A\tilde{x} - \tilde{\lambda}B\tilde{x}.$$

Ideally, we would like to have  $r = 0$ , but in practice  $r \approx 0$ . It is conceivable that a small residual error implies good accuracy in the computed  $\tilde{\lambda}$  and  $\tilde{x}$ . We are interested in knowing how accurate the computed  $\tilde{\lambda}$  and  $\tilde{x}$  are, given  $r$ .

**Transfer Residual Error to Backward Error.** It can be proved that there are Hermitian matrices  $E$ , e.g.,

$$E = -r\tilde{x}^* - \tilde{x}r^* + (\tilde{x}^*A\tilde{x} - \tilde{\lambda}\tilde{x}^*B\tilde{x})\tilde{x}\tilde{x}^*, \quad (5.29)$$

such that  $\tilde{\lambda}$  and  $\tilde{x}$  are an exact eigenvalue and its corresponding eigenvector of  $\{A + E, B\}$ . We are interested in such matrices  $E$  with smallest possible norms. It turns out the best possible  $E = E_2$  for the spectral norm  $\|\cdot\|_2$  and the best possible  $E = E_F$  for the Frobenius norm  $\|\cdot\|_F$  satisfy

$$\|E_2\|_2 = \|r\|_2, \quad \|E_F\|_F = \sqrt{2\|r\|_2^2 - (\tilde{x}^*A\tilde{x} - \tilde{\lambda}\tilde{x}^*B\tilde{x})^2}. \quad (5.30)$$

See [256, 431, 473]. In fact,  $E_F$  is given explicitly by (5.29). So if  $\|r\|_2$  is small, the computed  $\tilde{\lambda}$  and  $\tilde{x}$  are exact ones of *nearby* matrices. Error analysis of this kind is called *backward error analysis* and matrices  $E$  are *backward errors*.

We say an algorithm that delivers an approximate eigenpair  $(\tilde{\lambda}, \tilde{x})$  is  $\tau$ -*backward stable* for the pair with respect to the norm  $\|\cdot\|$  if it is an exact eigenpair for  $\{A + E, B\}$  with  $\|E\| \leq \tau$ . With these in mind, statements can be made about the backward stability of the algorithm which computes the eigenpair  $(\tilde{\lambda}, \tilde{x})$ . In convention, an algorithm is called *backward stable* if  $\tau = O(\epsilon_M \|(A, B)\|)$ , where  $\epsilon_M$  is the machine precision.

**Error Bounds for Computed Eigenvalues.** Rewrite  $r$  as

$$G^{-1}r = (G^{-1}AG^*)G^*\tilde{x} - \tilde{\lambda}(G^*\tilde{x}).$$

By (4.54) on p. 106, we have

$$|\tilde{\lambda} - \lambda| \leq \frac{\|G^{-1}r\|_2}{\|G^*\tilde{x}\|_2} = \frac{\|r\|_{B^{-1}}}{\|\tilde{x}\|_B} \leq \|B^{-1}\|_2\|r\|_2 \quad (5.31)$$

for some eigenvalue  $\lambda$  of the pair  $\{A, B\}$ . A good estimate to  $\|B^{-1}\|_2$  is needed to use this bound.

With more information, a much better bound can be obtained. Let us assume that  $(\tilde{\lambda}, \tilde{x})$  is an approximation of the eigenpair  $(\lambda, x)$  of the pair. The “best”  $\tilde{\lambda}$  corresponding to  $\tilde{x}$  is the Rayleigh quotient  $\tilde{\lambda} = \tilde{x}^*A\tilde{x}/\tilde{x}^*B\tilde{x}$ , so we assume that  $\tilde{\lambda}$  has this value. Suppose that  $\tilde{\lambda}$  is closer to  $\lambda$  than any other eigenvalues of the pair, and let  $\delta$  be the *gap* between  $\tilde{\lambda}$  and any other eigenvalue of the pair. Then

$$|\tilde{\lambda} - \lambda| \leq \frac{1}{\delta} \cdot \frac{\|r\|_{B^{-1}}^2}{\|\tilde{x}\|_B^2} \leq \|B^{-1}\|_2^2 \frac{\|r\|_2^2}{\delta}. \quad (5.32)$$

This improves (5.31) if the gap  $\delta$  is reasonably big. In practice we can always pick the better one. This bound also needs information on  $\delta$ , besides the residual error  $r$  and  $\|B^{-1}\|_2$ . Usually such information is available after a successful computation by, e.g., the shift-and-invert Lanczos algorithm, which usually delivers eigenvalues in the neighborhood of a shift and consequently yields good information on the  $\delta$ . This comment also applies to the bounds in (5.33) and (5.34) below.

**Error Bounds for Computed Eigenvectors.** Keep the assignments to  $\tilde{\lambda}$ ,  $\lambda$ , and  $\delta$  as above, and let  $x$  be the eigenvector of  $A$  corresponding to  $\lambda$ . The error bound for the computed eigenvector  $\tilde{x}$  and the true eigenvector  $x$  are

$$\sin \theta_B(x, \tilde{x}) \leq \frac{1}{\delta} \cdot \frac{\|r\|_{B^{-1}}}{\|\tilde{x}\|_B}. \quad (5.33)$$

This, (5.27), and (5.28) yield

$$\sin \theta(x, \tilde{x}) \leq \|B^{-1}\|_2 \frac{\sqrt{2\kappa(B)}}{\delta} \|r\|_2. \quad (5.34)$$

The factor  $\sqrt{2}$  can be removed by a more elaborate argument, but we shall omit it here.

**Remarks on Clustered Eigenvalues.** In the case when the eigenvalue  $\lambda$  has one or more other eigenvalues of  $A$  close by, in other words, when  $\lambda$  belongs to a cluster of eigenvalues, as guaranteed by (5.31) the computed  $\tilde{\lambda}$  is still accurate as long as  $\|B^{-1}\|_2 \|r\|_2$  is tiny, but the computed eigenvector  $\tilde{x}$  may be inaccurate because of the appearance of the gap  $\delta$  in the denominator of (5.33). It turns out that each individual eigenvector associated with the clustered eigenvalues is very sensitive to perturbations, but the eigenspace spanned by all the eigenvectors associated with the clustered eigenvalues is not. Thus for the clustered eigenvalues, we should instead compute the entire eigenspace. A theory along the lines given above can be established to show that the difference between the computed eigenspace and the eigenspace associated with the cluster is inversely proportional to the gap defined as the smallest difference between any eigenvalue in the cluster and any other eigenvalue not in the cluster. Because of the way it is defined, this gap is expected to be big.

### 5.7.2 Some Combination of $A$ and $B$ is Positive Definite

This is the general case for the definite matrix pair, and now  $B$  may be singular. To be able to handle infinite eigenvalues, it is standard practice [425] to introduce a homogeneous representation of an eigenvalue  $\lambda$  by a nonzero pair of numbers  $(\alpha, \beta)$ :

$$\lambda \equiv \alpha/\beta, \quad |\alpha|^2 + |\beta|^2 > 0.$$

When  $\beta = 0$ , such pairs represent eigenvalue  $\infty$ , and this occurs when  $B$  is singular. Such representations are clearly not unique since  $(\xi\alpha, \xi\beta)$  represents the same ratio for any  $\xi \neq 0$ , and consequently the same eigenvalue. So really a pair  $(\alpha, \beta)$  is a

representative from a class of pairs that give the same ratio. The difference of two eigenvalues is measured by the *chordal metric*: for  $\lambda = \alpha/\beta$  and  $\tilde{\lambda} = \tilde{\alpha}/\tilde{\beta}$ ,

$$\chi(\lambda, \tilde{\lambda}) \equiv \chi((\alpha, \beta), (\tilde{\alpha}, \tilde{\beta})) \equiv \frac{|\alpha\tilde{\beta} - \beta\tilde{\alpha}|}{\sqrt{|\alpha|^2 + |\beta|^2} \sqrt{|\tilde{\alpha}|^2 + |\tilde{\beta}|^2}} = \frac{|\lambda - \tilde{\lambda}|}{\sqrt{1 + |\lambda|^2} \sqrt{1 + |\tilde{\lambda}|^2}}. \quad (5.35)$$

An equivalent definition for a Hermitian matrix pair  $\{A, B\}$  being a definite pair is that the *Crawford number*

$$\gamma(A, B) \equiv \min_{\|x\|_2=1} \sqrt{(x^* Ax)^2 + (x^* Bx)^2} > 0.$$

It can be proved [425] that if  $\{A, B\}$  is a definite pair, then

- there is a  $\phi \in [0, 2\pi)$  such that  $B_\phi$  is *positive definite* and  $\gamma(A, B) = \lambda_{\min}(B_\phi)$ , the smallest eigenvalue of  $B_\phi$ , where

$$A_\phi = A \cos \phi - B \sin \phi, \quad B_\phi = A \sin \phi + B \cos \phi. \quad (5.36)$$

The reader is referred to [86, 87] for an algorithm that realizes such a  $B_\phi$ , which in turn yields the lower bound on  $\gamma(A, B)$  needed for estimating error bounds below.

- there is an  $n \times n$  nonsingular matrix  $X$  such that

$$X^* AX = \Lambda_A, \quad X^* BX = \Lambda_B, \quad (5.37)$$

where

$$\Lambda_A = \text{diag}(\alpha_1, \alpha_2, \dots, \alpha_n), \quad \Lambda_B = \text{diag}(\beta_1, \beta_2, \dots, \beta_n). \quad (5.38)$$

The decompositions (5.37) and (5.38) give a complete picture of the underlying eigenvalue problems. In fact, all eigenvalues are given by pairs  $(\alpha_i, \beta_i)$  with corresponding eigenvectors  $X e_i$ . If, in addition, in (5.37) and (5.38)  $|\alpha_i|^2 + |\beta_i|^2 = 1$  for all  $i$ , then [423]

$$\|X\|_2 \leq \frac{1}{\sqrt{\gamma(A, B)}}, \quad \|X^{-1}\|_2 \leq \frac{\|(A, B)\|_2}{\sqrt{\gamma(A, B)}}. \quad (5.39)$$

**Residual Vector.** Let  $(\tilde{\alpha}, \tilde{\beta})$  denote a computed eigenvalue, and let  $\tilde{x}$  be its corresponding computed eigenvector. For the simplicity, we normalize the computed eigenvector and eigenvalue so that

$$\|\tilde{x}\|_2 = 1, \quad |\tilde{\alpha}|^2 + |\tilde{\beta}|^2 = 1.$$

The corresponding *residual vector* or *residual error* is defined by

$$r = \tilde{\beta} A \tilde{x} - \tilde{\alpha} B \tilde{x}.$$

Ideally, we would like to have  $r = 0$ , but in practice  $\|r\|_2$  is small. It is conceivable that a small residual error implies good accuracy in the computed  $(\tilde{\alpha}, \tilde{\beta})$  and  $\tilde{x}$ . We are interested in knowing how accurate the computed  $(\tilde{\alpha}, \tilde{\beta})$  and  $\tilde{x}$  are, given  $r$ .

**Transfer Residual Error to Backward Error.** It can be proved that there are Hermitian matrices  $E$  and  $F$ , e.g.,

$$(E, F) = \left[ -r\tilde{x}^* - \tilde{x}r^* + (\tilde{\beta}\tilde{x}^* A\tilde{x} - \tilde{\alpha}\tilde{x}^* B\tilde{x}) \tilde{x}\tilde{x}^* \right] \cdot (\tilde{\beta}I, -\tilde{\alpha}I), \quad (5.40)$$

such that  $(\tilde{\alpha}, \tilde{\beta})$  and  $\tilde{x}$  are an exact eigenvalue and its corresponding eigenvector of  $\{A + E, B + F\}$ . We are interested in such matrices  $(E, F)$  with smallest possible norms. It turns out that the best possible  $(E, F) = (E_2, F_2)$  for the spectral norm  $\|\cdot\|_2$  and the best possible  $(E, F) = (E_F, F_F)$  for Frobenius norm  $\|\cdot\|_F$  satisfy

$$\|(E_2, F_2)\|_2 = \|r\|_2, \quad \|(E_F, F_F)\|_F = \sqrt{2\|r\|_2^2 - (\tilde{\beta}\tilde{x}^* A\tilde{x} - \tilde{\alpha}\tilde{x}^* B\tilde{x})^2}. \quad (5.41)$$

See [256, 431, 473].<sup>3</sup> In fact,  $(E_F, F_F)$  is given explicitly by (5.40). Therefore if  $\|r\|_2$  is small, the computed  $(\tilde{\alpha}, \tilde{\beta})$  and  $\tilde{x}$  are exact ones of *nearby* matrices. Error analysis of this kind is called *backward error analysis* and matrices  $(E, F)$  are *backward errors*.

We say an algorithm that delivers an approximate eigenpair  $((\tilde{\alpha}, \tilde{\beta}), \tilde{x})$  is  $\tau$ -*backward stable* for the pair with respect to the norm  $\|\cdot\|$  if it is an exact eigenpair for  $\{A + E, B + F\}$  with  $\|(E, F)\| \leq \tau$ . With these in mind, statements can be made about the backward stability of the algorithm which computes the eigenpair  $((\tilde{\alpha}, \tilde{\beta}), \tilde{x})$ . In convention, an algorithm is called *backward stable* if  $\tau = O(\epsilon_M\|(A, B)\|)$ .

**Error Bound for Computed Eigenvalues.** We have

$$\chi((\tilde{\alpha}, \tilde{\beta}), (\alpha, \beta)) \leq \frac{\|r\|_2}{\gamma(A, B)} \quad (5.42)$$

for some eigenvalue  $(\alpha, \beta)$  of  $\{A, B\}$ . To use this bound, we need an estimation to  $\gamma(A, B)$ . This can be done by first finding  $B_\phi$  as we mentioned above and then estimating its smallest eigenvalue.

**Error Bound for Computed Eigenvectors.** Keep the assignments to  $(\alpha, \beta)$ , let  $x$  be the eigenvector of  $\{A, B\}$  corresponding to  $(\alpha, \beta)$ , and let  $\eta$  be the smallest distance in chordal metric between  $(\tilde{\alpha}, \tilde{\beta})$  and all the other eigenvalues of the pair. Then we have

$$\sin \theta(x, \tilde{x}) \leq \frac{1}{\eta} \cdot \frac{\|r\|_2}{\gamma(A, B)}. \quad (5.43)$$

This bound also needs information on  $\eta$ , besides the residual error  $r$  and  $\gamma(A, B)$ . Usually such information is available after a successful computation by, e.g., the shift-and-invert Lanczos algorithm which usually delivers eigenvalues in the neighborhood of a shift and consequently yields good information on the  $\eta$ .

**Remarks on Clustered Eigenvalues.** As in the case for  $B$  being positive definite and well-conditioned, when the eigenvalue  $(\alpha, \beta)$  has one or more other eigenvalues of  $A, B$  close by, in other words, when  $(\alpha, \beta)$  belongs to a cluster of eigenvalues, as guaranteed by (5.42) the computed  $(\tilde{\alpha}, \tilde{\beta})$  is still accurate as long as  $\|r\|_2/\gamma(A, B)$  is tiny, but

<sup>3</sup>Kahan, Parlett, and Jiang [256, Theorem 1'] gave only optimal norms for  $\tilde{\beta}E - \tilde{\alpha}F \equiv Z$ . Take  $E = \tilde{\beta}Z$  and  $F = -\tilde{\alpha}Z$  to see that  $(E, F)$  has the same 2-norm and Frobenius norm as  $Z$ .

the computed eigenvector  $\tilde{x}$  may be inaccurate because of the appearance of the gap  $\eta$  in the denominator of (5.43). It turns out that each individual eigenvector associated with the clustered eigenvalues is very sensitive to perturbations, but the eigenspace spanned by all the eigenvectors associated with the clustered eigenvalues is not. Thus for the clustered eigenvalues, we should instead compute the entire eigenspace. It can be proved [299, 430] that the difference between the computed eigenspace and the eigenspace associated with the cluster is inversely proportional to the gap defined as the smallest difference in chordal metric between any eigenvalue in the cluster and any other eigenvalue not in the cluster. Because of the way it is defined, this gap is expected to be big.

*This page intentionally left blank*

# Chapter 6

# Singular Value Decomposition

## 6.1 Introduction

In this chapter we consider the singular value decomposition (SVD) of the  $m$  by  $n$  matrix  $A$ . We assume without loss of generality that  $m \geq n$ ; if  $m < n$  consider  $A^*$ . As described in §2.4, this decomposition may be written

$$A = U\Sigma V^*, \quad (6.1)$$

where  $U = [u_1, \dots, u_m]$  is an  $m$  by  $m$  unitary matrix,  $V = [v_1, \dots, v_n]$  is an  $n$  by  $n$  unitary matrix, and  $\Sigma$  is an  $m$  by  $n$  diagonal matrix with entries  $\Sigma_{ii} = \sigma_i$  for  $i = 1, \dots, n$ . The  $u_i$  are the *left singular vectors*, the  $v_i$  are the *right singular vectors*, and the  $\sigma_i$  are the *singular values*. The singular values are nonnegative and sorted so that  $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_n \geq 0$ . The number  $r$  of nonzero singular values is the rank of  $A$ . If  $A$  is real,  $U$  and  $V$  are real and orthogonal.

$A = U\Sigma V^*$  may also be written  $AV = U\Sigma$  or  $Av_i = u_i\sigma_i$  for  $i = 1, \dots, n$ .  $A = U\Sigma V^*$  may also be written  $A^*U = V\Sigma^*$  or  $A^*v_i = v_i\sigma_i$  for  $i = 1, \dots, n$  and  $A^*u_i = 0$  for  $i = n+1, \dots, m$ .

There are several “smaller” versions of the SVD that are often computed. Let  $U_t = [u_1, \dots, u_t]$  be an  $m$  by  $t$  matrix of the first  $t$  left singular vectors,  $V_t = [v_1, \dots, v_t]$  be an  $n$  by  $t$  matrix of the first  $t$  right singular vectors, and  $\Sigma_t = \text{diag}(\sigma_1, \dots, \sigma_t)$  be a  $t$  by  $t$  matrix of the first  $t$  singular values. Then we have the following SVD types.

*Thin SVD.*  $A = U_n\Sigma_nV_n^*$  is the *thin (or economy-sized) SVD* of  $A$ . The thin SVD is much smaller to store and faster to compute than the full SVD when  $n \ll m$ .

*Compact SVD.*  $A = U_r\Sigma_rV_r^*$  is a *compact SVD* of  $A$ . The compact SVD is much smaller to store and faster to compute than the thin SVD when  $r \ll n$ .

*Truncated SVD.*  $A_t = U_t\Sigma_tV_t^*$  is the *rank- $t$  truncated (or partial) SVD* of  $A$ , where  $t < r$ . Among all rank- $t$  matrices  $B$ ,  $B = A_t$  is the unique minimizer of  $\|A - B\|_F$  and also minimizes (perhaps not uniquely)  $\|A - B\|_2$ . The truncated SVD is much smaller to store and cheaper to compute than the compact SVD when  $t \ll r$  and is the most common form of the SVD computed in applications.

The thin SVD may also be written  $A = \sum_{i=1}^n \sigma_i u_i v_i^*$ . Each  $(\sigma_i, u_i, v_i)$  is called a *singular triplet*. The compact and truncated SVDs may be written similarly (the sum going from  $i = 1$  to  $r$ , or  $i = 1$  to  $t$ , respectively).

The SVD of  $A$  is closely related to the eigendecompositions of three related Hermitian matrices,  $AA^*$ ,  $A^*A$ , and  $H(A) \equiv [ \begin{smallmatrix} 0 & A \\ A^* & 0 \end{smallmatrix} ]$ , which were described in §2.4.7. Most iterative algorithms for the SVD amount to applying an algorithm from Chapter 4 to one of these Hermitian matrices, so we review and expand that material here. The choice of  $AA^*$ ,  $A^*A$ , or  $H(A)$  depends on which singular values and vectors one is interested in computing. The cost of some algorithms, like shift-and-invert (see below), may differ significantly for  $AA^*$ ,  $A^*A$ , and  $H(A)$ .

1. Consider  $A^*A$ , which is an  $n$  by  $n$  Hermitian matrix. Then the eigendecomposition of  $A^*A = V(\Sigma^*\Sigma)V^*$ , where  $A = U\Sigma V^*$  is the SVD of  $A$ . Note that  $\Sigma^*\Sigma = \text{diag}(\sigma_1^2, \dots, \sigma_n^2)$ . In other words, the eigenvectors of  $A^*A$  are the right singular vectors of  $A$ , and the eigenvalues of  $A^*A$  are the squares of the singular values of  $A$ .

Thus, if we find the eigenvalues  $\lambda_i$  and eigenvectors  $v_i$  of  $AA^*$ , then we can recover the compact SVD of  $A$  by taking  $\sigma_i = \lambda_i^{1/2}$  for  $i = 1, \dots, n$ , the right singular vectors as  $v_i$  for  $i = 1, \dots, n$ , and the first  $r$  left singular vectors by  $u_i = Av_i/\sigma_i$ . The left singular vectors  $u_{r+1}$  through  $u_m$  are not determined directly, but may be taken as any  $m - r$  orthonormal vectors also orthogonal to  $u_1$  through  $u_r$ . When  $\sigma_i \neq 0$  is very small, i.e.,  $0 < \sigma_i \ll \sigma_1$ , then  $u_i$  will not be determined very accurately.

2. Consider  $AA^*$ , which is an  $m$  by  $m$  Hermitian matrix. Then the eigendecomposition of  $AA^* = U(\Sigma\Sigma^*)U^*$ . Note that  $\Sigma\Sigma^* = \text{diag}(\sigma_1^2, \dots, \sigma_n^2, 0, \dots, 0)$ , with  $m - n$  zeros after  $\sigma_n^2$ . In other words, the eigenvectors of  $AA^*$  are the left singular vectors of  $A$ , and the eigenvalues of  $AA^*$  are the squares of the singular values of  $A$ , in addition to 0 with additional multiplicity  $m - n$ .

Thus, if we find the eigenvalues  $\lambda_i$  and eigenvectors  $u_i$  of  $AA^*$ , then we can recover the compact SVD of  $A$  by taking  $\sigma_i = \lambda_i^{1/2}$  for  $i = 1, \dots, n$ , the left singular vectors as  $u_i$  for  $i = 1, \dots, m$ , and the first  $r$  right singular vectors as  $v_i = A^*u_i/\sigma_i$ . The right singular vectors  $v_{r+1}$  through  $v_n$  are not determined directly, but may be taken as any  $n - r$  orthonormal vectors also orthogonal to  $v_1$  through  $v_r$ . When  $\sigma_i \neq 0$  is very small, i.e.,  $0 < \sigma_i \ll \sigma_1$ , then  $v_i$  will not be determined very accurately.

3. Consider  $H(A) = [ \begin{smallmatrix} 0^{m \times m} & A \\ A^* & 0^{n \times n} \end{smallmatrix} ]$ , which is an  $(m+n)$  by  $(m+n)$  Hermitian matrix. Write  $U = [U_n^{m \times n}, \tilde{U}_n^{m \times (m-n)}]$  and  $\Sigma = [ \begin{smallmatrix} \Sigma_n^{n \times n} \\ 0^{(m-n) \times n} \end{smallmatrix} ]$ . Then the eigendecomposition of  $H(A) = Q\Lambda Q^*$ , where  $\Lambda = \text{diag}(\Sigma_n, -\Sigma_n, O^{(m-n) \times (m-n)})$  and

$$Q = \left[ \begin{array}{ccc} \frac{1}{\sqrt{2}}U_n & -\frac{1}{\sqrt{2}}U_n & \tilde{U}_n \\ \frac{1}{\sqrt{2}}V & +\frac{1}{\sqrt{2}}V & O^{n \times (m-n)} \end{array} \right]. \quad (6.2)$$

In other words,  $\pm\sigma_i$  is an eigenvalue with unit eigenvector  $\frac{1}{\sqrt{2}}[\begin{smallmatrix} \pm u_i \\ v_i \end{smallmatrix}]$  for  $i = 1$  to  $n$ , and 0 is an eigenvalue with eigenvector  $[\begin{smallmatrix} u_i \\ 0^{n \times 1} \end{smallmatrix}]$  for  $i > n$ . Thus we can extract the singular values, left singular vectors, and right singular vectors of  $A$  directly from the eigenvalues and eigenvectors of  $H(A)$ . More precisely, we can directly extract the compact SVD from the eigenvalues and eigenvectors of  $H(A)$ . But if 0 is a singular value of  $A$ , then 0 will be (at least) a double eigenvalue of  $H(A)$ , so  $Q$

will not necessarily be of the form (6.2). (For example, if  $A = 0$  so that  $H(A) = 0$  too, then  $Q$  can be an arbitrary orthogonal matrix not necessarily of the form (6.2).) In this case, suppose the columns of  $\begin{bmatrix} \widehat{U}^{m \times z} \\ \widehat{V}^{n \times z} \end{bmatrix}$  form an orthonormal basis spanning the eigenspace for the zero eigenvalue of  $H(A)$ ; then the right singular vectors can be taken as any orthonormal vectors spanning  $\text{span}\widehat{V}$ , and the left singular vectors can be taken as any orthonormal vectors spanning  $\text{span}\widehat{U}$ .

The correspondence between the SVD of  $A$  and the eigendecomposition of  $H(A)$  shows that the discussion of perturbation theory for eigenvalues and eigenvectors of Hermitian matrices in §4.1 and §4.8 applies directly to the SVD, as we now describe.

Perturbing  $A$  to  $A + E$  can change the singular values by at most  $\|E\|_2$ :

$$|\sigma_i(A + E) - \sigma_i(A)| \leq \|E\|_2. \quad (6.3)$$

Now suppose  $(\widehat{\sigma}, \widehat{u}, \widehat{v})$  is an approximation of the singular triplet  $(\sigma_i, u_i, v_i)$ , where  $\widehat{u}$  and  $\widehat{v}$  are unit vectors. The “best”  $\widehat{\sigma}$  corresponding to  $\widehat{u}$  and  $\widehat{v}$  is the Rayleigh quotient  $\widehat{\sigma} = \text{Re}(\widehat{u}^* A \widehat{v})$ , so we assume that  $\widehat{\sigma}$  has this value. Suppose  $\widehat{\sigma}$  is closer to  $\sigma_i$  than any other  $\sigma_j$ , and let  $\delta$  be the *gap* between  $\widehat{\sigma}$  and any other singular value:  $\delta = \min_{j \neq i} |\widehat{\sigma} - \sigma_j|$ .

Define the *residual vector*  $r$  as

$$r = \frac{1}{\sqrt{2}} \begin{bmatrix} A\widehat{v} - \widehat{\sigma}\widehat{u} \\ A^*\widehat{u} - \widehat{\sigma}\widehat{v} \end{bmatrix}.$$

If  $r = 0$  then  $(\widehat{\sigma}, \widehat{u}, \widehat{v})$  is an exact singular triplet, and our error bounds will be small when  $\|r\|_2$  is small.

The difference between  $\widehat{\sigma}$  and  $\sigma_i$  is bounded by

$$|\widehat{\sigma} - \sigma_i| \leq \min \left\{ \|r\|_2, \frac{\|r\|_2^2}{\delta} \right\}.$$

Furthermore, the angles  $\angle(\widehat{v}, v_i)$  and  $\angle(\widehat{u}, u_i)$  between the approximate and true singular vectors are bounded by

$$\max\{\sin \angle(\widehat{v}, v_i), \sin \angle(\widehat{u}, u_i)\} \leq \frac{\sqrt{2}\|r\|_2}{\delta}.$$

In other words, the larger the gap  $\delta$  is between the approximate singular value  $\widehat{\sigma}$  and all the other  $\sigma_j$ , and the smaller the residual  $r$ , then the tighter one can bound the error in  $\widehat{\sigma}$ ,  $\widehat{u}$ , and  $\widehat{v}$ .

$\|r\|_2$  is easy to compute given  $A$ ,  $\widehat{u}$ , and  $\widehat{v}$ , but  $\delta$  requires more information about the singular values in order to approximate it. Typically one uses the computed singular values near  $\widehat{\sigma}$  to approximate  $\delta$ :  $\delta \approx \min\{\widehat{\sigma} - \widehat{\sigma}_-, \widehat{\sigma}_+ - \widehat{\sigma}\}$ , where  $\widehat{\sigma}_-$  is the next computed singular value smaller than  $\widehat{\sigma}$ , and  $\widehat{\sigma}_+$  is the next computed singular value larger than  $\widehat{\sigma}$ .

**Overview of Available Algorithms.** When  $A$  is not too large, so that it can be stored and manipulated as a dense  $m$  by  $n$  matrix, there are *transformation methods* that compute the SVD of  $A$ . These algorithms are available in LAPACK [12], ScaLAPACK [52], and MATLAB, and are discussed in §6.2.

The main emphasis in this book is on *iterative methods*, where  $A$  is stored and operated on sparsely. The iterative methods for the Hermitian eigenvalue problem discussed in Chapter 4 can be applied to one of the three Hermitian matrices  $AA^*$ ,  $A^*A$ , or  $H(A)$  discussed above. Rather than repeat this material, we will discuss the pros and cons of the different methods from Chapter 4, depending on which singular values and vectors one wants to compute, whether one chooses  $AA^*$ ,  $A^*A$ , or  $H(A)$ , the distribution of the singular values of  $A$ , the sparsity structure of  $A$ , whether one uses shift-and-invert, etc.

To this end, we note that for the task such as counting the number of singular values of  $A$  that are in a given interval  $[\alpha, \beta]$ , it does not require computing the singular values and so can be much cheaper. The key tool is the matrix inertia as presented in §4.1 (p. 49). It can be extended easily to apply to one of the Hermitian matrices  $AA^*$ ,  $A^*A$ , or  $H(A)$ .

## 6.2 Direct Methods

In this section, we briefly discuss methods for the computation of singular values and vectors of dense matrices. These methods are sometimes called transformation methods. Some or all of them are implemented in LAPACK, ScaLAPACK, and in the `svd` command in MATLAB.<sup>1</sup>

While it is possible to apply the transformation methods of §4.2 to one of the Hermitian matrices  $A^*A$ ,  $AA^*$ , or  $H(A)$ , the methods used in practice are specialized for the SVD and so are more efficient and accurate.

The most common transformation methods compute the thin SVD in three phases, shown below. (They can be easily modified to compute the full SVD, or just selected singular values and/or singular vectors, but we present just the thin SVD for simplicity.)

1. Find an  $n$  by  $n$  unitary matrix  $V_1$  and an  $m$  by  $n$  matrix  $U_1$  with orthonormal columns such that  $U_1^*AV_1 = B$  is an  $n$  by  $n$  *bidiagonal matrix*, i.e., is nonzero on the main diagonal and first superdiagonal only.
2. Compute the SVD of  $B$ :  $B = U_2\Sigma V_2^*$ .
3. Multiply  $U = U_1U_2$  and  $V = V_1V_2$  to get the thin SVD  $A = U\Sigma V^*$ .

Phase 1, reduction to bidiagonal form, is performed by LAPACK routine `xGEBRD` and ScaLAPACK routine `PxGEBRD` using a sequence of  $\min(m-1, n)$  unitary Householder reflections on the left and  $n-2$  unitary Householder reflections on the right, for a cost of  $4n^2(m - \frac{1}{2}n)$  floating point operations. If singular values only are desired, phase 2 costs just  $O(n^2)$ , which is much less than the cost of phase 1, and phase 3 is omitted. If all left and right singular vectors are desired, the cost depends strongly on which algorithm is selected, as described below.

The algorithms for phase 2 are analogous to those available for the symmetric tridiagonal eigenproblem as discussed in §4.2, with some differences outlined below.

Currently, the fastest available sequential or shared memory parallel dense SVD routine is LAPACK's `xGESDD`, but a faster one is on the way. The fastest available

---

<sup>1</sup>It has been announced that an LAPACK-based numerics library will be part of the next major release of MATLAB; see *The Newsletter for MATLAB, Simulink and Toolbox Users*, Winter 2000, available at <http://www.mathworks.com/company/newsletter/clevescorner/winter2000.cleve.shtml>

distributed memory parallel dense SVD routine is ScaLAPACK's PxGESVD, but again a better one is one the way.

*QR algorithm:* This algorithm finds all the singular values, and optionally all the left and/or right singular vectors of a bidiagonal matrix. It costs  $O(n^2)$  for singular values only and  $O(n^3)$  for the singular vectors as well. It can be implemented so that all singular values are computed with nearly all correct significant figures, even the tiniest ones (as long as underflow does not occur) [123]. QR is used to compute singular vectors in LAPACK's computational routine xBDSQR, which is used by driver routine xGESVD to compute the SVD of dense matrices but has been superseded by faster methods described below. xBDSQR is also used by ScaLAPACK driver routine PxGESVD.

*DQDS algorithm:* This algorithm finds just the singular values of a bidiagonal matrix in  $O(n^2)$  time, but as accurately and rather more efficiently than QR [168, 355, 360]. It is the sequential algorithm of choice for singular values, is implemented in LAPACK as xLASQ1, and is used by all LAPACK driver routines.

*Divide-and-conquer method:* This is currently the fastest method available in LAPACK to find all the singular values and vectors of a bidiagonal matrix larger than about 25 by 25 [208, 12]. It divides the matrix into two halves, computes the SVD of each half, and glues the solutions together by solving a special rational equation. For a large bidiagonal matrix this is repeated recursively until the parts are smaller than 25, when the QR algorithm is called.<sup>2</sup> It can be slightly less accurate than DQDS or QR, because the error in the tiniest singular values may be as large as machine epsilon times  $\sigma_1$ , but this is almost always good enough.

Divide-and-conquer is implemented by LAPACK computational routine xBDSDC, which is used by LAPACK driver routine xGESDD to compute the SVD of a dense matrix. xGESDD is currently the LAPACK algorithm of choice for the SVD of dense matrices.

*Bisection method and inverse iteration:* Bisection and inverse iteration can be used to find just those singular values and vectors of interest. Current algorithms work in time  $O(n)$  per singular triplet, unless the singular value is in a cluster of  $k$  nearby singular values, in which case the cost can rise to  $O(nk^2)$  if orthogonalization is used to try to guarantee orthogonal singular vectors. This could be as high as  $O(n^3)$  if many singular values are tightly clustered. Still, this can sometimes fail to guarantee orthogonality [129]. Neither LAPACK, ScaLAPACK, nor MATLAB currently contains an SVD routine based on bisection and inverse iteration.

Recent advances in [168, 128, 358, 356] promise an algorithm that costs  $O(n)$  work per singular triplet while guaranteeing orthogonality, but it is not yet complete. When done, both LAPACK and ScaLAPACK will provide this algorithm, which should be the fastest in all cases.

If  $A$  is banded, phase 1 can be performed more cheaply by LAPACK computational routine xGBBRD, but no LAPACK driver routines are available.

---

<sup>2</sup>This code makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Finally, we mention *Jacobi's method* [114, 198] for the SVD. This transformation method repeatedly multiplies  $A$  on the right by elementary orthogonal matrices (Jacobi rotations) until  $A$  converges to  $U\Sigma$ ; the product of the Jacobi rotations is  $V$ . Jacobi is slower than any of the above transformation methods (it can be made to run within about twice the time of QR [136]) but has the useful property that for certain  $A$  it can deliver the tiny singular values, and their singular vectors, much more accurately than any of the above methods provided that it is properly implemented [118, 115].

## 6.3 Iterative Algorithms

*J. Demmel*

Now we discuss the pros and cons of the methods for Hermitian eigenproblems from Chapter 4 as applied to  $A^*A$ ,  $AA^*$ , or  $H(A)$ . We note that the special structure of  $H(A)$  may be exploited to make Lanczos more efficient, as described in §6.3.3 below.

For simplicity, we let  $B$  denote any one of the above three Hermitian matrices in the discussion below. The basic tradeoffs from Table 4.1 remain true; the choice of method depends on the following:

- What operations can one afford to perform with the matrix and vectors? We measure cost in both time and space. Can one afford to solve  $(B - \sigma I)x = y$  for  $x$ , where  $\sigma$  is a shift near an eigenvalue of  $B$  (called SI in Table 4.1, for shift-and-invert)? Or can one only solve it approximately with a preconditioned iterative method (called Prec)? Or can one only afford to multiply vectors by  $A$  and  $A^*$  (called Dir)? Regarding storage and manipulation of vectors, the cheapest and least accurate option is local orthogonalization (called local); next is selective orthogonalization (called SO), and the most expensive and accurate option is full orthogonalization (called FO).
- Which singular values and vectors are desired? Does one want a few of the largest or smallest singular values of  $A$ , which are far away (isolated) from the others (called IE); or more of the largest or smallest singular values, even though they are tightly clustered together (called CE); or some interior singular values, in the middle of the spectrum (called M)?

What differs from Chapter 4 is how the above decisions depend on the choice of  $B$ .

### 6.3.1 What Operations Can One Afford to Perform?

The cheapest matrix operation is multiplication, by  $A^*A$ ,  $AA^*$ , or  $H(A)$ . Note that  $y = A^*Ax$  is computed as  $y = A^*(Ax)$ , i.e., as one multiplication by  $A$  and then one by  $A^*$ . There are two reasons not to form  $A^*A$  itself: First,  $A^*A$  can be much denser than  $A$  and so more expensive to multiply by. Indeed,  $A^*A$  can be dense even when  $A$  has only one nonzero row. Second, it can cost as much to form  $A^*A$  as  $n/4$  products  $A^*(Ax)$ , which is often more than enough to compute the desired singular triplets.

Similar comments apply to  $AA^*$ , except that since  $AA^*$  is  $m$ -by- $m$ , and  $m \geq n$ ,  $AA^*$  can be (arbitrarily) larger than  $A^*A$ . Also, storage and manipulation of  $n$  vectors can be (arbitrarily) cheaper than  $m$  vectors. So in general it is better to use  $A^*A$

instead of  $AA^*$ , and recover the left singular vectors  $u_i$  from the eigenvectors  $v_i$  of  $A^*A$  (right singular vectors of  $A$ ) by  $u_i = Av_i/\sigma_i$ . (But see the comments on shift-and-invert below for a case when  $AA^*$  is much better than  $A^*A$ .)

One multiplication by  $H(A)$  costs the same as one multiplication by either  $A^*A$  or  $AA^*$ .

Now consider shift-and-invert, which requires computing an  $LU$  or  $LDL^*$  factorization of either  $A^*A - \sigma I$ ,  $AA^* - \sigma I$ , or  $H(A) - \sigma I$ . Here  $\sigma$  is a *shift*, or approximate eigenvalue of the matrix from which it is being subtracted. The cost of these factorizations depends strongly on the sparsity structure of the matrix. Depending on the dimensions and sparsity structure of  $A$ , it may be *much* cheaper to factorize one of  $A^*A - \sigma I$ ,  $AA^* - \sigma I$ , or  $H(A) - \sigma I$  instead of the others. Here are some examples:

1. If  $A$  is nonzero in the first column and along the main diagonal, then  $A^*A$  and  $H(A)$  are nearly as sparse as  $A$  but  $AA^*$  is dense. So forming and factorizing  $A^*A - \sigma I$  costs time  $O(m + n)$  and space  $O(n)$ , but  $AA^* - \sigma I$  costs time  $O(m^3)$  and space  $O(m^2)$ , both vastly more. Forming and factorizing  $H(A) - \sigma I$  also costs time and space  $O(m + n)$ , but the constants are larger than for  $A^*A - \sigma I$ .
2. If  $m \approx n$ , and  $A$  is nonzero in the first row and along the main diagonal, then  $AA^*$  and  $H(A)$  are nearly as sparse as  $A$  but  $A^*A$  is dense. So  $AA^* - \sigma I$  costs  $O(n)$  time and space to form and factor, but  $A^*A - \sigma I$  costs time  $O(n^3)$  and space  $O(n^2)$ , both vastly more. Forming and factorizing  $H(A) - \sigma I$  also costs time and space  $O(n)$ , but the constants are larger than for  $AA^* - \sigma I$ .
3. If  $m \approx n$ , and  $A$  is nonzero in the first row, first column and along the main diagonal, then  $H(A)$  is nearly as sparse as  $A$  but both  $AA^*$  and  $A^*A$  are dense. So forming and factorizing  $H(A) - \sigma I$  costs time and space  $O(n)$ , but either  $AA^* - \sigma I$  or  $A^*A - \sigma I$  costs  $O(n^3)$  time and  $O(n^2)$ , both vastly more.

The above examples are chosen to represent extremes, where the behavior of  $A^*A$ ,  $AA^*$ , and  $H(A)$  are all as different as possible. It also assumes that the matrices being factorized are *well ordered*; i.e., the rows and columns are ordered to minimize fill and operations during factorization. For the above examples, symmetric minimum-degree ordering was used. In general, it is possible to compute the ordering and estimate the work and space required to factor  $AA^* - \sigma I$ ,  $A^*A - \sigma I$ , and  $H(A) - \sigma I$  by a *symbolic factorization* much more cheaply than actually performing the factorizations themselves. See §10.3 for details.

### 6.3.2 Which Singular Values and Vectors Are Desired?

Since the eigenvectors of  $A^*A$  are the right singular vectors, and the eigenvectors of  $AA^*$  are the left singular vectors, it might seem natural to use  $A^*A$  and  $AA^*$  to find right and left singular vectors, respectively. But as we pointed out earlier, left (or right) singular vectors for nonzero singular values can be recovered from right (or left) singular vectors by multiplying them by  $A$  (or  $A^*$ ), so either  $AA^*$  or  $A^*A$  can be used, whichever is cheaper.

The eigenvalues of  $A^*A$  and  $AA^*$  are the squares of the singular values of  $A$ . In contrast, the eigenvalues of  $H(A)$  are positive and negative singular values of  $A$  (as well as  $m - n$  additional zero eigenvalues). Since the speed with which all algorithms

converge depends on the distribution and location of eigenvalues, there are significant differences between  $A^*A$  and  $AA^*$  on the one hand, and  $H(A)$  on the other hand.

$AA^*$  and  $A^*A$  are appropriate for computing the largest singular values, since squaring keeps the largest singular values largest, and many of the algorithms of Chapter 4 converge most easily to the largest eigenvalues. Indeed, squaring increases any gaps between the largest singular values and the rest of the spectrum, accelerating convergence.

On the other hand, the smallest singular values are squared too. In the most extreme case, a singular value near the square root of machine precision  $\sqrt{\epsilon_M}$  turns into  $\epsilon_M$  after squaring, so its value may be entirely obscured by rounding errors. Also, small singular values that are clustered appear even more clustered after squaring. In other words, getting the smallest singular values can be challenging.

$H(A)$  does not square singular values, so the problems just mentioned do not arise. On the other hand, since the eigenvalues of  $H(A)$  are plus and minus the singular values of  $A$ , tiny singular values of  $A$  are near the middle of the spectrum of  $H(A)$ . These are the hardest eigenvalues to find and generally require shift-and-invert or Jacobi–Davidson to find. In summary, the most accurate (but most expensive) way to find the smallest singular values is to use shift-and-invert on  $H(A)$ .

### 6.3.3 Golub–Kahan–Lanczos Method

In this section we consider applying the Lanczos method of §4.4 to  $H(A)$ . The special structure of  $H(A)$  lets us choose a special starting vector that leads to a cheaper algorithm that produces two sequences of vectors, one intended to span the left singular vectors and one for the right singular vectors. In addition, it reduces  $A$  to bidiagonal form  $B$ ; i.e.,  $B$  is nonzero only on the main diagonal and first superdiagonal. We derive it from first principles and then show how it is related to Lanczos as described in §4.4.

**Golub–Kahan–Lanczos Bidiagonalization Procedure.** As discussed in §6.2, the first phase of a transformation method for the SVD is to compute unitary matrices  $U$  and  $V$  such that  $U^*AV$  is in bidiagonal form. In fact, the first column  $v_1$  of  $V$  can be chosen as an arbitrary unit vector, after which the other columns of  $U$  and  $V$  are generally determined uniquely. We write this as

$$U^*AV = B = \begin{bmatrix} \alpha_1 & \beta_1 & & & \\ & \alpha_2 & \beta_2 & & \\ & & \alpha_2 & \beta_3 & \\ & & & \ddots & \ddots \\ & & & & \alpha_{n-1} & \beta_{n-1} \\ & & & & & \alpha_n \end{bmatrix}. \quad (6.4)$$

All  $\alpha$ s and  $\beta$ s are real even if  $A$  was complex.

The constants  $\alpha_k$  and  $\beta_k$  are given by

$$\alpha_k = u_k^* A v_k \quad \text{and} \quad \beta_k = u_k^* A v_{k+1}.$$

From the bidiagonal form (6.4) we may derive a double recursion for the columns  $u_k$  and  $v_k$  of  $U$  and  $V$ . Multiplying by  $U$ , we have

$$A \begin{bmatrix} v_1 & v_2 & \dots & v_n \end{bmatrix} = \begin{bmatrix} u_1 & u_2 & \dots & u_n \end{bmatrix} \begin{bmatrix} \alpha_1 & \beta_1 & & \\ & \alpha_2 & \beta_2 & \\ & & \ddots & \ddots \\ & & & \alpha_{n-1} & \beta_{n-1} \\ & & & & \alpha_n \end{bmatrix}.$$

Equating the  $k$ th columns on both sides, we get

$$Av_k = \beta_{k-1}u_{k-1} + \alpha_k u_k$$

or

$$\alpha_k u_k = Av_k - \beta_{k-1}u_{k-1}, \quad \beta_0 = 0. \quad (6.5)$$

On the other hand, from the relation

$$A^* \begin{bmatrix} u_1 & u_2 & \dots & u_n \end{bmatrix} = \begin{bmatrix} v_1 & v_2 & \dots & v_n \end{bmatrix} \begin{bmatrix} \alpha_1 & \beta_1 & \alpha_2 & & \\ & \beta_2 & & \ddots & \\ & & \ddots & \alpha_{n-1} & \beta_{n-1} \\ & & & \beta_{n-1} & \alpha_n \end{bmatrix},$$

we get

$$A^* u_k = \alpha_k v_k + \beta_k v_{k+1}$$

or

$$\beta_k v_{k+1} = A^* u_k - \alpha_k v_k. \quad (6.6)$$

Since the columns of  $U$  and  $V$  are normalized, we must have

$$\alpha_k = \|Av_k - \beta_{k-1}u_{k-1}\|_2$$

and

$$\beta_k = \|A^* u_k - \alpha_k v_k\|_2.$$

We summarize the recursion in the following algorithm.

**ALGORITHM 6.1: Golub–Kahan–Lanczos Bidiagonalization Procedure**

- (1) choose  $v_1 = \text{unit 2-norm vector}$  and set  $\beta_0 = 0$
- (2) **for**  $k = 1, 2, \dots,$
- (3)    $u_k = Av_k - \beta_{k-1}u_{k-1}$
- (4)    $\alpha_k = \|u_k\|_2$
- (5)    $u_k = u_k / \alpha_k$
- (6)    $v_{k+1} = A^* u_k - \alpha_k v_k$
- (7)    $\beta_k = \|v_{k+1}\|_2$
- (8)    $v_{k+1} = v_{k+1} / \beta_k$
- (9) **end**

Collecting the computed quantities from the first  $k$  steps of the algorithm, we have the following important relations:

$$AV_k = U_k B_k, \quad (6.7)$$

$$A^* U_k = V_k B_k^* + \beta_k v_{k+1} e_k^*, \quad (6.8)$$

and

$$U_k^* U_k = I, \quad V_k^* V_k = I, \quad \text{and} \quad V_k^* v_{k+1} = 0, \quad (6.9)$$

where  $B_k$  is the  $k$  by  $k$  leading principal submatrix of  $B$  defined in (6.4).

**Relationship to Symmetric Lanczos.** From equations (6.7), (6.8), and (6.9), we have

$$A^* A V_k = V_k B_k^* B_k + \alpha_k \beta_k v_{k+1} e_k^*.$$

We note that  $B_k^* B_k$  is symmetric and tridiagonal, since  $B_k$  is bidiagonal. Comparing to equation (4.10), we see that Algorithm 6.1 computes the same information as Lanczos (Algorithm 4.6) applied to the Hermitian matrix  $A^* A$ . Conversely, if we apply Lanczos to  $A^* A$  to get a tridiagonal matrix  $T_k = B_k^* B_k$ , then  $B_k$  can be obtained by taking the upper triangular Cholesky factor of  $T_k$ .

Similarly, one gets

$$A A^* U_k = U_k B_k B_k^* + \beta_k (A v_{k+1}) e_k^*.$$

Again,  $B_k B_k^*$  is symmetric and tridiagonal. So again comparing to equation (4.10), we see that Algorithm 6.1 computes the same information as Lanczos (Algorithm 4.6) applied to the Hermitian matrix  $A A^*$ . Conversely, if we apply Lanczos to  $A A^*$  to get a tridiagonal matrix  $T_k = B_k B_k^*$ , then  $B_k$  can be obtained by taking the upper triangular Cholesky factor  $B'$  of  $J T_k J$ , where  $J$  is the identity matrix with its columns in reverse order (so that  $J T_k J$  is gotten by reversing the order of the rows and then the columns of  $T_k$ ), and then  $B_k = J B' J$ .

Finally, suppose one applies Lanczos (Algorithm 4.6) to  $H(A)$  with the special starting vector

$$z_1 = \begin{bmatrix} 0 \\ v_1 \end{bmatrix}$$

to generate the Lanczos vectors  $z_2, z_3, \dots$ . The first step of Algorithm 4.6 yields

$$r = H(A) z_1 = \begin{bmatrix} Av_1 \\ 0 \end{bmatrix},$$

$$\hat{\alpha}_1 = r^* z_1 = 0,$$

$$r = r - 0 * z_1,$$

$$\hat{\beta}_1 = \|r\|_2,$$

$$z_2 = r / \hat{\beta}_1$$

$$= \begin{bmatrix} u_1 \\ 0 \end{bmatrix}, \quad \text{where } u_1 \text{ is as computed in line (5) of Algorithm 6.1.}$$

The next step of Algorithm 4.6 yields

$$\begin{aligned}
 r &= H(A)z_2 = \begin{bmatrix} 0 \\ A^*u_1 \end{bmatrix}, \\
 \hat{\alpha}_2 &= r^*z_2 = 0, \\
 r &= r - 0 * z_2 - \hat{\beta}_1 z_1 = \begin{bmatrix} 0 \\ Au_1 \end{bmatrix} - \begin{bmatrix} 0 \\ \hat{\beta}_1 v_1 \end{bmatrix}, \\
 \hat{\beta}_2 &= \|r\|_2, \\
 z_3 &= r/\hat{\beta}_2 \\
 &= \begin{bmatrix} 0 \\ v_2 \end{bmatrix}, \quad \text{where } v_2 \text{ is as computed in line (8) of Algorithm 6.1.}
 \end{aligned}$$

Continuing in this fashion, we see that two steps of Algorithm 4.6 compute the same information as one step of Algorithm 6.1:

$$z_{2i-1} = \begin{bmatrix} 0 \\ v_i \end{bmatrix} \quad \text{and} \quad z_{2i} = \begin{bmatrix} u_i \\ 0 \end{bmatrix}.$$

However, Algorithm 4.6 does twice as many matrix-vectors multiplications by  $A$  and  $A^*$  as Algorithm 6.1 (half of them by zero vectors), so that Algorithm 6.1 will generally use half the time and space. Conversely, if Lanczos is applied to  $H(A)$  to obtain a tridiagonal matrix  $T_k$ , then  $T_k$  will have zeros on its diagonal, and the off-diagonal entries will be identical to the nonzero entries of  $B_k$  (notation from equation (6.4)):

$$T_{2k} = \begin{bmatrix} 0 & \alpha_1 & & & & & \\ \alpha_1 & 0 & \beta_1 & & & & \\ & \beta_1 & 0 & \alpha_2 & & & \\ & & \alpha_2 & 0 & \beta_2 & & \\ & & & \beta_2 & 0 & \ddots & \\ & & & & \ddots & \ddots & \alpha_k \\ & & & & & \alpha_k & 0 \end{bmatrix}.$$

Because of these equivalences, all the algorithmic variations and convergence properties of Lanczos from §4.4 apply to Algorithm 6.1.

### 6.3.4 Software Availability

As mentioned before, all of the software from Chapter 4 for the Hermitian eigenvalue problem can be applied to one of  $A^*A$ ,  $AA^*$ , or  $H(A)$ . However, some of this software and other software is specialized for the SVD, as we now describe.

Implicitly restarted Lanczos from §4.5 is available in ARPACK, as documented in the *ARPACK Users' Guide* [295]. Any of its variations may be applied to  $AA^*$ ,  $A^*A$ , or  $H(A)$ . Both ARPACK and a parallel version, PARPACK, are available electronically. ARPACK also has a driver for the SVD which applies its algorithm directly to  $A^*A$ , i.e., assuming only the ability to multiply by  $A$  and  $A^*$ . This is most suitable for finding a few of the largest singular values and their singular vectors.

PLANSO [462] is a parallel Lanczos algorithm with “thick restart” that compares favorably to PARPACK in speed and memory.

SVDPACK [49] is designed specifically for finding a few of the largest singular values of  $A$ . It includes many methods, of which the fastest is Lanczos (§4.4), which applies to  $A^*A$ . Other methods in SVDPACK are subspace iteration (§4.3.4), a variation of block-Lanczos that is a simpler version of the one in §4.6, and trace minimization. Any of these methods may be applied either to  $A^*A$  or  $H(A)$ . When applying block Lanczos to  $H(A)$ , a variation on the algorithm in §6.3.3 is used. Trace minimization is a case of the methods for nonlinear eigenproblems discussed in §9.4.

### 6.3.5 Numerical Example

See §4.4.6 (p. 63) where the second example is an SVD.

## 6.4 Related Problems

*J. Demmel*

As described in §2.4.7, there are several problems closely related to the SVD whose solutions we describe here. This list is not exhaustive, because the use of low rank approximations to matrices (for which the “optimal” one is supplied by the truncated SVD) is very widespread in applications. Depending on the application, it may be appropriate to use an SVD algorithm described here or another low rank approximation.

1. The *quotient or generalized SVD* of  $A$  and  $B$  is defined as follows. Suppose first that  $A$  is  $m$  by  $n$  and  $B$  is  $n$  by  $n$  and nonsingular. Let the SVD of  $AB^{-1} = U\Sigma V^*$ . Then we may also write two equivalent decompositions of  $A$  and  $B$  as  $A = U\Sigma_A X$  and  $B = V\Sigma_B X$ , where  $X$  is  $n$  by  $n$  and nonsingular,  $\Sigma_A = \text{diag}(\sigma_{A,1}, \dots, \sigma_{A,n})$  is  $m$  by  $n$ , and  $\Sigma_B = \text{diag}(\sigma_{B,1}, \dots, \sigma_{B,n})$  is  $n$  by  $n$ , with  $\sigma_{A,i}^2 + \sigma_{B,i}^2 = 1$ . Then  $\Sigma = \Sigma_A \Sigma_B^{-1} = \text{diag}(\sigma_1, \dots, \sigma_n)$ . The values  $\sigma_i = \sigma_{A,i}/\sigma_{B,i}$  are called the *generalized singular values* of  $A$  and  $B$ .

Software using transformation methods (i.e., suitable for dense matrices) is available in LAPACK driver routine `xGGSVD`, for the more general case of  $B$  being singular or  $p$  by  $n$ . It is also available as the MATLAB command `gsvd`. No ScaLAPACK software is available. The advantage of using these algorithms is accuracy only; that is, when  $B$  is ill-conditioned they can be much more accurate than simply forming  $AB^{-1}$  and computing its SVD. But they are also rather slower, so if  $B$  is well-conditioned, it is better to just form and compute the SVD of  $AB^{-1}$ .

When  $A$  and  $B$  are large and sparse and  $B$  is square, nonsingular, and can be factored, we recommend applying SVD algorithms discussed above that only require multiplication by the product  $AB^{-1}$  and its transpose.

If  $B$  is not square, we recommend finding the eigenvalues and eigenvectors of the generalized Hermitian eigenvalue problem  $A^*A - \lambda B^*B$ , using techniques from Chapter 5. This is because the eigendecomposition of  $A^*A - \lambda B^*B$  is  $X^*A^*AX = \Sigma_A^*\Sigma_A$  and  $X^*B^*BX = \Sigma_B^*\Sigma_B$ .

2. Yet *more generalized SVDs* can be defined for  $AB$  instead of  $AB^{-1}$  (the *product SVD*), or indeed arbitrary products of the form  $A_1^{\pm 1} A_2^{\pm 1} \cdots A_k^{\pm 1}$ . No specialized software for these cases is available in LAPACK, ScaLAPACK, or MATLAB. So

in the dense case, one would form these products explicitly (although algorithms have been proposed that avoid this [54, 3, 53]), and in the sparse case, one would multiply by them without forming them.

3. Finding  $x$  that minimizes  $\|Ax - b\|_2$  is the *linear least squares problem*. Suppose  $A$  is  $m$  by  $n$  with  $m > n$ ; then we say that the least squares problem is *overdetermined*. If  $A$  is nonsingular and  $A = U_n \Sigma_n V_n^*$  is the thin SVD of  $A$ , then the solution is  $x = V_n \Sigma_n^{-1} U_n^* b$ . It is generally cheaper to either solve the normal equations  $A^* A x = A^* b$  or use the QR decomposition  $A = QR$  to compute  $x = R^{-1} Q^* b$ , but when  $A$  is badly conditioned, i.e.,  $\sigma_n \ll \sigma_1$ , then the SVD can be much more accurate. In particular, when  $A$  is very ill-conditioned one typically uses the truncated SVD of  $A$  instead of the thin SVD:  $x = V_t \Sigma_t^{-1} U_t^* b$ .

It is not necessary to compute the thin or truncated SVD of  $A$  explicitly to solve the least squares problem. For dense problems, LAPACK driver routine `xGELSD` is the method of choice, using divide-and-conquer to solve the least squares problem quickly without forming an explicit thin SVD. In SCALAPACK, there is only a routine for computing a thin SVD, `PxGESVD`, which could then be used for the least squares problem. For sparse problems, we can use the approximate factorization  $A \approx U_k B_k V_k^*$  derived from equations (6.7) or (6.8) to get  $x = V_k B_k^{-1} U_k^{-1}$ .

*This page intentionally left blank*

# Chapter 7

# Non-Hermitian Eigenvalue Problems

## 7.1 Introduction

In this chapter we discuss the non-Hermitian eigenvalue problem (NHEP)

$$Ax = \lambda x, \quad (7.1)$$

where the square matrix  $A \neq A^*$ .  $x \neq 0$  is called a right eigenvector. A vector  $y \neq 0$  satisfying

$$y^* A = \lambda y^* \quad (7.2)$$

is called a left eigenvector of  $A$ .

For a presentation of relevant theory for NHEPs, as well as pointers to literature, we refer to §2.5. In particular, we mention that the perturbation theory for these problems is delicate. One should keep in mind that a small norm of the residual  $A\hat{x} - \hat{\lambda}\hat{x}$  for a computed eigenpair  $\hat{\lambda}, \hat{x}$  does not necessarily imply small errors in the computed values. For more information, see §2.5 and §7.13. Here we will give a brief introduction to those aspects that play a role in the selection of the algorithms. This will be followed by short characterizations of the different classes of methods, covered in this chapter.

In contrast to a Hermitian matrix, a non-Hermitian matrix does not have an orthogonal set of eigenvectors; in other words, a non-Hermitian matrix  $A$  can in general not be transformed by an orthogonal matrix  $Q$  to diagonal form  $D = Q^* A Q$ . Most non-Hermitian matrices can be transformed by a nonorthogonal  $X$  to diagonal form  $D = X^{-1} A X$ , but there exist matrices for which even this is not possible. Such matrices can be viewed as limit cases for which  $X$  converges to a singular operator, and these matrices do not have a complete set of eigenvectors; they are called *defective*. This reveals a source for numerical instability: if  $X$  is in some sense close to a singular operator, then the transformation may be expected to be sensitive to perturbations in  $A$ . Such perturbations are introduced by the process for the computation of  $X$  and  $D$ . Therefore, it is of great importance to work with orthogonal, or close to orthogonal, transformations as often as possible. It is well known that for any non-Hermitian matrix  $A$  there exists a unitary matrix  $U$  that transforms it to upper triangular form

$$T = U^* A U. \quad (7.3)$$

The matrix  $T$  is called the *Schur form* of  $A$ . The eigenvalues of  $A$  appear along the diagonal of  $T$ . Furthermore, the matrix  $U$  can be orthogonally transformed so that the eigenvalues of  $A$  occur in a prescribed order along the diagonal of  $R$ . If  $z$  is an eigenvector of  $R$  corresponding to the eigenvalue  $\lambda$ ,  $Tz = \lambda z$ , it follows that

$$A(Uz) = \lambda(Uz),$$

so that  $Uz$  is an eigenvector of  $A$  corresponding to the eigenvector  $\lambda$ .

Reduction to Schur form can be done but it has its price. Let  $n$  be the order of a non-Hermitian matrix  $A$ . For  $n > 3$ , there is in general no finite algorithm that reduces  $A$  to Schur form (if we exclude trivial cases, such as where  $A$  is already upper triangular). The usual approach for matrices of modest order, say  $n \leq 1000$ , is to reduce it first to upper Hessenberg form which can be done in a finite number of steps. Subsequently this upper Hessenberg matrix is reduced in an iterative manner (QR iterations) to Schur form. See §7.3 for more details and for pointers to software.

The main problem with this standard approach is that the computational costs are proportional to  $n^3$  (hence the limitation to matrices of order of a few thousands at most) and it requires storage proportional to  $n^2$  elements. In general it is not possible to exploit sparsity of the matrix  $A$  in order to reduce the storage requirements. For these reasons alternatives have been proposed. These alternatives are all completely iterative, that is, there is no finite direct reduction step. They are usually only suitable for the computation of a handful of interesting eigenpairs. We will now briefly summarize the iterative methods, to be presented in this chapter.

*Single- and multiple-vector iterations*, §7.4. The largest eigenvalue in absolute value, and the corresponding eigenvector, of a larger matrix can be computed iteratively by the *Power iteration*. This is only advisable if this largest eigenvalue is significantly larger, in a relative sense, than the absolute values of the other eigenvalues. In order to create a matrix with well-separated largest eigenvalue, one usually works formally with the operator  $(A - \sigma I)^{-1}$  for a user-defined  $\sigma$  close to the desired eigenvalue. This is referred to as *inverse iteration*. The method has been generalized to a block method for a set of eigenvalues that is well separated from the remaining part of the spectrum. These methods, members of the family of single- and multiple-vector iterations, work well when the desired eigenvalues are well separated from the unwanted ones. Even then, however, the methods can hardly compete with more modern subspace methods and the only niche for their usage seems to be when one cannot afford storage for more than two iteration vectors.

*Arnoldi methods*, §7.5, §7.6, and §7.7. Modern subspace iteration methods attempt to build a subspace that is rich in the eigenvectors that correspond to the wanted eigenvalues. In fact, they can be interpreted as methods that keep a number of vectors that are generated in the power method. With respect to this basis for a subspace of restricted dimension they usually reduce the matrix to a more manageable form. In the Arnoldi method one starts with an initial guess  $v$  and generates an orthonormal basis for a *Krylov subspace* of dimension  $m$ :

$$\mathcal{K}^m(A; v) \equiv \text{span}\{v, Av, A^2v, \dots, A^{m-1}v\}.$$

(As for the power method one may also use the shifted inverse operator  $(A - \sigma I)^{-1}$ .)

The orthonormalization process leads to an  $m$  by  $m$  upper Hessenberg matrix  $H_m$  (the approach is only practical for  $m \ll n$ ) and this matrix can be interpreted as a suitably reduced form of the matrix  $A$  restricted to the Krylov subspace. The eigenvalues of  $H_m$  are approximations for some of the eigenvalues of  $A$  and they can be computed by reducing  $H_m$  to Schur form by the same standard method as discussed above for small dense matrices. An introduction to the Arnoldi method is given in §7.5. Clever restart techniques have been designed to keep memory requirements and computational overhead as low as possible: these techniques are known as *implicitly restarted Arnoldi methods* (IRAMs) and are discussed in §7.6. The computational overhead in the Arnoldi methods is not very suitable for parallelism, in particular not for some distributed memory computers. In order to create a larger ratio between computation and memory references block variants have been suggested. These *block Arnoldi variants* are discussed in §7.7.

*Lanczos methods*, §7.8, §7.9, §7.10, and §7.11. The Arnoldi methods work with subspaces and one has to store the vectors that represent a basis for the current subspace. Moreover, adding a new vector to the basis involves the orthogonalization of this new vector with respect to all the vectors of the basis. For symmetric  $A$  there is a simple three-term recurrence that generates a basis for the subspace, and if one is interested in the eigenvalues only, then only the last three basis vectors have to be stored. A variant of this *Lanczos* process is also available, but then one has to give up the orthogonality of the basis vectors. Hence economy in memory and computational overhead comes at the risk of instability. The original unsymmetric Lanczos method, or simply *two-sided Lanczos*, suffers also from (near) breakdowns. Despite these drawbacks, the method has received much attention and variants have been suggested that have been proven to be efficient and useful for relevant problems. The method, and several enhancements to it, have been described in §7.8, along with appropriate software.

A *block version* of the Lanczos method, and an implementation of it called *ABLE*, are presented in §7.9. ABLE is an adaptively blocked method designed to cure breakdown, and it attempts to eliminate the causes of slow convergence (such as a loss of biorthogonality, and clusters of nearby eigenvalues). Deflation within Krylov subspaces is nicely incorporated in yet another variant, the *band Lanczos method*, presented in §7.10. This is specially tailored for the reduced-order modeling of multiple-input and multiple-output linear dynamical systems of high dimensions.

A variant of the two-sided Lanczos method for *complex symmetric* systems is discussed in §7.11. With respect to the unsymmetric case, this variant reduces CPU time and computer storage by a factor of 2.

*Jacobi-Davidson method*, §7.12. The Arnoldi and Lanczos methods are most effective when applied with shift-and-invert; that is, systems of the form  $(A - \sigma I)x = b$  have to be solved efficiently and accurately. If neither of this is practical, but if some reasonable approximation for  $x = (A - \sigma I)^{-1}b$  is cheaply available, then the *Jacobi-Davidson methods* can prove useful. These methods work with orthogonal transformations, just as the Arnoldi methods, but in the construction of an orthonormal basis for the involved subspace there is no attempt to create a special structure for the reduced matrix. Therefore, the computational overhead

is larger than for Arnoldi. All the possible gain has to come from an effective and cheap approximation for the shift-and-invert step.

In Table 7.1 we present a summary of the various classes of methods. The table is not intended to replace further reading; it serves only as an indication of where to start and it may help the reader to follow a certain path in discovering which method serves which need best. The actual performance of each method depends, often critically, on many parameters, and one method that is best for one class of problems may encounter convergence problems for some particular matrices.

Table 7.1: Summary of algorithms for NHEPs

| Algorithm         | Mode                  | FL- $\lambda$ | $\lambda$ -ext | $\lambda$ -int | Memory/overhead |
|-------------------|-----------------------|---------------|----------------|----------------|-----------------|
| Power             | $A$                   | Slow          | No             | No             | Low             |
|                   | $(A - \sigma I)^{-1}$ | Yes           | No             | No             | Low             |
| Subspace          | $A$                   | Yes           | Yes            | No             | Modest          |
|                   | $(A - \sigma I)^{-1}$ | Yes           | Yes            | Yes            | Modest          |
| Arnoldi<br>(IRAM) | $A$                   | Yes           | Yes            | No             | Modest          |
|                   | $(A - \sigma I)^{-1}$ | Yes           | Yes            | Yes            | Modest          |
| Lanczos           | $A$                   | Yes           | Yes            | No             | Low             |
|                   | $(A - \sigma I)^{-1}$ | Yes           | Yes            | Yes            | Low             |
| Block Lan.        | $A$                   | Yes           | Yes            | No             | Modest          |
|                   | $(A - \sigma I)^{-1}$ | Yes           | Yes            | Yes            | Modest          |
| Jac.-Dav.         | $A$                   | Yes           | Yes            | Slow           | Modest          |
|                   | $(A - \sigma I)^{-1}$ | Yes           | Yes            | Yes            | Modest          |
|                   | Precond               | Yes           | Yes            | Yes            | Modest          |

### Explanation of Table 7.1

FL- $\lambda$ : A few well-isolated largest (in absolute value) eigenvalues.

$\lambda$ -ext: (A group of) eigenvalues at the exterior of the spectrum.

$\lambda$ -int: (A group of) eigenvalues in the interior of the spectrum, near a specified target.

$A$ : Operations with (shifted)  $A$ .

$(A - \sigma I)^{-1}$ : With shift-and-invert operations; that is, one needs an efficient solution mechanism for determining  $x$  from  $(A - \sigma I)x = b$  ( $\sigma$  and  $b$  depend on the method).

Precond: Operations with a preconditioner for  $A - \sigma I$ ; that is, inexact solution of  $(A - \sigma I)x = b$  is permitted, which gives possibilities for more efficiency (the success depends of course on the quality of the preconditioner).

## 7.2 Balancing Matrices

*T. Chen and J. Demmel*

Numerical algorithms that compute the eigenvalues of a nonsymmetric matrix  $A$  typically make roundoff errors of size roughly  $\epsilon_M \|A\|$ , where  $\epsilon_M$  is the machine precision. Therefore, applying a simple and accurate similarity transform  $DAD^{-1}$  to reduce the

norm of the matrix  $A$ , or to reduce the condition numbers of some subset of  $A$ 's eigenvalues, can make the computed eigenvalues of  $A$  more accurate.

For example, consider the matrix

$$A = \begin{bmatrix} 1 & 0 & 10^{-4} \\ 1 & 1 & 10^{-2} \\ 10^4 & 10^2 & 1 \end{bmatrix}.$$

Choosing  $D = \text{diag}(100, 1, .01)$  gives

$$B = DAD^{-1} = \begin{bmatrix} 1 & 0 & 1 \\ 10^{-2} & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}.$$

Whereas  $\|A\|_F$  is approximately  $10^4$ ,  $\|B\|_F$  is approximately 2.6. Furthermore, the condition numbers of the eigenvalues of  $B$  are all approximately 1, whereas the condition numbers of the eigenvalues of  $A$  range in magnitude from  $10^1$  to  $10^3$ .

Osborne [346] first noted that the norm of a matrix  $A$  can often be reduced with a similarity transform of the form

$$B = DAD^{-1}, \quad (7.4)$$

where  $D$  is a diagonal matrix. Looking at irreducible matrices and ignoring their diagonal elements, Osborne also proved that the  $D$  which minimizes  $\|B\|_F$  also *balances*  $B$  in the 2-norm: a matrix is balanced in the  $\alpha$ -norm if for any  $i$ , the  $\alpha$ -norm of row  $i$  is the same as the  $\alpha$ -norm of column  $i$ . He introduced the first balancing algorithm, which repeatedly sweeps through the diagonal entries of  $D$ , updating  $D_{ii}$  to balance row and column  $i$ .

Although balancing in the 2-norm is equivalent to minimizing the Frobenius norm, balancing a matrix in an arbitrary norm may not have such a simple effect on a matrix norm. Other work discusses the mathematical properties of using diagonal scaling to balance matrices and to minimize matrix norms [81, 82]. Focusing on practice instead of theory, we present here two styles of algorithms for balancing sparse matrices. The algorithms are analyzed more thoroughly in [81] and [82]; software can be accessed through the book's homepage, ETHOME.

### 7.2.1 Direct Balancing

We will refer to the traditional dense balancing algorithm, found in LAPACK, as `XGEBAL`. It consists of two phases, fully described in [361]. The first phase permutes the rows and columns of the matrix  $A$  so that rows which isolate an eigenvalue are at the bottom of the matrix and columns which isolate an eigenvalue are at the left of the matrix. Because the algorithm assumes balancing is done to improve the accuracy of computed eigenvalues, there is no reason to balance these rows and columns from which eigenvalues can already be determined exactly. The second phase of the algorithm balances the matrix by iterating over the remaining rows and columns—each row/column pair is balanced in turn until significant progress can no longer be made.

The sparse implementation, `SPBALANCE`, takes as input a matrix in column compressed format; see §10.1. Instead of permuting a matrix to be as upper triangular as possible, `SPBALANCE` finds a permutation  $P$  that makes  $PAP^T$  as block upper triangular as possible, in the sense of maximizing the number of diagonal blocks. This

reduces the eigenvalue problem to the easier problem of finding the eigenvalues of each diagonal block. As the blocks found by SPBALANCE may be much smaller than those found by GEBAL, the eigenproblem may be easier. For example, on the 2000 by 2000 tolosa matrix [28], SPBALANCE found 1529 blocks of maximum size 90, whereas GEBAL found 1146 blocks of maximum size 854. In addition to the permutation and scaling arrays returned by LAPACK's xGEBAL, SPBALANCE also returns the number of diagonal blocks found and the indices of the diagonal blocks.

The balancing phase of SPBALANCE, done on the individual diagonal blocks found in the permuting phase, performs the same balancing algorithm used in GEBAL, but on a sparse matrix data structure and running in  $O(N_z)$  time, where  $N_z$  is the number of nonzeros. In our experiments SPBALANCE is much cheaper than the subsequent eigenvalue computation. SPBALANCE is the algorithm of choice when the matrix entries are given explicitly.

### 7.2.2 Krylov Balancing Algorithms

Direct balancing algorithms such as GEBAL and SPBALANCE calculate exact row and column norms, making them inappropriate for sparse matrices whose entries are not given explicitly. In [81] and [82] we describe three new balancing algorithms which use only Krylov information, i.e., matrix-vector and/or matrix-transpose-vector multiplies, to access the original matrix. The KRYLOVCUTOFF algorithm, Algorithm 7.1, performs best of all three Krylov algorithms on our test matrices.

Since  $A$  is not given explicitly, we assume functions for computing  $Az$  and  $A^Tz$  are available. (See [82] for a description of KRYLOVAz, a Krylov algorithm which uses only  $Az$  and not  $A^Tz$ .) In line (1)  $\|A\|_\infty$  can be approximated by multiplying  $A$  with a vector of random  $\pm 1$ s and taking the largest component of the absolute value of the result.

The number of iterations  $t$  and the value of *cutoff* are left to the user since the best stopping criteria and cutoff value can depend on the matrix  $A$ . Based on experimental evidence, we chose default values of 5 for  $t$  and  $10^{-8}$  for *cutoff*. This means that balancing costs at most 10 matrix-vector multiplications, and so is probably very cheap compared to subsequent eigenvalue calculations.

**ALGORITHM 7.1: Krylov Balancing Algorithm for NHEP (KRYLOVCUTOFF)**

```

(1) normA = ||A||∞
(2) for i = 1, n, D(i) = 1
(3) for j = 1, 2, ..., t
(4) z = a length n vector of random ±1s
(5) p = D(A(D-1z))
(6) r = D-1(AT(Dz))
(7) for i = 1, 2, ..., n
(8) if (|p(i)| > cutoff · normA) and (r(i) ≠ 0)
(9) D(i) = D(i) · √|r(i)/p(i)|
(10) end if
(11) end for
(12) end for

```

### 7.2.3 Accuracy of Eigenvalues Computed after Balancing

We conclude with a few examples showing the accuracy to which eigenvalues are computed, by dense and sparse eigensolvers, with and without balancing. This is the “bottom line,” showing how much balancing helps the overall eigenvalue computation process.

To study the accuracy of computed eigenvalues we need to know the true eigenvalues. For these experiments we computed the “true” eigenvalues in double precision, first using SPBALANCE to locate and balance the individual diagonal blocks of the matrix, then using a dense eigensolver to find the eigenvalues of each diagonal block separately. We then compared eigenvalues computed in single precision to these “true” eigenvalues.

Figures 7.1 and 7.2 plot the relative error in the eigenvalues calculated with balancing against the error in the eigenvalues calculated without balancing. Crosses below the dotted diagonal line represent eigenvalues calculated more accurately with balancing; the further a cross lies below the dotted line, the greater the difference in eigenvalue accuracy with and without balancing.

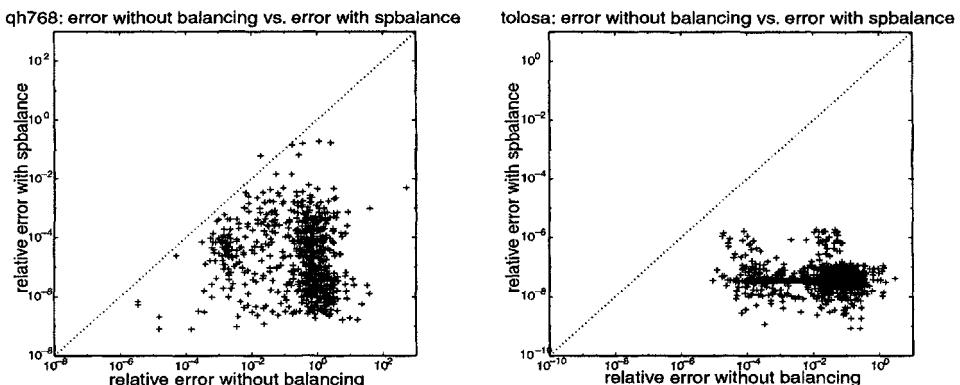


Figure 7.1: Relative accuracy of eigenvalues computed with and without direct balancing for the QH768 and TOLOSA matrices.

Figure 7.1 clearly shows that SPBALANCE is effective in improving the accuracy to which most eigenvalues of the QH768 and TOLOSA matrices, described in [28], are computed, often by many orders of magnitude.

Figure 7.2 shows that KRYLOVCUTOFF also improves the accuracy to which most eigenvalues of the QH768 and TOLOSA matrices are calculated, however the improvement is less than with the direct algorithm SPBALANCE.

In Figure 7.2, after Krylov-based balancing, the eigenvalues are computed using a dense eigensolver rather than a sparse solver; this is in order to isolate the effect of balancing. In practice, if a sparse matrix were given explicitly and one were willing to run an  $O(n^3)$  dense eigensolver, one would precondition the matrix using SPBALANCE and not a Krylov-based method. In a more practical situation one would use KRYLOVCUTOFF to balance matrices whose eigenvalues are computed using algorithms using only matrix-vector multiplications.

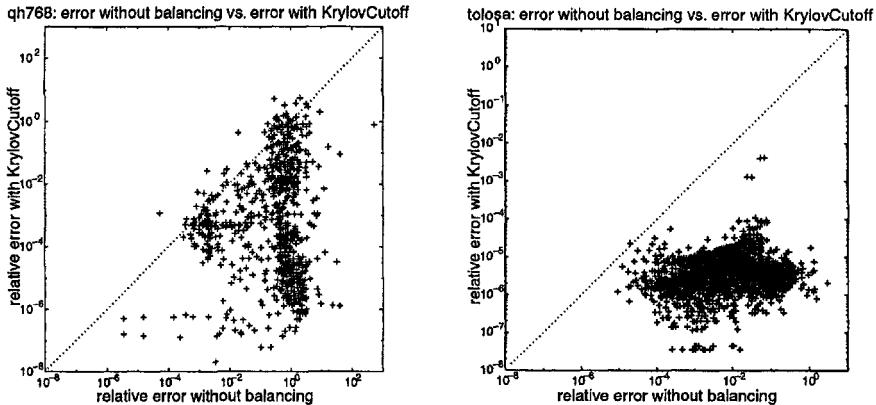


Figure 7.2: Relative accuracy of eigenvalues computed with and without Krylov balancing for the QH768 and TOLOSA matrices.

Figures 7.3 and 7.4 illustrate this more practical situation. They show the relative accuracy of the 10 largest and smallest eigenvalues when calculated without balancing, with SPBALANCE, with KRYLOVCUTOFF, and with two other Krylov balancing algorithms described in [82] (KRYLOVAZ, which uses only  $Az$ , and KRYLOVATZ, which uses  $Az$  and  $A^T z$  but no cutoff). The eigenvalues were computed in MATLAB using the `eigs` function, which uses the IRAM of §7.6.

Figure 7.3 shows little variation in the relative errors for the smallest eigenvalues of the QH768 matrix. However, for the largest eigenvalues KRYLOVCUTOFF does almost as well as SPBALANCE and significantly better than the results without balancing. On the TOLOSA matrix the smallest eigenvalues are computed most accurately after balancing by KRYLOVCUTOFF, even when compared to direct balancing. On the

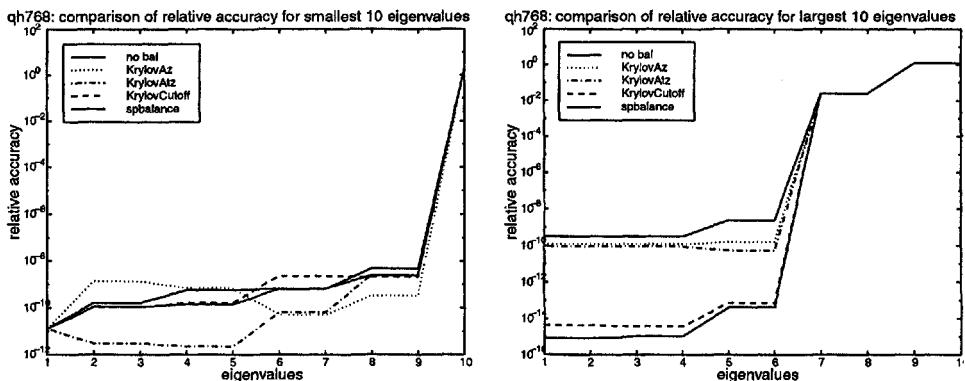


Figure 7.3: Comparison of the relative accuracy of the largest and smallest (in magnitude) eigenvalues of the QH768 matrix, with different Krylov-based balancing algorithms, using the default settings of five iterations and a cutoff value of  $10^{-8}$ .

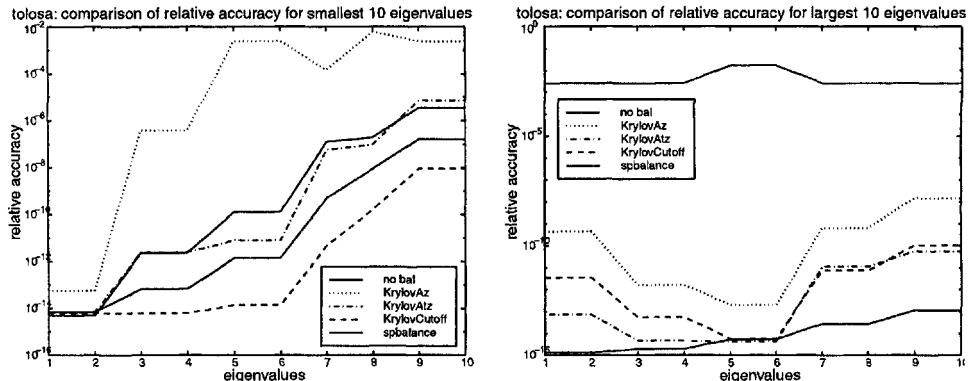


Figure 7.4: Comparison of the relative accuracy of the largest and smallest (in magnitude) eigenvalues of the TOLOSA matrix, with different Krylov-based balancing algorithms, using the default settings of five iterations and a cutoff value of  $10^{-8}$ .

largest eigenvalues SPBALANCE is most effective, though all three Krylov balancing algorithms do significantly better than no balancing at all.

### 7.3 Direct Methods

The primary direct method<sup>1</sup> used in practice to solve the NHEP (7.1) (and (7.2)) is the QR algorithm. It first computes the Schur canonical form (or simply called Schur decomposition) of a non-Hermitian matrix  $A$ :

$$A = UTU^*, \quad (7.5)$$

where  $U$  is a unitary matrix,  $U^*U = I$ , and  $T$  is an upper tridiagonal matrix. The eigenvalues  $\lambda$  of  $A$  are the diagonal entries of  $T$  (see also §2.5). The eigenvectors  $x$  of  $A$  are  $Us$ , where  $s$  are the eigenvectors of  $T$ , which can be obtained by solving triangular systems. When  $A$  is real, the QR algorithm computes the real Schur decomposition to avoid complex numbers for saving in floating point operations and memory.

The QR algorithm is originated from the simple QR iteration:

```

Let $A_0 = A$,
For $k = 1, 2, \dots$,
 QR-factorize $A_{k-1} = Q_k R_k$
 Compute $A_k = R_k Q_k$
```

Under certain conditions,  $A_k$  converges to Schur form  $T$ . However, the convergence of the QR iteration is extremely slow for practical usage. To make QR iteration a fast, effective method for computing Schur decomposition, a number of crucial improvements

<sup>1</sup>Note that a direct method must still iterate, since finding eigenvalues is mathematically equivalent to finding zeros of polynomials, for which no noniterative methods can exist. We call a method *direct* if experience shows that it (nearly) never fails to converge in a fixed number of iterations.

have been developed, including Hessenberg reduction, implicitly shifting, deflation, and matrix balancing. We refer the interested reader to [198, 114] and references therein for the details of the related theory and implementations.

The QR algorithm costs  $O(n^3)$  floating point operations and  $O(n^2)$  memory for a general  $n \times n$  matrix. A crude estimate for the costs for a real  $n \times n$  matrix is  $25n^3$  floating point operations if both  $U$  and  $T$  are computed. If only eigenvalues are desired, then about  $10n^3$  floating point operations are necessary.

The QR algorithm is backward stable; i.e., for the computed unitary matrix  $\widehat{U}$  (within machine precision) and the computed upper triangular matrix  $\widehat{T}$ , we have

$$A + E = \widehat{U}\widehat{T}\widehat{U}^*,$$

where  $\|E\| \leq p(n)\epsilon\|A\|$ , where  $p(n)$  is a modestly growing polynomial function of  $n$ .

The subroutine that implements the QR algorithm is available in almost every linear algebra-related software package. It is used as the `eig` command in MATLAB.<sup>2</sup> In LAPACK [12], the following driver routines are available for performing a variant of tasks for computing Schur decompositions, eigenvalues, eigenvectors, and estimates for the accuracy of computed results:

|                     |                                                       |
|---------------------|-------------------------------------------------------|
| <code>xGEES</code>  | compute Schur decomposition with eigenvalue ordering, |
| <code>xGEESX</code> | <code>xGEES</code> plus condition estimates,          |
| <code>xGEEV</code>  | eigenvalues and eigenvectors,                         |
| <code>xGEEVX</code> | <code>xGEEV</code> plus condition estimates,          |

where `x` is `S` or `D` for real single or double precision data types, or `C` or `Z` for complex single or double precision data types.

In ScaLAPACK [52], computational routines are provided for solving the parallel Hessenberg reduction and the parallel QR iteration with implicit shifting. They are `PxGEHRD` and `PxLAHQR`, respectively.

## 7.4 Single- and Multiple-Vector Iterations

*M. Gu*

The single- and multiple-vector iteration methods for the Hermitian eigenproblem in §4.3 can also be used to solve the NHEP, in many cases with similar convergence properties. Although these methods are in general not as competitive as the other methods to be covered in later sections, they are a good choice for those who only want to find a few extreme eigenvalues with a very simple method. The Rayleigh quotient iteration (RQI) will have an ultimately quadratic rate of convergence, which is slower than the cubic rate for the Hermitian case. When converging to a defective eigenvalue it has a slow linear rate  $(m - 1)/m$ , where  $m$  is the multiplicity of the eigenvalue.

---

<sup>2</sup>It has been announced that an LAPACK-based numerics library will be part of the next major release of MATLAB; see *The Newsletter for MATLAB, Simulink and Toolbox Users, Winter 2000*, available at <http://www.mathworks.com/company/newsletter/clevescorner/winter2000.cleve.shtml>

### 7.4.1 Power Method

The power method, described in Algorithm 4.1, can be used to solve the NHEP without any apparent change.

Under conditions similar to those in the Hermitian case, the power method for the non-Hermitian matrix  $A$  converges to  $\lambda_{\max}(A)$ , the largest eigenvalue in magnitude. The convergence rate depends on the ratio  $|\lambda_2/\lambda_{\max}|$ , where  $\lambda_2$  is the second largest eigenvalue of  $A$  in magnitude. For detailed discussions of the power method, see Wilkinson [457], Golub and Van Loan [198], and Demmel [114].

### 7.4.2 Inverse Iteration

Inverse iteration, described in Algorithm 4.2, can also be used to solve the NHEP without any apparent change. As in the Hermitian case, assume that  $\lambda_j$  and  $q_j$  are an eigenvalue and eigenvector pair of  $A$  so that  $|\lambda_j - \sigma|^{-1}$  is the largest eigenvalue of  $(A - \sigma I)^{-1}$  in magnitude. The inverse power method converges if the starting vector  $v$  is not perpendicular to  $q_j$ . The convergence rate is  $|(\lambda_j - \sigma)/(\lambda_k - \sigma)|$ , where  $\lambda_k$  is an eigenvalue of  $A$  such that  $|\lambda_k - \sigma|^{-1}$  is the second largest eigenvalue of  $(A - \sigma I)^{-1}$  in magnitude.

In general, inverse iteration tends to have much more rapid convergence than the power method if  $\sigma$  is chosen to be very close to a desired eigenvalue. However, inverse iteration does require a factorization of the matrix  $A - \sigma I$ , making it less attractive when this factorization is expensive.

### 7.4.3 Subspace Iteration

The subspace iteration methods, also called simultaneous iteration methods, for HEPs can also be generalized for the NHEP. Algorithm 7.2 is a modification of the subspace iteration method of §4.3.4 for the HEP. It includes deflation (locking) for computing the *new* dominant eigenvalues. Since eigenvalues for NHEPs can be very ill-conditioned, we impose a more stringent convergence criterion.

We now describe some implementation details.

- (1) The initial starting matrix  $V$  should be constructed to be dominant in eigenvector directions of interest to accelerate convergence. In case no such information is known *a priori*, a random matrix is as good a choice as any other.
- (3) The iteration parameter *iter* is introduced to reduce the costly orthonormalization computation as much as possible. However, *iter* must not be too large lest one lose numerical accuracy in the computation of the matrix  $\widehat{Y}$ , leading to inaccurate computation of some of the eigenvalues. A typical value of *iter* is 3 to 5.
- (6)–(13) The convergence criterion in the above algorithm checks for convergence only for groups of eigenvalues that have nearly the same modulus. The diagonal blocks in step (6) are ordered from top to bottom, with block 1 at the top of  $\Lambda$ . In step (12), convergence testing is stopped as soon as the first block of eigenvalues in  $\Lambda$  fails to converge.
- (16) The iteration parameter *iter* should be chosen to minimize orthonormalization cost while maintaining a reasonable amount of numerical accuracy.

**ALGORITHM 7.2: Subspace Iteration with Projection and Deflation for NHEP**

```

(1) QR-factorize $V R := Z$ for the starting vectors Z
(2) while $j \leq nev$ do
(3) compute $\widehat{Y} := [v_1, v_2, \dots, v_{j-1}, A^{iter} V]$.
(4) orthonormalize the columns of \widehat{Y} (starting at j) into V .
(5) $\Theta := V^* A V$ and compute the Schur decomposition
 form $\Theta = Q \Lambda Q^*$ with nearly equimodular eigenvalues
 grouped in diagonal blocks in Λ .
(6) for each diagonal block Λ_k , $k = 1, 2, \dots$ in Λ do
(7) let Q_k be the corresponding columns in Q
(8) if $\|A(V Q_k) - (V Q_k) \Lambda_k\| \leq \epsilon$ then
(9) append $V Q_k$ to $[v_1, v_2, \dots, v_{j-1}]$
(10) set $j = j + \text{number of columns in } V Q_k$
(11) else
(12) break
(13) end if
(14) end for
(15) choose a new iteration parameter $iter$
(16) end while

```

If eigenvalues near a shift  $\sigma$  are desired, and a factorization of  $A - \sigma I$  can be easily obtained, then one can apply the above algorithm to  $(A - \sigma I)^{-1}$ . The eigenvalues near  $\sigma$  will converge faster.

One can also use polynomial acceleration to speed up the computation by replacing the power  $A^{iter}$  by a polynomial  $T_{iter}[(A - \sigma I)/\rho]$  in which  $T_{iter}$  is the Chebyshev polynomial of the first kind of degree  $m$ , and  $\rho$  is an estimate of the spectral radius of  $A$ .

Much of the material in this section is based on Bai and Stewart [37] and Saad [387]. For further discussion on subspace iteration, the reader is referred to Wilkinson [457] and Stewart [422].

#### 7.4.4 Software Availability

There are several pieces of software available for subspace iteration described in this section. EB12 is a routine by Duff and Scott for subspace iteration [143]. It is part of the Harwell Subroutine Library. Given a real unsymmetric matrix, this routine computes the  $r$  eigenvalues that have the largest positive real part, the largest (in modulus) negative real part, or the largest modulus, with the option of returning the associated eigenvectors. It can also be used to compute other eigenvalues.

SRRIT is a package by Bai and Stewart for calculating a few eigenvalues with largest modulus for a real nonsymmetric matrix [37]. It also computes an approximate orthonormal basis for the associated invariant subspace. SRRIT is nonproprietary. The template in §7.4.3 is closely modeled on SRRIT.

LOPSI is a package by Stewart and Jennings that uses subspace iteration combined with an oblique projection to compute a few eigenvalues of largest modulus and the corresponding eigenvectors of a large sparse matrix [427]. However, LOPSI is based on eigendecompositions instead of the Schur form, and hence one may expect less accuracy in LOPSI than in SRRIT, especially for ill-conditioned eigenvalue problems.

For more information about this software including how to access it, see the book's homepage ETHOME.

## 7.5 Arnoldi Method

*Y. Saad*

The Arnoldi method was first introduced as a *direct* algorithm for reducing a general matrix into upper Hessenberg form [19]. It was later discovered that this algorithm leads to a good *iterative* technique for approximating eigenvalues of large sparse matrices.

The algorithm works for non-Hermitian matrices. It is most useful for cases when the matrix  $A$  is large but matrix-vector products are relatively inexpensive to perform. This is the situation, for example, when  $A$  is large and sparse. We begin with a presentation of the basic algorithm and then describe a number of variations.

### 7.5.1 Basic Algorithm

The Arnoldi method is an orthogonal projection method onto a Krylov subspace. It starts with the Arnoldi procedure as described in Algorithm 7.3. The procedure can be essentially viewed as a modified Gram–Schmidt process for building an orthogonal basis of the Krylov subspace  $\mathcal{K}^m(A, v)$ .

**ALGORITHM 7.3: Arnoldi Procedure**

- (1)       $v_1 = v/\|v\|_2$  for the starting vector  $v$
- (2)      **for**  $j = 1, 2, \dots, m$  **do**
- (3)         $w := Av_j$
- (4)        **for**  $i = 1, 2, \dots, j$  **do**
- (5)           $h_{ij} = w^* v_i$
- (6)           $w := w - h_{ij} v_i$
- (7)        **end for**
- (8)           $h_{j+1,j} = \|w\|_2$
- (9)          **if**  $h_{j+1,j} = 0$ , **stop**
- (10)          $v_{j+1} = w/h_{j+1,j}$
- (11)        **end for**

The above procedure will stop if the vector  $w$  computed in line (8) vanishes. The vectors  $v_1, v_2, \dots, v_m$  form an orthonormal system by construction and are called *Arnoldi vectors*. An easy induction argument shows that this system is a basis of the Krylov subspace  $\mathcal{K}^m(A, v)$ .

Next we consider a fundamental relation between quantities generated by the algorithm. The following equality is readily derived:

$$Av_j = \sum_{i=1}^{j+1} h_{ij} v_i, \quad j = 1, 2, \dots, m. \quad (7.6)$$

If we denote by  $V_m$  the  $n \times m$  matrix with column vectors  $v_1, \dots, v_m$ , and by  $H_m$  the  $m \times m$  Hessenberg matrix whose nonzero entries  $h_{ij}$  are defined by the algorithm, then the following relations hold:

$$AV_m = V_m H_m + h_{m+1,m} v_{m+1} e_m^*, \quad (7.7)$$

$$V_m^* AV_m = H_m. \quad (7.8)$$

Relation (7.8) follows from (7.7) by multiplying both sides of (7.7) by  $V_m^*$  and making use of the orthonormality of  $\{v_1, \dots, v_m\}$ .

As was noted earlier the algorithm breaks down when the norm of  $w$  computed on line (8) vanishes at a certain step  $j$ . As it turns out, this happens if and only if the starting vector  $v$  is a combination of  $j$  eigenvectors (i.e., the minimal polynomial of  $v_1$  is of degree  $j$ ). In addition, the subspace  $\mathcal{K}_j$  is then invariant and the approximate eigenvalues and eigenvectors are exact [387].

The approximate eigenvalues  $\lambda_i^{(m)}$  provided by the projection process onto  $\mathcal{K}_m$  are the eigenvalues of the Hessenberg matrix  $H_m$ . These are known as *Ritz values*. A *Ritz approximate eigenvector* associated with a Ritz value  $\lambda_i^{(m)}$  is defined by  $u_i^{(m)} = V_m y_i^{(m)}$ , where  $y_i^{(m)}$  is an eigenvector associated with the eigenvalue  $\lambda_i^{(m)}$ . A number of the Ritz eigenvalues, typically a small fraction of  $m$ , will usually constitute good approximations for corresponding eigenvalues  $\lambda_i$  of  $A$ , and the quality of the approximation will usually improve as  $m$  increases.

The original algorithm consists of increasing  $m$  until all desired eigenvalues of  $A$  are found. For large matrices, this becomes costly both in terms of computation and storage. In terms of storage, we need to keep  $m$  vectors of length  $n$  plus an  $m \times m$  Hessenberg matrix, a total of approximately  $nm + m^2/2$ . For the arithmetic costs, we need to multiply  $v_j$  by  $A$ , at the cost of  $2 \times N_z$ , where  $N_z$  is number of nonzero elements in  $A$ , and then orthogonalize the result against  $j$  vectors at the cost of  $4(j+1)n$ , which increases with the step number  $j$ . Thus an  $m$ -dimensional Arnoldi procedure costs  $\approx nm + m^2/2$  in storage and  $\approx N_z + 2nm^2$  in arithmetic operations.

Obtaining the residual norm, for a Ritz pair, as the algorithm progresses is fairly inexpensive. Let  $y_i^{(m)}$  be an eigenvector of  $H_m$  associated with the eigenvalue  $\lambda_i^{(m)}$ , and let  $u_i^{(m)}$  be the Ritz approximate eigenvector  $u_i^{(m)} = V_m y_i^{(m)}$ . We have the relation

$$(A - \lambda_i^{(m)} I) u_i^{(m)} = h_{m+1,m} (e_m^* y_i^{(m)}) v_{m+1},$$

and, therefore,

$$\|(A - \lambda_i^{(m)} I) u_i^{(m)}\|_2 = h_{m+1,m} |e_m^* y_i^{(m)}|. \quad (7.9)$$

Thus, the residual norm is equal to the absolute value of the last component of the eigenvector  $y_i^{(m)}$  multiplied by  $h_{m+1,m}$ . The residual norms are not always indicative of actual errors in  $\lambda_i^{(m)}$ , but can be quite helpful in deriving stopping procedures.

### 7.5.2 Variants

The description of the Arnoldi procedure given earlier was based on the modified Gram–Schmidt process. Other orthogonalization algorithms could be used as well. One improvement is to reorthogonalize when necessary. Whenever the final vector obtained at the end of the second loop in the above algorithm has been computed, a test is performed to compare its norm with the norm of the initial  $w$  (which is  $\|Av_j\|_2$ ). If the reduction falls below a certain threshold (an indication that severe cancelation might have occurred), a second orthogonalization is made. It is known from a result by Kahan that more than two orthogonalizations are superfluous (see, for example, Parlett [353]).

One of the most reliable orthogonalization techniques, from the numerical point of view, is the Householder algorithm [198]. This has been implemented for the Arnoldi procedure by Walker [455]. The Householder algorithm is numerically more reliable than the Gram–Schmidt or modified Gram–Schmidt versions, but it is also more expensive, requiring roughly the same storage as modified Gram–Schmidt but about twice as many operations. The Householder orthogonalization is a reasonable choice when developing general purpose, reliable software packages where robustness is a critical criterion.

### 7.5.3 Explicit Restarts

As was mentioned earlier, the standard implementations of the Arnoldi method are limited by their high storage and computational requirements as  $m$  increases. Suppose that we are interested in only one eigenvalue/eigenvector of  $A$ , namely, the eigenvalue of largest real part of  $A$ . Then one way to circumvent the difficulty is to *restart* the algorithm. After a run with  $m$  Arnoldi vectors, we compute the approximate eigenvector and use it as an initial vector for the next run with the Arnoldi method. This process, which is the simplest of this kind, is iterated to convergence to compute one eigenpair. For computing other eigenpairs, and for improving the efficiency of the process, a number of strategies have been developed, which are somewhat related. These include deflation procedures briefly discussed in the next section, and the implicit restarting strategy described in §7.6.

**ALGORITHM 7.4: Explicitly Restarted Arnoldi Method for NHEP**

- (1) **Iterate:** Perform  $m$  steps of Algorithm 7.3
- (2) **Restart:** Compute the approximate eigenvector  $u_1^{(m)}$  associated with the rightmost eigenvalue  $\lambda_1^{(m)}$ .
- (3) If satisfied stop, else set  $v_1 \equiv u_1^{(m)}$  and goto (1)

### 7.5.4 Deflation

We now consider the following implementation which incorporates a deflation process. So far we have described algorithms that compute only one eigenpair. In case several eigenpairs are sought, there are two possible options. The first is to take  $v_1$  to be a

linear combination of the approximate eigenvectors when we restart. For example, if we need to compute the  $p$  rightmost eigenvectors, we may take

$$\hat{v}_1 = \sum_{i=1}^p \rho_i \tilde{u}_i,$$

where the eigenvalues are numbered in decreasing order of their real parts. The vector  $v_1$  is then obtained from normalizing  $\hat{v}_1$ . The simplest choice for the coefficients  $\rho_i$  is to take  $\rho_i = 1, i = 1, \dots, p$ . There are several drawbacks to this approach, the most important of which being that there is no easy way of choosing the coefficients  $\rho_i$  in a systematic manner. The result is that for hard problems, convergence is difficult to achieve.

A more reliable alternative is to compute one eigenpair at a time and use deflation. The matrix  $A$  can be deflated explicitly by constructing progressively the first  $k$  Schur vectors. If a previous orthogonal basis  $U_{k-1} = [u_1, \dots, u_{k-1}]$  of the invariant subspace has already been computed, then to compute the eigenvalue  $\lambda_k$ , we can work with the matrix

$$\tilde{A} = A - U_{k-1} \Sigma U_{k-1}^*,$$

in which  $\Sigma = \text{diag}(\sigma_i)$  is a diagonal matrix of shifts. The eigenvalues of  $\tilde{A}$  consist of two groups. Those eigenvalues associated with the Schur vectors  $u_1, \dots, u_{k-1}$  will be shifted to  $\tilde{\lambda}_i = \lambda_i - \sigma_i$  and the others remain unchanged. If the eigenvalues with largest real parts are sought, then the shifts are selected so that  $\lambda_k$  becomes the next eigenvalue with largest real part of  $\tilde{A}$ . It is also possible to deflate by simply projecting out the components associated with the invariant subspace spanned by  $U_{k-1}$ ; this would lead to operating with the matrix

$$\tilde{A} = A(I - U_{k-1} U_{k-1}^*).$$

Note that if  $AU_{k-1} = U_{k-1}R_{k-1}$  is the partial Schur decomposition associated with the first  $k-1$  Ritz values, then  $\tilde{A} = A - U_{k-1}R_{k-1}U_{k-1}^*$ . Those eigenvalues associated with the Schur vectors  $u_1, \dots, u_{k-1}$  will now all be moved to zero.

A better implementation of deflation, which fits in well with the Arnoldi procedure, is to work with a single basis  $v_1, v_2, \dots, v_m$  whose first vectors are the Schur vectors that have already converged. Suppose that  $k-1$  such vectors have converged and call them  $v_1, v_2, \dots, v_{k-1}$ . Then we start by choosing a vector  $v_k$  which is orthogonal to  $v_1, \dots, v_{k-1}$  and of norm 1. Next we perform  $m-k$  steps of an Arnoldi procedure in which orthogonality of the vector  $v_j$  against all previous  $v_i$ 's, including  $v_1, \dots, v_{k-1}$  is enforced. This generates an orthogonal basis of the subspace

$$\text{span}\{v_1, \dots, v_{k-1}, v_k, Av_k, \dots, A^{m-k}v_k\}. \quad (7.9)$$

Thus, the dimension of this modified Krylov subspace is constant and equal to  $m$  in general. A sketch of this implicit deflation procedure combined with the Arnoldi method appears in the following.

**ALGORITHM 7.5: Explicitly Restarted Arnoldi Method with Deflation for NHEP**

```

(1) set $k := 1$
(2) Loop
(3) for $j = k, k + 1, \dots, m$ do
(4) $w := Av_j$.
(5) compute a set of j coefficients h_{ij} so that $w := w - \sum_{i=1}^j h_{ij}v_i$
 is orthogonal to all previous v_i s, $i = 1, 2, \dots, j$.
(6) $h_{j+1,j} = \|w\|_2$
(7) $v_{j+1} = w/h_{j+1,j}$.
(8) compute approximate eigenvector of A associated with the
 eigenvalue $\tilde{\lambda}_k$ and its associated residual norm estimate ρ_k .
(9) orthonormalize this eigenvector against all previous v_j s to get
 the approximate Schur vector \tilde{u}_k and define $v_k := \tilde{u}_k$.
(10) if ρ_k is small enough, then (accept eigenvalue)
(11) $h_{i,k} = v_i^*Av_k$, $i = 1, \dots, k$,
(12) set $k := k + 1$,
(13) if $k \geq nev$, then stop else goto (2)
(14) else
(15) go to (3)
(16) end if

```

Note that in the Loop, the Schur vectors associated with the eigenvalues  $\lambda_1, \dots, \lambda_{k-1}$  will not be touched in subsequent steps. They are sometimes referred to as “locked vectors.” Similarly, the corresponding upper triangular matrix corresponding to these vectors is also locked.

$$\underbrace{[v_1, v_2, \dots, v_{k-1}, v_k, v_{k+1}, \dots, v_m]}_{\text{Locked}} \quad \underbrace{[v_k, v_{k+1}, \dots, v_m]}_{\text{Active}}$$

When a new Schur vector converges, step (10) computes the  $k$ th column of  $R$  associated with this new basis vector. In the subsequent steps, the approximate eigenvalues are the eigenvalues of the  $m \times m$  Hessenberg matrix  $H_m$  defined in the algorithm and whose  $k \times k$  principal submatrix is upper triangular. For example, when  $m = 6$  and after the second Schur vector,  $k = 2$ , has converged, the matrix  $H_m$  will have the form

$$H_m = \begin{bmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \end{bmatrix}. \quad (7.10)$$

In the subsequent steps, only the eigenvalues not associated with the  $2 \times 2$  upper triangular matrix need to be considered.

It can be shown that, in exact arithmetic, the  $(n-k) \times (n-k)$  Hessenberg matrix in the lower  $(2 \times 2)$  block is the same matrix that would be obtained from an Arnoldi run applied to the matrix

$$\tilde{A} = (I - U_{k-1}U_{k-1}^*)A.$$

Thus, we are implicitly projecting out the invariant subspace already computed from the range of  $A$ .

## 7.6 Implicitly Restarted Arnoldi Method

*R. Lehoucq and D. Sorensen*

Perhaps the most successful numerical algorithm for computing the complete eigensystem of a general square matrix  $A$  is the implicitly shifted QR algorithm. One of the keys to the success of this method is its relationship to the Schur decomposition

$$A = UTU^*. \quad (7.11)$$

This well-known decomposition asserts that every square matrix  $A$  is *unitarily similar* to an upper triangular matrix  $T$ .

The QR algorithm produces a sequence of unitary similarity transformations that iteratively reduce  $A$  to upper triangular form (see §7.3). In other words, it computes a Schur decomposition. A practical implementation of the QR algorithm begins with an initial unitary similarity transformation of  $A$  to the condensed form  $V^*AV = H$  where  $H$  is upper Hessenberg (“almost upper triangular”) and  $V$  is unitary. Then the following iteration is performed.

```

SHIFTED QR METHOD
factor $V^*AV = H$
for $j = 1, 2, 3, \dots$, until convergence
 select a shift μ
 QR-factorize $QR = H - \mu I$
 $H := Q^*HQ$
 $V := VQ$
end for

```

In this scheme,  $Q$  is unitary and  $R$  is upper triangular (i.e., the QR factorization of  $H - \mu I$ ). It is easy to see that  $H$  is unitarily similar to  $A$  throughout the course of this iteration. The iteration is continued until the subdiagonal elements of  $H$  converge to zero, i.e., until a Schur decomposition has been (approximately) obtained.

If  $U_k$  represents the leading  $k$  columns of  $U$ , and  $T_k$  the leading principal  $k \times k$  submatrix of  $T$  in (7.11), then

$$AU_k = U_k T_k,$$

and we refer to this as a *partial Schur decomposition* of  $A$ . Since there is a Schur decomposition with the eigenvalues of  $A$  appearing on the diagonal in any given ordering, there is always a partial Schur decomposition of  $A$  with the diagonal elements of  $T_k$  consisting of any specified subset of  $k$  eigenvalues of  $A$ . Moreover,  $\text{span}(U_k)$  is an invariant subspace of  $A$  for these eigenvalues.

We are going to exploit this aspect of the Schur decomposition. We shall also exploit the fact that a variant of the QR iteration can be devised that will force eigenvalues of interest to appear in this leading  $k \times k$  block of  $T$  as the iteration proceeds. Our goal will be to develop a truncated form of the QR method that is suitable for computing a selected set of  $k$  eigenvalues of a large matrix  $A$ . We call implicitly restarted Arnoldi method the IRAM.

### 7.6.1 Arnoldi Procedure in GEMV Form

To begin our discussion, we shall look at the Arnoldi procedure from a slightly different perspective. The Arnoldi procedure was derived previously in §7.5 and has been presented in modified Gram–Schmidt algorithmic form in Algorithm 7.3. In this section, we describe it in matrix–vector (or GEMV) form using the classical Gram–Schmidt process together with the orthogonalization refinement [96]. The Arnoldi relation between the matrix  $A$ , the basis matrix  $V_k$ , and the residual vector  $f_k$  is of the form

$$AV_k = V_k H_k + f_k e_k^*,$$

where  $V_k \in \mathbb{C}^{n \times k}$  has orthonormal columns,  $V_k^* f_k = 0$ , and  $H_k \in \mathbb{C}^{k \times k}$  is upper Hessenberg with nonnegative subdiagonal elements. We shall call this a *k-step Arnoldi factorization* of  $A$ . It is easily seen from the construction that  $H_k = V_k^* A V_k$  is upper Hessenberg. When  $A$  is Hermitian, this implies that  $H_k$  is real symmetric and tridiagonal. The columns of  $V_k$  are referred to as the *Arnoldi vectors*.

A template for computing a *k*-step Arnoldi factorization is given in Algorithm 7.6.

**ALGORITHM 7.6: *k*-Step Arnoldi Factorization**

```

(1) $v_1 = v / \|v\|$ of the starting vector v
(2) $w = Av_1$
(3) $h = v_1^* w; f_1 = w - v_1 h$
(4) $V_1 = [v_1]; H_1 = [h]$
(5) for $j = 1, 2, 3, \dots, k - 1$
(6) $\beta_j = \|f_j\|$
(7) $v_{j+1} = f_j / \beta_j$
(8) $V_{j+1} := [V_j, v_{j+1}]; \quad \widehat{H}_j := \begin{bmatrix} H_j \\ \beta_j e_j^* \end{bmatrix}$
(9) $w = Av_{j+1}$
(10) $h = V_{j+1}^* w$
(11) $f_{j+1} = w - V_{j+1} h$
(12) if ($\|f_{j+1}\| < \eta \|h\|$) then
(13) $s = V_{j+1}^* f_{j+1}$
(14) $f_{j+1} := f_{j+1} - V_{j+1} s$
(15) $h := h + s$
(16) end if
(17) $H_{j+1} := [\widehat{H}_j, h]$
(18) end for

```

We will now describe some implementation details, referring to the respective phases in Algorithm 7.6.

- (1) Choose the initial starting vector  $v$  and normalize. Ideally, for eigenvalue calculations, one should attempt to construct a  $v$  that is dominant in eigenvector directions of interest. In the absence of any other considerations, a random vector is a reasonable choice.
- (2) Initialize the Arnoldi procedure. In the subsequent step,  $f_1$  will become the new Arnoldi basis vector. This step should include a reorthogonalization just as all of the others do.

- (7) Normalize  $f_j$  to get the new basis vector  $v_{j+1}$ , partially update  $H_j$  to get the  $(j + 1) \times j$  matrix  $\widehat{H}_j$ , and compute the new information  $w \leftarrow Av_{j+1}$ . The norm calculation  $\beta_j = \|f_j\|$  need not be recomputed here. It has already been computed at step (12) and can be saved for reuse here.
- (10) This is the classical Gram–Schmidt step to orthogonalize  $w$  with respect to the columns of  $V_{j+1}$ . This formulation allows level 2 matrix-vector operations. A more detailed discussion is given below.
- (12) The sequence of statements in this if-clause assure that the new direction  $f_{j+1}$  is numerically orthogonal to the previously computed directions, i.e., to the columns of  $V_{j+1}$ . The parameter  $\eta$  must be specified ( $0 \leq \eta < \infty$ ). The test asks the question, “Is  $w = Av_{j+1}$  nearly in the space that already has been constructed?” The ratio  $\frac{\|f_{j+1}\|}{\|h\|} = \frac{\sin(\theta)}{\cos(\theta)}$ , where  $\theta$  is the angle that the vector  $w$  makes with  $\text{span}(V_{j+1})$  (i.e., the angle  $\theta$  between  $w$  and its projection  $V_{j+1}h$ ). A larger value of  $\eta$  is a more stringent orthogonality requirement. With  $\eta = 0$  the algorithm reverts to classical Gram–Schmidt (CGS). An additional detail is needed to completely specify this process. If, after updating  $f_{j+1}$  and  $h$ , the relation  $\|f_{j+1}\| < \eta\|h\|$  still holds, then numerically we have  $w \in \text{span}(V_{j+1})$ . In this case, the Arnoldi procedure should either be terminated or a randomly generated vector should be orthogonalized with respect to the columns of  $V_{j+1}$  to replace  $f_{j+1}$ , and  $\beta_{j+1}$  should be set to zero to continue the factorization. The value of  $\eta$  used in the ARPACK implementation is  $1/\sqrt{2}$  (see §7.6.9). Additional discussion of this orthogonalization device is given below.
- (17)  $H_{j+1}$  is now a  $(j + 1) \times (j + 1)$  upper Hessenberg matrix.

The dense matrix-vector products  $V_{j+1}^*w$  and  $V_{j+1}h$  may be expressed with the Level 2 BLAS operation GEMV. This provides a significant performance advantage on virtually every platform from workstation to supercomputer. Moreover, considerable effort has been made within the ScaLAPACK project [52] and also by several high-performance computer vendors to optimize these kernels for a variety of parallel machines.

The mechanism used to orthogonalize the new information  $Av_k$  against the existing basis  $V_k$  is the CGS. It is notoriously unstable and will fail miserably in this setting without modification. One remedy is to use the modified Gram–Schmidt process as used in Algorithm 7.3. Unfortunately, this will also fail to produce orthogonal vectors in the restarting situation we are about to discuss and it cannot be expressed with Level 2 BLAS in this setting. At step (5), we have included an iterative refinement technique proposed by Daniel, Gragg, Kaufman, and Stewart (DGKS) in [96]. This scheme provides an excellent way to construct a vector  $f_{j+1}$  that is numerically orthogonal to  $V_{j+1}$ . The simple test specified at step (5) is used to avoid this DGKS correction if it is not needed.

This mechanism maintains orthogonality to full working precision at very reasonable cost. The special situation imposed by the restarting scheme we are about to discuss makes this modification essential for obtaining accurate eigenvalues and numerically orthogonal Schur vectors (eigenvectors in the Hermitian case). Schemes based on MGS are often sufficient for solving linear systems because they do construct a basis set that is linearly independent even though it might not be numerically orthogonal. The quality of MGS orthogonality is dependent upon the condition number (linear independence)

of the original set of vectors. The implicit restarting mechanism we are about to present will be less effective and may even fail if numerical orthogonality is not maintained.

It has been well documented that failure to maintain orthogonality leads to several numerical difficulties within the Lanczos or Arnoldi procedure. In the Hermitian case, Paige [347] showed that the loss of orthogonality occurs precisely when an eigenvalue of  $H_j$  is close to an eigenvalue of  $A$ . In fact, the Arnoldi vectors lose orthogonality in the direction of the associated approximate eigenvector. Moreover, failure to maintain orthogonality results in spurious copies of the approximate eigenvalue produced by the Arnoldi method. In the Hermitian case, several remedies short of full reorthogonalization have been proposed. These are discussed in §4.4 (p. 56).

### 7.6.2 Implicit Restart

An unfortunate aspect of the Lanczos or Arnoldi procedure is that there is no way to determine in advance how many steps will be needed to determine the eigenvalues of interest within a specified accuracy. The eigeninformation obtained through this process is completely determined by the choice of the starting vector  $v_1$ . Unless there is a very fortuitous choice of  $v_1$ , eigeninformation of interest probably will not appear until  $k$  gets very large. Clearly, it becomes intractable to maintain numerical orthogonality of  $V_k$ . Extensive storage will be required, and repeatedly finding the eigensystem of  $H_k$  also becomes expensive at a cost of  $O(k^3)$  floating point operations.

The obvious need to control this cost has motivated the development of restarting schemes. Restarting means replacing the starting vector  $v_1$  with an “improved” starting vector  $v_1^+$  and then computing a new Arnoldi factorization with the new vector. The structure of  $f_k$  serves as a guide: Our goal is to iteratively force  $v_1$  to be a linear combination of eigenvectors of interest. In theory,  $f_k$  will vanish if  $v_1$  is a nontrivial linear combination of  $k$  eigenvectors of  $A$ . However, a more general and, in fact, a better numerical strategy, is to force the starting vector to be a linear combination of Schur vectors that span the desired invariant subspace.

The need for restarting was recognized early on by Karush [258] soon after the appearance of the original algorithm of Lanczos [285]. Subsequently, there were developments by Paige [347], Cullum and Donath [89], and Golub and Underwood [197]. More recently, a restarting scheme for eigenvalue computation was proposed by Saad based upon the polynomial acceleration scheme originally introduced by Manteuffel [316] for the iterative solution of linear systems. All of these schemes are *explicit* in the sense that a new starting vector is produced by some process, and then an entirely new Arnoldi factorization is constructed.

There is another approach to restarting that offers a more efficient and numerically stable formulation. This approach, called *implicit restarting*, is a technique for combining the implicitly shifted QR scheme with a  $k$ -step Arnoldi or Lanczos factorization to obtain a truncated form of the implicitly shifted QR iteration. The numerical difficulties and storage problems normally associated with Arnoldi and Lanczos procedures are avoided. The algorithm is capable of computing a few ( $k$ ) eigenvalues with user-specified features such as largest real part or largest magnitude using  $2nk + O(k^2)$  storage. The computed Schur basis vectors for the desired  $k$ -dimensional eigenspace are numerically orthogonal to working precision.

Implicit restarting provides a means to extract interesting information from large Krylov subspaces while avoiding the storage and numerical difficulties associated with

the standard approach. It does this by continually compressing the interesting information into a fixed-size  $k$ -dimensional subspace. This is accomplished through the implicitly shifted QR mechanism. An Arnoldi factorization of length  $m = k + p$ ,

$$AV_m = V_m H_m + f_m e_m^*, \quad (7.12)$$

is compressed to a factorization of length  $k$  that retains the eigeninformation of interest. This is accomplished using QR steps to apply  $p$  shifts implicitly. The first stage of this shift process results in

$$AV_m^+ = V_m^+ H_m^+ + f_m e_m^* Q, \quad (7.13)$$

where  $V_m^+ = V_m Q$ ,  $H_m^+ = Q^* H_m Q$ , and  $Q = Q_1 Q_2 \cdots Q_p$ . Each  $Q_j$  is the orthogonal matrix associated with the shift  $\mu_j$  used during the shifted QR algorithm. Because of the Hessenberg structure of the matrices  $Q_j$ , it turns out that the first  $k - 1$  entries of the vector  $e_m^* Q$  are zero. This implies that the leading  $k$  columns in equation (7.13) remain in an Arnoldi relation. Equating the first  $k$  columns on both sides of (7.13) provides an updated  $k$ -step Arnoldi factorization

$$AV_k^+ = V_k^+ H_k^+ + f_k^+ e_k^*, \quad (7.14)$$

with an updated residual of the form  $f_k^+ = V_m^+ e_{k+1} \hat{\beta}_k + f_m \sigma$ . Using this as a starting point it is possible to apply  $p$  additional steps of the Arnoldi procedure to return to the original  $m$ -step form.

A template for computing a  $k$ -step Arnoldi factorization with implicit restart is given in Algorithm 7.7.

**ALGORITHM 7.7: IRAM for NHEP**

- (1) starting with  $v_1 = v / \|v\|$
- (2) compute an  $m$ -step Arnoldi factorization  

$$AV_m = V_m H_m + f_m e_m^*$$
- (3) repeat until convergence
  - (4) compute  $\sigma(H_m)$  and select  $p$  shifts  $\mu_1, \mu_2, \dots, \mu_p$
  - (5)  $Q = I_m$
  - (6) for  $j = 1, 2, \dots, p$ 
    - (7) QR factorize  $Q_j R_j = H_m - \mu_j I$
    - (8)  $H_m := Q_j^* H_m Q_j$
    - (9)  $Q := Q Q_j$
  - (10) end
  - (11)  $\beta_k = H_m(k+1, k); \sigma_k = Q(m, k)$
  - (12)  $f_k := v_{k+1} \beta_k + f_m \sigma_k$
  - (13)  $V_k := V_m Q(:, 1:k); H_k := H_m(1:k, 1:k)$
  - (14) beginning with the  $k$ -step Arnoldi factorization
    - $$AV_k = V_k H_k + f_k e_k^*$$
    - apply  $p$  additional steps of the Arnoldi procedure
    - to obtain a new  $m$ -step Arnoldi factorization
    - $$AV_m = V_m H_m + f_m e_m^*$$
  - (15) end repeat

We will now describe some implementation details, referring to the respective phases in Algorithm 7.7.

- (1) Choose the initial starting vector  $v$ , normalize and compute an initial ( $m = k + p$ )-step Arnoldi factorization. Ideally, for eigenvalue calculations, one should attempt to construct a  $v$  that is dominant in eigenvector directions of interest. If there is a sequence of closely related eigenvalue problems, this vector should be taken as a linear combination of the eigenvectors (or Schur vectors) of interest from the previous problem. In the absence of any other considerations, a random vector is a reasonable choice.
- (3) A Ritz pair  $(x, \theta)$  is “converged” if  $\|f_k\| |e_k^* y| < \|H_k\| \epsilon_D$  where  $x = V_k y$  and  $\theta$  is “wanted.” Upon convergence, this pair should be deflated. See the discussion of deflation in §7.6.6.
- (4) Shift selection: The shifts  $\mu_j$  are selected with respect to the user’s “wanted set” specification and the current and perhaps past information about the spectrum of  $H_m$ . A successful strategy has been to sort the eigenvalues of  $H_m$  into a “wanted set”  $\theta_1, \theta_2, \dots, \theta_k$  and an “unwanted set”  $\mu_1, \mu_2, \dots, \mu_p$  and take the latter as the selected set of shifts. This is called the *exact shift* strategy. Other strategies are discussed below.

Examples of the “wanted set” specification are

- the  $k$  eigenvalues with largest real part,
- the  $k$  eigenvalues with largest magnitude,
- the  $k$  eigenvalues with smallest real part,
- the  $k$  eigenvalues with smallest magnitude.

- (6)–(10) Apply  $p$  steps of the shifted QR iteration to  $H_m$  using the  $\mu_1, \mu_2, \dots, \mu_p$  as shifts. This should be done using the implicitly shifted QR variant. If exact shifts are used, then  $\beta_k = H_m(k+1, k)$  should be zero on completion of the  $p$  steps of QR, and the leading submatrix  $H_m(1 : k, 1 : k)$  should have  $\theta_1, \theta_2, \dots, \theta_k$  as its eigenvalues.
- (12) When exact shifts are used, then the properties mentioned in (6)–(10) are usually approximately true in finite precision. However, they might not be achieved due to rounding errors. Therefore, it is important to include both terms in the update of the residual vector  $f_k \leftarrow v_{k+1} \beta_k + f_m \sigma_k$ , even though the term  $v_{k+1} \beta_k$  should be zero in exact arithmetic. Note, with exact shifts, the updated  $V_k$  and  $H_k$  will provide a new  $k$ -step Arnoldi factorization with Ritz values and vectors that are the best approximations to the user specification that have been produced so far.
- (14) This step requires a slight modification of the Arnoldi factorization discussed previously. It simply begins with step  $k$  of the usual scheme.

There are many ways to select the shifts  $\{\mu_j\}$  that are applied by the QR steps. Virtually any explicit polynomial restarting scheme could be applied through this implicit mechanism. Considerable success has been obtained with the choice of *exact shifts*. This selection is made by sorting the eigenvalues of  $H_m$  into two disjoint sets

of  $k$  “wanted” and  $p$  “unwanted” eigenvalues and using the  $p$  unwanted ones as shifts. With this selection, the  $p$  shift applications result in  $H_k^+$  having the  $k$  wanted eigenvalues as its spectrum.

Other interesting strategies include the roots of Chebyshev polynomials [383], harmonic Ritz values [331, 337, 349, 411], the roots of Leja polynomials [23], the roots of least squares polynomials [384], and refined shifts [244]. In particular, the Leja and harmonic Ritz values have been used to estimate the interior eigenvalues of  $A$ .

### 7.6.3 Convergence Properties

As convergence takes place, the subdiagonals of  $H_k$  tend to zero and the most desired eigenvalue approximations appear as eigenvalues of the leading  $k \times k$  block of  $R$  in a Schur decomposition of  $A$ . The basis vectors  $V_k$  tend to orthogonal Schur vectors.

An alternate interpretation stems from the fact that each of these shift cycles results in the implicit application of a polynomial in  $A$  of degree  $p$  to the starting vector.

$$v_1 \leftarrow \psi(A)v_1 \quad \text{with} \quad \psi(\lambda) = \prod_{j=1}^p (\lambda - \mu_j). \quad (7.15)$$

The roots of this polynomial are the shifts used in the QR process and these may be selected to enhance components of the starting vector in the direction of eigenvectors corresponding to desired eigenvalues and damp the components in unwanted directions. Of course, this is desirable because it forces the starting vector into an invariant subspace associated with the desired eigenvalues. This in turn forces  $f_k$  to become small and hence convergence results. Full details may be found in [419].

There is a fairly straightforward intuitive explanation of how this repeated updating of the starting vector  $v_1$  through implicit restarting might lead to convergence. If  $v_1$  is expressed as a linear combination of eigenvectors  $\{q_j\}$  of  $A$ , then

$$v_1 = \sum_{j=1}^n q_j \gamma_j \quad \Rightarrow \quad \psi(A)v_1 = \sum_{j=1}^n q_j \psi(\lambda_j) \gamma_j.$$

Applying the same polynomial (i.e., using the same shifts) repeatedly for  $\ell$  iterations will result in the  $j$ th original expansion coefficient being attenuated by a factor

$$\left( \frac{\psi(\lambda_j)}{\psi(\lambda_1)} \right)^\ell,$$

where the eigenvalues have been ordered according to decreasing values of  $|\psi(\lambda_j)|$ . The leading  $k$  eigenvalues become dominant in this expansion and the remaining eigenvalues become less and less significant as the iteration proceeds. Hence, the starting vector  $v_1$  is forced into an invariant subspace as desired. The adaptive choice provided with the exact shift mechanism further enhances the isolation of the wanted components in this expansion. Hence, the wanted eigenvalues are approximated better and better as the iteration proceeds.

It is worth noting that if  $m = n$  then  $f_m = 0$  and this iteration is precisely the same as the implicitly shifted QR iteration. Even for  $m < n$ , the first  $k$  columns of  $V_m$  and the Hessenberg submatrix  $H_m(1 : k, 1 : k)$  are mathematically equivalent to the

matrices that would appear in the full implicitly shifted QR iteration using the same shifts  $\mu_j$ . In this sense, the implicitly restarted Arnoldi method may be viewed as a truncation of the implicitly shifted QR iteration. The fundamental difference is that the standard implicitly shifted QR iteration selects shifts to drive subdiagonal elements of  $H_n$  to zero from the bottom up while the shift selection in the implicitly restarted Arnoldi method is made to drive subdiagonal elements of  $H_m$  to zero from the top down. Shifted power-method-like convergence will be obtained.

When the exact shift strategy is used, implicit restarting can be viewed both as a means to damp unwanted components from the starting vector and also as directly forcing the starting vector to be a linear combination of wanted eigenvectors. See [419] for information on the convergence of IRAM and [22, 421] for other possible shift strategies for Hermitian  $A$ . The reader is referred to [292, 333] for studies comparing implicit restarting with other schemes.

#### 7.6.4 Numerical Stability

Robust implementations of a practical QR algorithm, such as are available in LAPACK [12], compute an approximate Schur form for a matrix  $A$  that satisfies

$$(A + E)\widehat{V} = \widehat{V}\widehat{R}, \quad (7.16)$$

where  $\widehat{R}$  is upper triangular,  $\|\widehat{V}^*\widehat{V} - I\| \approx \epsilon_M$ , and  $\|E\| \approx \epsilon_M\|A\|$ , where  $\epsilon_M$  is machine precision (unit roundoff). The fact that  $\|E\| \approx \epsilon_M\|A\|$  is a direct result of the exclusive use of unitary transformations during the iteration. This is what makes the QR algorithm backward stable. In other words, the QR method computes the Schur form of a matrix  $A + E$  that is *near*  $A$ .

Backward stability is a very reassuring quality for any numerical algorithm, but it is important to keep in mind that backward stability alone does not imply accurate answers. The accuracy of the computed eigenvalues and eigenvectors depends upon the sensitivity (conditioning) of the eigensystem of  $A$ . A backward stable algorithm produces accurate answers for well-conditioned problems. The reader is referred to §7.13 for a more detailed discussion of this issue.

As explained in the last section, the IRAM is a truncated QR iteration. In most implementations of the QR method, Householder transformations are used to obtain the initial reduction to Hessenberg form. The Arnoldi procedure, if extended to  $n$  steps, is simply another way to obtain this reduction. The introduction of the orthogonal correction [96] allows the Arnoldi procedure to produce a reduction of the same numerical quality as the Householder reduction but it is more suitable to the large scale setting. With this correction, the computed Arnoldi vectors are unitary to within machine precision ( $\|V_k^*V_k - I_k\| \approx \epsilon_M$ ), and since the restarting mechanism involves the same unitary transformations used in the QR mechanism, the IRAM is also backward stable.

At any stage of the IRAM iteration, we have

$$(A + E_o)V_k = V_kH_k + f_k e_k^*,$$

with  $\|E_o\| \approx \|A\|\epsilon_M$ . If we are successful in making  $\|f_k\| < \|A\|\epsilon_M$  then

$$(A + E)V_k = V_kH_k,$$

with  $E = E_o + f_k(V_k e_k)^*$ . If  $H_k Z_k = Z_k R_k$  is a Schur decomposition of  $H_k$  then

$$(A + E)\widehat{V}_k = \widehat{V}_k R_k \text{ with } \widehat{V}_k = V_k Z_k$$

is an exact partial Schur decomposition of a nearby matrix  $A + E$ .

It is more likely that convergence will be realized with a small last component of an eigenvector of  $H_m$  ( $m = k + p$ ) at some stage. We still have

$$(A + E_1)V_m = V_m H_m + f_m e_m^*,$$

with  $E_1$  small relative to  $\|A\|$ . If  $H_m y = y\theta$  where  $\eta = e_m^* y$  is small ( $\|f_m\|\|\eta\| < \epsilon_U \|H_m\|$ , say), then

$$(A + E)x = x\theta, \quad (7.17)$$

with  $x = V_m y$  and  $E = E_1 + f_m(e_m^* y)x^*$  (assuming  $\|y\| = 1$ ). Thus, in any case, converged approximate eigenpairs  $(x, \theta)$  are exact for a nearby matrix  $A + E$ .

Through the use of the deflation techniques that we shall present here in §7.6.6, it is possible to use unitary transformations to obtain a backward stability statement with respect to a partial Schur decomposition for a set of  $k$  converged eigenvalues. If there are  $k$  eigenvalues of  $H_m$  that have converged, it is possible to construct a unitary matrix  $Q$  such that the leading  $k \times k$  submatrix  $R_k$  of  $Q^* H_m Q$  is upper triangular and

$$(A + \widehat{E})\widehat{V}_k = \widehat{V}_k R_k, \text{ with } \widehat{V}_k = V_m Q,$$

where  $\widehat{E}$  is small relative to  $\|A\|$ . A discussion of this is given in §7.6.6, and more detail is available in [420].

### 7.6.5 Computational Costs and Tradeoffs

This implicit scheme costs  $p$  rather than the  $k + p$  matrix-vector products the explicit scheme would require to apply the same  $p$ -degree polynomial and rebuild the  $k$ -step Arnoldi factorization. When the operation  $w \leftarrow Av$  is relatively expensive, this savings can be considerable. However, there are tradeoffs. It is impossible to predict optimal values for  $k$  and  $p$  in general. The distribution of the spectrum of  $A$  has a great deal to do with convergence behavior. Two matrices with precisely the same sparsity pattern could exhibit completely different convergence characteristics for a given choice of  $k$  and  $p$ . Therefore, it would be misleading to draw any conclusions based solely on operation counts per iteration. Nevertheless, it can be useful to have a notion of the major costs of an iteration to guide initial choices.

In the following, we assume that the cost of a matrix-vector product  $w \leftarrow Av$  is  $\gamma n$ . For a sparse matrix, one could think of  $\gamma$  as twice the average number of nonzero entries per row of  $A$ . For a dense matrix  $\gamma = 2n$ , and for an FFT matrix  $\gamma \approx \log(n)$ . See §10.2. For a fixed value of  $k$  and  $p$ , the cost (in floating point operations) for one iteration of IRAM is

1. matrix-vector products  $w \leftarrow Av$ :  $\gamma pn$ ;
2. Arnoldi steps (extend from  $k$  to  $k + p$ ) :  $[(4k - 2)p + 2p^2]n$ ;

3. orthogonalization corrections: roughly  $4(k + p)n$  per correction. A worst-case estimate is  $[(4k - 2)p + 2p^2]n$  (assumes one correction per each new Arnoldi vector);
4. implicit restarting:  $2n(k^2 + kp) + O((k + p)^3)$ ;
5. total cost:  $\gamma pn + 2[(5k - 2)p + 2p^2]n + 2k^2n + O((k + p)^3)$ .

Note that the factor of 2 in the second term is due to a worst-case scenario with respect to orthogonalization corrections. This factor will usually be around 1.5 in practice. If the matrix-vector product  $Av$  is cheap ( i.e.  $\gamma$  is quite small) then the cost of implicit restarting is significant and alternatives should be considered. One of these might be to use a polynomial spectral transformation. This would involve operating with a suitably constructed polynomial function of  $\psi(A)$  in place of  $A$ . An example would be a Chebyshev polynomial designed to damp the unwanted portion of the spectrum. The action  $w \leftarrow \psi(A)$  would, of course, be applied as a succession of matrix-vector products involving  $A$ .

### 7.6.6 Deflation and Stopping Rules

Deflation is an important concept in the practical implementation of the QR iteration and therefore equally important to the IRAM. In the context of the QR iteration, deflation amounts to setting a small subdiagonal element of the Hessenberg matrix  $H$  to zero. This is called deflation because it splits the Hessenberg matrix into two smaller subproblems which may be independently refined further.

However, in the context of IRAM, the usual QR deflation techniques are not always appropriate. Additional deflation capabilities specific to implicit restarting are needed. It is often desirable to deflate at an accuracy level  $\epsilon_D$  with  $1 > \epsilon_D > \epsilon_M$ , where  $\epsilon_M$  is machine precision. In theory (i.e., in exact arithmetic), when  $A$  has a multiple eigenvalue corresponding to distinct eigenvectors, it would be impossible for IRAM to compute more than one instance of this multiplicity. This is because it is a “single vector” rather than a block method. However, in practice, there is usually little difficulty in computing multiple eigenvalues because the method deflates itself as convergence takes place, and roundoff usually introduces components in new eigenvector directions in the subsequent starting vectors. Nevertheless, this can be unreliable and miss a multiple eigenvalue. In any case, this approach typically will require a stringent convergence tolerance to succeed in finding all instances of a multiple eigenvalue. It is far more efficient to deflate (i.e., lock) an approximate eigenvalue once it has converged to a certain level of accuracy and then force subsequent Arnoldi vectors to be orthogonal to the converged subspace. With this capability, additional instances of a multiple eigenvalue can be computed to the same specified accuracy without the expense of converging them to unnecessarily high accuracy.

In the Arnoldi procedure, as with a QR iteration, it is possible for some of the leading  $k$  subdiagonals to become small during the course of implicit restarting. However, it is usually the case that there are converged Ritz values appearing in the spectrum of  $H$  long before small subdiagonal elements appear. This convergence is usually detected through observation of a small last component in an eigenvector  $y$  of  $H$ . When this happens, we are able to construct an orthogonal similarity transformation of  $H$  that will give an equivalent Arnoldi factorization with a slightly perturbed  $H$  that does indeed have a zero subdiagonal, and this is the basis of our deflation schemes.

In the context of IRAM, there are two types of deflation required:

*Locking:* If a Ritz value  $\theta$  has converged (meaning  $\|Ax - x\theta\| < \epsilon_D$ ) and thought to be a member of the wanted set of eigenvalues, then we wish to declare it converged, decouple the eigenpair  $(x, \theta)$ , and continue to compute remaining eigenvalues with no further alteration of  $x$  or  $\theta$ . This process is called locking.

*Purging:* If a Ritz value  $\theta$  has converged but is not a member of the wanted set, then we wish to decouple and remove the eigenpair  $(x, \theta)$  from the current Krylov subspace spanned by the Arnoldi vectors. This process is called purging.

Techniques for this deflation were first developed in [289, 294]. The scheme we present here is based an improvement developed in [420]. This scheme allows for stable and efficient deflation (or locking) of Ritz values that have converged with a specified relative accuracy of  $\epsilon_D$ , which may be considerably larger than machine precision  $\epsilon_M$ . This is particularly important when an SI is not available to accelerate convergence. Typically, in this setting the number of matrix-vector products will be large and it will be highly desirable to lock converged Ritz values at low tolerances. This will avoid the expense of the matrix-vector products that would be required to achieve an accuracy that would allow normal QR-type deflation. Also, it is very important to be able to purge converged but unwanted Ritz values. As pointed out in [289], the forward instability of the QR bulge-chase process discovered by Parlett and Le [359] will prevent implicit restarting to be used for purging converged unwanted Ritz values.

### 7.6.7 Orthogonal Deflating Transformation

We shall utilize a special orthogonal transformation to implement these deflation schemes. The deflation schemes are related to an eigenvector associated with a Ritz value that is to be deflated (either locked or purged). Given a vector  $y$  of unit length, Algorithm 7.8 computes an orthogonal matrix  $Q$  such that  $Qe_1 = y$  (hence  $y = Q^*e_1$ ). The following orthogonal deflating transformation Algorithm 7.8 is identical to Algorithm 4.9 used in IRLM (see §4.5), but for completeness, we present it again.

**ALGORITHM 7.8: Orthogonal Deflating Transformation for IRAM**

```

(1) start with the vector y of dimension k with $\|y\| = 1$.
(2) $Q = 0$; $Q(:, 1) = y$
(3) $\sigma = y(1)^2$; $\tau_o = |y(1)|$
(4) for $j = 2, \dots, k$
(5) $\sigma := \sigma + y(j)^2$, $\tau = \sqrt{\sigma}$
(6) if $\tau_o \neq 0$
(7) $\gamma = (y(j)/\tau)/\tau_o$
(8) $Q(1:j-1, j) = -y(1:j-1)\gamma$
(9) $Q(j, j) = \tau_o/\tau$
(10) else
(11) $Q(j-1, j) = 1$
(12) end if
(13) $\tau_o = \tau$
(14) end for

```

The orthogonal matrix  $Q$  constructed as prescribed in Algorithm 7.8 has a very special form and may be written as

$$Q = R + ye_1^*, \quad \text{with } Re_1 = 0, \quad R^*y = 0, \quad (7.18)$$

where  $R$  is upper triangular. It may also be written as

$$Q = L + yg^*, \quad \text{with } Le_1 = 0, \quad L^*y = e_1 - g, \quad (7.19)$$

where  $L$  is lower triangular, and  $g^* \equiv e_1^* + \frac{1}{\eta_1}e_1^*R$ .

Now, consider the matrix  $Q^*HQ$ . The substitutions  $Q^* = (L+yg^*)^*$ ,  $Q = (R+ye_1^*)$  from (7.19) and (7.18) will give

$$\begin{aligned} Q^*HQ &= Q^*H(R + ye_1^*) \\ &= (L^* + gy^*)HR + Q^*Hy e_1^* \\ &= L^*HR + gy^*HR + Q^*Hy e_1^*. \end{aligned}$$

Since both  $L^*$  and  $R$  are upper triangular, it follows that  $L^*HR$  is upper Hessenberg, with the first row and the first column each being zero due to  $Le_1 = Re_1 = 0$ .

From this discussion, we see that when  $y$  is a right eigenvector of  $H$  with  $Hy = y\theta$ , then  $H_+ \equiv Q^*HQ$  is of the form

$$H_+ = \begin{bmatrix} \theta & h^* \\ 0 & H_2 \end{bmatrix}. \quad (7.20)$$

On the other hand, when  $y$  is a left eigenvector of  $H$  with  $y^*H = \theta y^*$ , then  $H_+ \equiv Q^*HQ$  is of the form

$$H_+ = \begin{bmatrix} \theta & 0 \\ h & H_2 \end{bmatrix}, \quad (7.21)$$

where  $H_2$  is upper Hessenberg.

We shall use (7.20) for locking a converged Ritz value, and we shall use (7.21) to purge an unwanted but converged Ritz value.

It should be noted that, as computed by Algorithm 7.8,  $Q$  will have componentwise relative errors on the order of machine precision  $\epsilon_M$  with no element growth. Moreover, extension to complex arithmetic is completely straightforward (unlike Givens or Householder transformations).

**Locking or Purging a Single Eigenvalue.** The orthogonal transformations developed in the previous section will provide stable and efficient transformations needed to implement locking and purging. We shall give a somewhat detailed discussion of this deflation in complex arithmetic. However, it is clearly important from the standpoint of efficiency to be able to compute in real arithmetic if the matrix  $A$  is real and non-symmetric. In this case, it is usually desirable to lock or purge complex conjugate pairs of Ritz values as a unit so that complex arithmetic never need be introduced. This can be accomplished with a block formulation (block size = 2) of the single-vector case we are about to present. The purging in this case will be a direct analog. Unfortunately, there may be numerical complications with locking a complex conjugate pair. A com-

plete discussion would involve considerable detail and analysis. We feel these details are beyond the scope of a template presentation. They may be found in [420].

**Locking  $\theta$ .** The first instance to discuss is the locking of a single converged Ritz value. Assume that

$$Hy = y\theta, \quad \|y\| = 1,$$

with  $e_k^*y = \eta$ , where  $|\eta| \leq \epsilon_D\|H\|$ . Here, it is understood that  $\epsilon_M \leq \epsilon_D < 1$  is a specified relative accuracy tolerance between  $\epsilon_M$  and 1.

If  $\theta$  is “wanted” then it is desirable to lock  $\theta$ . However, in order to accomplish this it will be necessary to arrange a transformation of the current Arnoldi factorization to one with a small subdiagonal to isolate  $\theta$ . This may be accomplished by constructing a  $k \times k$  orthogonal matrix  $Q = Q(y)$  using Algorithm 7.8:

$$Qe_1 = y \quad \text{and} \quad e_k^*Q = (\eta, \tau e_{k-1}^*),$$

with  $\eta^2 + \tau^2 = 1$ .

Now, when we apply  $AVQ = VQ(Q^*HQ) + fe_k^*Q$ , we obtain

$$A[v_1, V_2] = [v_1, V_2] \begin{bmatrix} \theta & h^* \\ 0 & H_2 \end{bmatrix} + f(\eta, \tau e_{k-1}^*).$$

Unfortunately,  $H_2$  is not upper Hessenberg. Further work will have to be done to bring it to Hessenberg form. This must be done without disturbing the form of the residual term  $f(\eta, \tau e_{k-1}^*)$ . We need to construct an orthogonal  $U$  such that  $\hat{H}_2 = U^*H_2U$  is upper Hessenberg and  $e_{k-1}^*U = e_{k-1}^*$ . This can be done with Householder transformations or with a variant of the transformations defined in Algorithm 7.8 working from the last row upwards. The following MATLAB code segment shows a slightly inefficient way to obtain such a  $U$ .

```
rev = [k-1:-1:1];
C = H_2';
[U,H_2] = hess(C(rev,rev));
U = U(rev,rev);
H_2 = H_2(rev,rev);
```

Once  $U$  is constructed, replace  $Q$  with  $Q : Q \begin{bmatrix} 1 & 0 \\ 0 & U \end{bmatrix}$ , and the end result of these transformations will be

$$\begin{aligned} Av_1 &= v_1\theta + f\eta, \quad \text{where } v_1^*f = 0, \\ AV_2 &= [v_1, V_2] \begin{bmatrix} h^*, \\ H_2 \end{bmatrix} + f\tau e_{k-1}^*, \end{aligned}$$

where  $[v_1, V_2] = VQ$  and  $H_2$  is upper Hessenberg. After this, all subsequent implicit restarting takes place as if

$$AV_2 = V_2H_2 + f\tau e_{k-1}^*.$$

All the subsequent orthogonal transformations associated with implicit restarting are applied to  $V_2$  and  $\begin{bmatrix} h^* \\ H_2 \end{bmatrix}$  while never disturbing the relation  $Av_1 = v_1\theta + f\eta$ . In subsequent Arnoldi steps,  $v_1$  participates in the orthogonalization so that the selective

orthogonalization recommended by Parlett and Scott [363, 353] is accomplished automatically.

**Purging  $\theta$ .** If  $\theta$  is “unwanted” then we may wish to remove  $\theta$  from the spectrum of the projected matrix  $H$ . However, the implicit restart strategy using exact shifts will sometimes fail to purge a converged unwanted Ritz value [294].

We shall use (7.21) to purge an unwanted but converged Ritz value. In this case, a left eigenvector  $y$  is needed with

$$y^* h = \theta y^*, \quad \|y\| = 1.$$

Now, when we apply  $AVQ = VQ(Q^*HQ) + fe_k^*Q$ , we obtain

$$A[v_1, V_2] = [v_1, V_2] = \begin{bmatrix} \theta & 0 \\ h & H_2 \end{bmatrix} + f(\eta, \tau e_{k-1}^*),$$

where  $H_2$  is upper Hessenberg. Here,  $\eta = e_k^*y$  as before, but there is no requirement that  $\eta$  be small. The desired purging amounts to simply discarding the first column on both sides of this equation. We are then left with

$$AV_2 = V_2 H_2 + f\tau e_{k-1}^*.$$

No error other than an acceptable level of roundoff will be introduced through this purging process. Moreover, there is no requirement that  $y$  be an accurate left eigenvector for  $H$ . It is only necessary that the residual  $y^*H - \theta y^*$  be small.

**Stability of  $Q^*HQ$ .** The deflating orthogonal transformation construction shown in Algorithm 7.8 is clearly stable (i.e., componentwise relatively accurate representation of the transformation one would obtain in exact arithmetic). There is no question that the similarity transformation  $Q^*HQ$  numerically preserves the eigenvalues of  $H$ . However, there is a serious question about how well these transformations perform numerically in preserving Hessenberg form during purging. A modification to the basic algorithm is needed to assure that if  $y^*H = \theta y^*$ , then  $H_+ \equiv Q^*HQ$  is numerically Hessenberg (i.e., that the entries below the subdiagonal are all tiny relative to  $\|H\|$ ).

The fact that  $H_+ = Q^*HQ$  is Hessenberg depends upon the term  $gy^*HR$  vanishing in the expression

$$Q^*HQ = L^*HR + gy^*HR + \theta e_1 e_1^*.$$

However, on closer examination, we see that

$$e_1^*Q = e_1^*L + (e_1^*y)g^* = \eta_1 g^*,$$

where  $\eta_1$  is the first component of  $y$ . Therefore,

$$\|g\| = \frac{1}{|\eta_1|}, \tag{7.22}$$

so there may be numerical difficulty when the first component of  $y$  is small. To be specific,  $y^*H = \theta y^*$  and thus  $y^*HR = 0$  in exact arithmetic. However, in finite precision, the computed  $fl(y^*H) = \theta y^* + z^*$ . The error  $z$  will be on the order of  $\epsilon_M$  relative to  $\|H\|$ , but

$$\|gy^*HR\| = \|g\| \cdot \|z^*R\| = \frac{1}{|\eta_1|} \|z^*R\|,$$

so this term may be quite large. It may be as large as order  $O(1)$  if  $\eta_1 = O(\epsilon_M)$ . This is of serious concern and will occur in practice without the modification we now propose.

The remedy is to introduce a step-by-step acceptable perturbation and rescaling of the vector  $y$  to simultaneously force the conditions

$$Q^*y = e_1 \text{ and } y^*HQ = \theta e_1^*$$

to hold with sufficient accuracy in finite precision. To accomplish this, we shall devise a scheme to achieve

$$y^*Hq_j = 0 \text{ for } j > 1$$

numerically. As shown in [420], this modification is sufficient to establish  $q_i^*Hq_j = 0$  numerically, relative to  $\|H\|$  for  $i > j + 1$ .

The basic idea is as follows: If, at the  $j$ th step, the computed quantity  $fl(y^*Hq_j)$  is not sufficiently small, then it is adjusted to be small enough by scaling the vector  $y_j$  with a number  $\phi$  and the component  $\eta_{j+1}$  with a number  $\psi$ . With this rescaling just prior to the computation of  $q_{j+1}$ , we then have  $y_j \leftarrow y_j \phi$  and  $\hat{y}_j \leftarrow \hat{y}_j \psi$ , where  $y^* = [y_j^*, \hat{y}_j^*]$ . Certainly,  $\|y\|$  should not be altered with this scaling and this is therefore required. This gives the following system of equations to determine  $\phi$  and  $\psi$ : if  $|y_j^*H_j\hat{q}_j + \rho_j\beta_j\eta_{j+1}| > \epsilon_M\tau_{j+1}$ ,

$$\begin{aligned} (\tau_j\phi)^2 + (1 - \tau_j^2)\psi^2 &= 1, \\ y_j^*H_j\hat{q}_j\phi + \rho_j\beta_j\eta_{j+1}\psi &= \pm\epsilon_M\tau_{j+1}. \end{aligned}$$

If  $\rho_j$  is on the order of  $\sqrt{\epsilon_M}$ , then the scaling may be absorbed into  $\rho_j$  without alteration of  $y$  and also without effecting the numerical orthogonality of  $Q$ . When  $y$  is modified, it turns out that none of the previously computed  $q_i$ ,  $2 \leq i < j$ , need to be altered. After step  $j$ , the vector  $y_j$  is simply rescaled in subsequent steps, and the formulas defining  $q_i$ ,  $2 \leq i \leq j$ , are invariant with respect to scaling of  $y_j$ . For complete detail, one should consult [420].

The code shown in Algorithm 7.9 implements this scheme to compute an acceptable  $Q$ .<sup>3</sup> In practice, this transformation may be computed and applied in place to obtain  $H := Q^*HQ$  and  $V := VQ$  without storing  $Q$ . However, that implementation is quite subtle and the construction of  $Q$  is obscured by the details required to avoid creating additional storage. This code departs slightly from the above description since the vector  $y$  is rescaled to have unit norm only after all of the columns 2 to  $n$  have been determined.

---

<sup>3</sup>There is now a much better algorithm for constructing and applying these transformations; see D. Sorensen, Deflation for Implicitly Restarted Arnoldi Methods, CAAM Technical Report, TR 98-12, Rice University, 1998 (revised August 2000).

**ALGORITHM 7.9: Stable Orthogonal Deflating Transformation for IRAM**

```

(1) start with n -vector y with $\|y\| = 1$ and $\eta = e_n^* y$
 and T symmetric tridiagonal of order n with $Ty = \theta y$
(2) $Q = 0$
(3) while $y(i) = 0$, $y(i) = \epsilon_M/n$, $i := i + 1$, end while
(4) $\sigma = y(1)^2$, $\tau_o = |y(1)|$
(5) for $j = 2, \dots, n$
(6) $\sigma = \sigma + y(j)^2$, $\tau = \sqrt{\sigma}$
(7) $\gamma = -(y(j)/\tau)/\tau_o$
(8) $\rho = \tau_o/\tau$
(9) $Q(1:j-1, j) = y(1:j-1)\gamma$, $Q(j, j) = \rho$
(10) if $(j < n \text{ and } \tau < .05)$
(11) $\tau_p = \sqrt{\sigma + y(j+1)^2}$
(12) $\alpha = y(1:j)^* H(1:j, 1:j) Q(1:j, j)$
(13) $\delta = y(j+1) H(j, j+1) \rho$
(14) $\sigma_o = -1$
(15) if $\text{sign}(\alpha) \cdot \text{sign}(\delta) < 0$, $\sigma_o = 1$, end if
(16) if $|\delta + \alpha| > \epsilon_M \tau_p$
(17) if $\rho < \sqrt{\epsilon_M}/100$
(18) $Q(j, j) = Q(j, j) \sigma_o (\epsilon_M \tau_p + |\alpha|) / |\delta|$
(19) else
(20) if $|\alpha| > |\delta|$
(21) $\phi = \sigma_o (\epsilon_M \tau_p + |\delta|) / |\alpha|$
(22) $y(1:j) = y(1:j) \phi$
(23) $\sigma = \sigma \phi^2$, $\tau = \tau |\phi|$
(24) else
(25) $\psi = \sigma_o (\epsilon_M \tau_p + |\alpha|) / |\delta|$
(26) $y(j+1:n) = y(j+1:n) \psi$
(27) end if
(28) end if
(29) end if
(30) end if
(31) $\tau_0 = \tau$
(32) end for
(33) $Q(:, 1) = y / \|y\|$

```

There are several implementation issues.

- (3) The perturbation shown here avoids problems with exact zero initial entries in the eigenvector  $y$ . In theory, this should not happen when  $H$  is unreduced but it may happen numerically when a diagonal entry of  $H$  is small but not zero. There is a cleaner implementation possible that does not modify zero entries. This is the simplest (and crudest) correction.
- (10) As soon as  $\tau_j = \|y_j\|$  is sufficiently large, there is no need for any further corrections. Deleting this if-clause reverts to the unmodified calculation of  $Q$  shown in Algorithm 7.8.

- (16) This shows one of several possibilities for modifying  $y$  to achieve the desired goal of numerically tiny elements below the subdiagonal of  $Q^*HQ$ . More sophisticated strategies would absorb as much of the scaling as possible into the diagonal element  $Q(j,j) = \rho$ . Here, the branch that does scale  $\rho$  instead of  $y$  is designed so that  $\text{fl}(1 + (\rho\psi)^2) = \text{fl}(1 + \rho^2) = 1$ .

**Locking and Purging in IRAM.** Although the basic deflation ideas of locking and purging are conceptually straightforward, there are still a number of implementation details and various strategies one might consider. In the end, some ad-hoc decisions will have to be made. The deflation strategy adopted here is somewhat conservative. However, the performance appears to be quite reasonable in practice. In [420], computational examples demonstrate that very low tolerances can be specified without missing multiple or clustered eigenvalues.

The implementation details become quite involved and it is difficult to convey the ideas by displaying code. Instead, we shall indicate the main ideas of the strategy with a fairly high-level verbal description.

**ALGORITHM 7.10: Deflation for IRAM**

- (1) Lock a single Ritz value  $\theta$  each time one converges ( $\|Ax - x\theta\| < \epsilon_D$ ) until  $k$  values have been locked.
- (2) Continue to iterate and lock each newly converged Ritz value that is a “better” value than the existing ones. Follow each locking operation with a purge operation to delete the least wanted but locked Ritz value. Thus, there are always  $k$  locked Ritz values and vectors in place.
- (3) Continue step (2) until the next Ritz value to converge is not a “better” value. Replace the  $(k+1)$ st basis vector with a randomly generated one and orthogonalize this against the previous ones and then build a new  $((k+p)$ -step) Arnoldi factorization. Repeat step (2).
- (4) When step (3) has been executed two consecutive times with no replacement of existing locked Ritz values, the iteration is halted.

Implementation notes:

- (1) Initially, we work with an  $(m = k+p)$ -step Arnoldi factorization and apply  $p$  shifts per each implicit restart. Each time a Ritz value is locked, it is advantageous to decrease the effective value of  $p$  by 1 ( $p \leftarrow p - 1$ ). This allows the polynomial filter to have a larger relative magnitude on the Ritz value that is most likely to converge next (see §7.6.3). Of course, this must be limited to avoid lowering the degree of the filter so much that it becomes ineffective. If  $k_o$  is the number of locked Ritz values then the IRAM iteration takes place in columns  $k_o + 1 : m$  of  $V$  working within an  $(m - k_o)$ -length Arnoldi factorization. The effective value of  $k$  becomes  $m - k_o - p_o$  and the effective value of  $p$  becomes  $p_o = p - k_o$ . This has two important effects. The rate of convergence as described in §7.6.3 is increased and the amount of work per implicit restart is decreased.
- (2) One might also wish to purge all converged but unwanted Ritz pairs at this stage.

- (3) The purpose of introducing the random start vector here is to greatly increase the likelihood of components in directions of wanted eigenvectors that have not yet been found.
- (4) This ad-hoc stopping strategy is reasonable. However, there is no ultimate assurance that the  $k$  wanted eigenvalues have all been found (especially in the case of clustered or multiple eigenvalues).

### 7.6.8 Eigenvector Computation with Spectral Transformation

The SI was introduced in §3.3. Specifically, we use the matrix

$$C \equiv (A - \sigma I)^{-1}$$

in place of  $A$  and rely on the fact that

$$Cx = x\theta \iff Ax = x \left( \sigma + \frac{1}{\theta} \right)$$

to recover eigenvalues of the original matrix. The Arnoldi method is extremely effective when used in conjunction with SI. Convergence to eigenvectors corresponding to the eigenvalues of  $A$  that are nearest to  $\sigma$  is generally very rapid. Hence, this approach should be used whenever it is possible to solve linear systems efficiently with  $A - \sigma I$  as the coefficient matrix.

When a spectral transformation is used, additional considerations should be made with respect to stopping criteria to take advantage of the special nature of the transformed operator  $C$ . Moreover, the quality of the approximate eigenvectors can be improved significantly with a minor amount of postprocessing. In order to compute eigenvalues of  $A$  near to  $\sigma$  we will compute the eigenvalues  $\theta$  of  $C$  that are of largest magnitude. It is not really necessary for  $\sigma$  to be extremely close to a desired eigenvalue. However, for the following discussion it is worth keeping in mind that in practice, it is typical to take  $\sigma$  near to desired eigenvalues, and hence it is typical for  $|\theta| \gg 1$  to hold.

Let  $x = V_m y$  with  $H_m y = y\theta$ , where  $\|y\|_2 = 1$ . Since

$$(A - \sigma I)^{-1}x - x\theta = CV_m y - V_m H_m y = f_m e_m^* y, \quad (7.23)$$

it follows that  $\|f_m\| |e_m^* y|$  is the norm of the residual  $\|Cx - x\theta\|$ . If  $\|f_m\| |e_m^* y| \leq \epsilon_U$ , where  $\epsilon_U$  is a user-specified tolerance, then according to (7.17),  $x$  and  $\theta$  are an exact eigenpair for a matrix near  $C$ . However, we are really interested in eigenvalue-eigenvector approximations for the original matrix  $A$ .

A simple rearrangement of (7.23) gives

$$Ax - x \left( \sigma + \frac{1}{\theta} \right) = -(A - \sigma I)f_m \frac{e_m^* y}{\theta}, \quad (7.24)$$

and this is the residual of the computed eigenpair  $(x, \sigma + \frac{1}{\theta})$  for the matrix  $A$ . However, with a minor amount of additional arithmetic, the quality of the eigenvector can be improved and the corresponding residual norm can be reduced significantly.

Adding and subtracting  $f_m \frac{e_m^* y}{\theta^2}$  on the right-hand side of (7.24) and rearranging terms will result in

$$Az - z \left( \sigma + \frac{1}{\theta} \right) = -f_m \frac{e_m^* y}{\theta^2}, \quad (7.25)$$

where  $z \equiv x + f_m(e_m^*y/\theta)$ . In the case  $|\theta| \gg 1$ , we see that  $z$  will be a much better approximate eigenvector than  $x$ . Moreover, if  $\|A\|$  is large relative to  $|\sigma|$ , then the potential magnification of error due to the term  $(A - \sigma I)f_m$  is avoided.

A heuristic argument further supports the use of  $z$  instead of  $x$ . From (7.23) it follows that

$$(A - \sigma I)^{-1}x = x\theta + f_m e_m^* y, \quad (7.26)$$

and hence  $z = x + f_m(e_m^*y/\theta)$  is the (normalized) vector that would result if we performed one step of inverse iteration with  $A - \sigma I$  on  $x$ .

This vector  $z$  has not yet been scaled to have unit norm. However,  $\|z\|^2 = 1 + (|e_m^*y| \|f_m\| / |\theta|)^2$ , so the error bound will decrease after  $z$  is normalized. Moreover, if  $|e_m^*y| \|f_m\| / |\theta| < \sqrt{\epsilon_M}$ , then the floating point computation of the norm will already result in  $\|z\| = 1$  without any rescaling.

From (7.23), the residual  $Cx - x\theta$  is orthogonal to the Krylov space spanned by the columns of  $V_m$ . However, this Galerkin condition is lost upon transforming the computed eigenpair to the original system, regardless of whether we use  $x$  or  $z$ . This is because  $\sigma + 1/\theta$  is not the Rayleigh quotient associated with  $x$  or  $z$ . However, from (7.25)

$$\left| \frac{z^* A z}{z^* z} - \left( \sigma + \frac{1}{\theta} \right) \right| = \frac{1}{|\theta|} \left( \frac{\|f_m\| |e_m^* y|}{|\theta| \|z\|} \right)^2. \quad (7.27)$$

In other words, the approximate eigenvalue  $\sigma + 1/\theta$  is nearly a Rayleigh quotient for  $A$  when the vector  $z$  is used as the approximate eigenvector. In fact,  $\sigma + 1/\theta$  will be within roundoff level of being a Rayleigh quotient when  $\frac{|e_m^* y| \|f_m\|}{|\theta|} < \sqrt{\epsilon_M}$ .

On the other hand, from (7.24) we deduce that

$$\left| x^* A x - \left( \sigma + \frac{1}{\theta} \right) \right| = \frac{|x^* A f_m|}{\|f_m\|} \left( \frac{\|f_m\| |e_m^* y|}{|\theta|} \right) \quad (7.28)$$

is the error in using  $\sigma + 1/\theta$  as a Rayleigh quotient for  $A$  when  $x$  is used.

Equations (7.25) and (7.27) imply that the vector  $z$  is a better approximation than  $x$  to the eigenvector associated with the approximate eigenvalue  $\sigma + 1/\theta$  provided that  $|\theta|$  is greater than 1. Moreover, when  $|\theta| \gg 1$ , only a moderately small Ritz estimate is needed to achieve an acceptably small direct residual and Rayleigh quotient error. If the  $\sigma$  is near the desired eigenvalues, then these eigenvalues are mapped by  $C$  to large eigenvalues and typically  $|\theta| \gg 1$ .

The above analysis is based on that given by Ericsson and Ruhe [162] for the generalized symmetric definite eigenvalue problem.

### 7.6.9 Software Availability

While we have presented the IRAM as much as possible in template form, we do not recommend implementation from this description. High-quality software is freely available in the form of ARPACK. The fine detail of implementation is quite important to the robustness and ultimate success of this method. While a working method could be obtained from the description given here, it would almost certainly be deficient in some respect. One should regard this description as a detailed but still conceptual treatment.

ARPACK is a collection of FORTRAN subroutines designed to solve large scale eigenvalue problems. ARPACK stands for ARnoldi PKage. ARPACK software is capable of solving large scale Hermitian or non-Hermitian (standard and generalized) eigenvalue problems. It has been used for a wide range of applications. Parallel ARPACK (P\_ARPACK) is provided as an extension to the current ARPACK library and is targeted for distributed memory message passing systems. The message passing layers currently supported are BLACS and MPI.

The ARPACK software is based upon the IRAM that is presented in §7.6.2. When the matrix  $A$  is symmetric (Hermitian) this reduces to the implicitly restarted Lanczos method (IRLM) described in §4.5.

ARPACK provides various drivers in template form for SI. There are special drivers for banded matrices. It also provides a driver for computing a partial SVD of a general rectangular matrix. For complete information, one should consult the *ARPACK Users' Guide* [295].

For more information about ARPACK, including how to access it, see the book's homepage, ETHOME.

## 7.7 Block Arnoldi Method

*R. Lehoucq and K. Maschhoff*

Block methods are used for two major reasons. The first one is for reliably determining multiple and/or clustered eigenvalues. The second reason is related to issues dealing with computational efficiency. In many instances, the cost of computing a few matrix-vector products is commensurate with that of one matrix-vector product. On the other hand, two major drawbacks of block methods are the (not insignificant) added complexity of the software implementation and the comparative lack of theoretical understanding. There also remains the selection of the block size.

Although an unblocked method coupled with a deflation strategy (such as in §7.6) may be used to compute multiple and/or clustered eigenvalues, an unblocked method may prove inefficient for some eigenvalue problems because of the cost of computing the underlying subspace. Moreover, a relatively small convergence tolerance is required to reliably compute nearby eigenvalues. Many problems do not require this much accuracy, and such a criterion can result in unnecessary computation.

The simplest block method, subspace iteration, has already been discussed in §7.4. Here the block size is just equal to the subspace size. Just as Arnoldi methods are procedures for computing a basis for a sequence of power iterates (the members of the Krylov space), block Arnoldi methods string together a sequence of subspace iterates.

### 7.7.1 Block Arnoldi Reductions

Let  $A$  be a matrix of order  $n$  and  $b > 0$  be the block size. We say that

$$AV_{[m]} = V_{[m]}H_{[m]} + F_m E_m^* \quad (7.29)$$

is a block Arnoldi reduction of length  $m$  when  $V_{[m]}^*AV_{[m]} = H_{[m]}$  is a band upper Hessenberg matrix of order  $m \cdot b$ ,  $V_{[m]}^*V_{[m]} = I_{mb}$ , and  $V_{[m]}^*F_m = 0$ . For us, a band upper Hessenberg matrix is an upper triangular matrix with  $b$  subdiagonals. The columns of

$V_{[m]} = [V_1, V_2, \dots, V_m]$  are an orthogonal basis for the block Krylov subspace

$$\mathcal{K}^m(A, V_1) \equiv \text{span}\{V_1, AV_1, \dots, A^{m-1}V_1\}.$$

If  $m > \bar{m} \equiv \text{ceiling}(n/b)$ , then  $F_m = 0$  and  $H_{[m]}$  is the orthogonal reduction of  $A$  into banded upper Hessenberg form. We assume, for the moment, that  $F_m$  is of full rank and further suppose that the elements on the diagonal of  $H_{m+1,m}$  are positive. Thus, a straightforward extension of the implicit Q theorem [198, pp. 367–368] gives that  $F_m$  is (uniquely) specified by the starting block  $V_1$ . Note that if  $A = A^*$ , then  $H_{[m]}$  is a block tridiagonal matrix. Algorithm 7.11 lists an algorithm to compute a block Arnoldi reduction.

**ALGORITHM 7.11: Extending a Block Arnoldi Reduction**

- (1) compute the QR factorization  $V_{m+1}H_{m+1,m} = F_m$  using iterated CGS
- (2)  $V_{[m+1]} = [V_{[m]} \quad V_{m+1}]$
- (3)  $W = AV_{m+1}$
- (4)  $H_{m+1,m+1} = V_{m+1}^*W$
- (5)  $H_{[m+1]} = \begin{bmatrix} H_{[m]} & V_{[m]}^*W \\ H_{m+1,m}E_m^* & H_{m+1,m+1} \end{bmatrix}$
- (6)  $F_{m+1} = W - V_{[m+1]} \begin{bmatrix} V_{[m]}^*W \\ H_{m+1,m+1} \end{bmatrix}$

We now comment on some steps of Algorithm 7.11.

- (1) Here the QR factorization is computed via an iterated CGS algorithm using a possible correction step. See [96] for details and the simple test used to determine whether a correction step is necessary. One benefit of this scheme is that it allows the use of the Level 2 BLAS [133] matrix-vector multiplication subroutine xGEMV (also see §10.2). Moreover, this scheme also gives a simple way to fill out a rank-deficient  $F_m$ . For instance, if a third step of orthogonalization is needed when generating column  $j$  of  $V_{m+1}$ , then the corresponding column of  $F_m$  is linearly dependent on the previous  $j - 1$  columns of  $V_{m+1}$ . The  $j$ th diagonal element of  $H_{m+1,m}$  is set to zero, and a random unit vector is orthogonalized against  $V_{[m]}$  and the first  $j - 1$  columns of  $V_{m+1}$ .
- (3) This step allows the application of  $A$  to a group of vectors. This might prove essential when accessing  $A$  is expensive. Clearly, the goal is to amortize the cost of applying  $A$  over several vectors.
- (4) This allows the use of the Level 3 BLAS [134] matrix-matrix multiplication subroutine xGEMM for computing  $V_m^*W$ .
- (6) This is one step of block classical Gram–Schmidt (bCGR). The Level 3 BLAS [134] matrix-matrix multiplication subroutine xGEMM is used for computing the rank- $b$  update needed for the computation of  $F_{m+1}$ . To ensure the orthogonality of  $F_{m+1}$  with  $V_{m+1}$ , a second step of bCGR is performed except when  $b = 1$ . In this latter case, the simple test in DGKS [96] is used to determine whether a second orthogonalization step is needed. See [295] for details.

The scheme given for computing  $V_{[m+1]}$  is equivalent to the one proposed by Ruhe [375] (for symmetric matrices) and is the one used by the implicitly restarted block Lanczos code [24]. Although the approach in [24] cleanly deals with the problem of rank-deficient  $F_m$ , the implementation does not exploit the ability to apply  $A$  as in (3) above. Instead, as proposed in [375],  $A$  is applied to each column of  $F_m$  followed by computing the corresponding column of  $V_{[m+1]}$  and  $H_{[m+1]}$ . Our implementation further reorganizes Ruhe's approach so that the computation of the matrix of coefficients  $V_{[m]}^* W$  is separated from the QR factorization of  $F_m$ . The advantage is that steps (3)–(5) reduce the cost of I/O by a factor of the block size and increase the amount of floating point operations per memory reference.

### 7.7.2 Practical Algorithm

Algorithm 7.12 lists a generic block implicitly restarted Arnoldi method (BIRAM). The remainder of this section discusses some of the implementation issues necessary for a robust software implementation. In particular, we address the issues of block size ( $b$ ), convergence considerations, deflation, implicit restarting, and the selection of shifts and choice of  $p$ .

**ALGORITHM 7.12: BIRAM for NHEP**

- ```

for  $i = 1, 2 \dots$  until convergence
(1) extend the length  $r$  block Arnoldi reduction by  $p$  blocks:
       $AV_{[r+p]} = V_{[r+p]}H_{[r+p]} + F_{r+p}E_{r+p}^*$  by Algorithm 7.11
(2) determine whether the  $k$  Ritz values of interest are sufficiently
      approximate those of  $A$ .
(3) lock (or deflate) the Ritz values that satisfy the convergence tolerance.
(4) implicitly restart with  $p$  shifts and retain a length  $r$  block Arnoldi
      reduction.

```

Block Size. We now consider some of the issues and tradeoffs that should be considered when selecting the block size. For this discussion we assume that comparisons are made using a fixed maximum dimension for the subspace.

As the block size increases, the length of the Arnoldi reduction $m = r + p$ decreases. Since the degree of the largest power of A in the corresponding Krylov space is $m - 1$, smaller block sizes allow polynomials of larger degree to be applied. The down side to an unblocked method is that it cannot compute multiple copies of an eigenvalue of A unless the reduction already well-approximates some of the associated eigenvectors. For example, the first Ritz pair should give a residual of $O(\epsilon_M)$ or smaller relative to the norm of A before the second copy emerges.

One of the benefits of block methods is that they are more reliable for computing approximations to the clustered and/or multiple eigenvalues using a relatively large convergence criterion. Note that the block size used may be varied during each restart.

Stopping Criterion. Suppose that (s, θ) is an eigenpair of $H_{[m]}$. It follows easily from equation (7.29) that

$$AV_{[m]}s - \theta V_{[m]}s = F_m E_m^* s, \quad (7.30)$$

and so $\|AV_{[m]}s - \theta V_{[m]}s\| = \|F_m\| \|E_m^*s\| = \|H_{m+1,m}\| \|E_m^T s\|$. Thus, if the last b components of s are small relative to the size of $\|H_{m+1,m}\|$, then the Ritz pair $(z = V_{[m]}s, \theta)$ is an exact eigenpair for a matrix near A . This follows since (7.30) may be rewritten as $(A - F_m E_m^* s z^*)z = \theta z$.

The iteration in BIRAM terminates at the value of i when the k wanted eigenvalues of $H_{[m]}$ satisfy (7.30). The eigenvalues are partitioned as in (7.31) so that the wanted ones correspond to the eigenvalues of A desired.

Deflation. The main advantages of a numerically stable deflation strategy are the reduction of the working size of the reduction and the ability to determine clusters of nearby eigenvalues without requiring the block size to be greater than or equal to the size of the cluster. The deflation scheme developed for the BIRAM implementation is an extension of the techniques discussed in §7.6. We refer the reader to this section for further details.

An algorithmic issue that arises is how to handle situations where the number of converged vectors to be locked is not a multiple of the block size. At the completion of the deflation procedure, the active reduction has an incomplete last block. To simplify the implicit restarting mechanism, we fill out this last block so that the active reduction is of length $m \cdot b$.

Restarting a Block Arnoldi Reduction. In §7.6, we motivated and explained how to implicitly restart the Arnoldi method. The experimental results in [292, 290, 24] show that an implicitly restarted scheme is superior to other block methods that have appeared in the literature [391, 398]. These explicit restarting techniques select a subsequent starting block based on some criterion and then restart from scratch. Not only does BIRAM use dramatically fewer matrix-vector products; it also requires significantly fewer Arnoldi vectors to be stored. The reader is referred to [391, 227, 245, 398] for information on explicitly restarted Arnoldi methods.

The extension of the scheme in §7.6 within a block Arnoldi method is straightforward. The chief difference is that an implicitly shifted QR algorithm that retains band Hessenberg form is needed. As with the standard implicitly shifted QR algorithm, a sequence of unitary matrices are constructed and applied so that an updated band Hessenberg matrix results.

Numerous choices are possible for the selection of the p shifts used during the QR algorithm, including the specific choice of p . If the shifts are in complex conjugate pairs, the implicit double shift [198, pp. 355–358] can be used to avoid complex arithmetic.

Typically, the p shifts are selected by utilizing the spectral information contained in $H_{[r+p]}$. Partition the eigenvalues of $H_{[m]}$ so that

$$\underbrace{\{\theta_1, \dots, \theta_r\}}_{\text{wanted}} \cup \underbrace{\{\theta_{r+1}, \dots, \theta_{m \cdot b}\}}_{\text{unwanted}}. \quad (7.31)$$

For an unblocked reduction, the p shifts are selected from the unwanted eigenvalues of $H_{[m]}$ where $r = k$. Sorensen [419] proposed this as an *exact shift* strategy. This strategy is equivalent to restarting the subsequent reduction with a linear combination of the approximate Schur vectors associated with the k wanted eigenvalues. Other interesting strategies include the roots of Chebyshev polynomials [383], harmonic Ritz values [331, 337, 349, 411], the roots of Leja polynomials [23], the roots of least squares

polynomials [384], and refined shifts [244]. In particular, the Leja and harmonic Ritz values have been used to estimate the interior eigenvalues of A .

In Algorithm 7.12, the integer r is typically set to k , the number of wanted eigenvalues, during the input step. Once the iteration loop has been entered, the values of r , p , and thus $m = r + p$ may vary for every value of i . When $b > 1$, we cannot apply all $p = m \cdot b - k$ unwanted eigenvalues as shifts. We are then faced with the question of selecting which p shifts to apply and whether we should consider applying more than p shifts.

For example, m shifts can be applied until a Ritz pair satisfies the convergence tolerance. The Ritz pairs can then be deflated (or locked). (This is equivalent to the *deflated iterative Arnoldi* algorithm given by Saad [387, p. 181] and used in the implementations in [24, 398].) This approach allows us to implicitly apply a polynomial filter of the maximum degree. (Application of more than $r + p$ shifts will require applying explicit polynomials in A .) However, as more shifts are applied, the cost in computing the subsequent Arnoldi reduction increases.

A strategy that varies r , p (relative to k) and the shifts used during every iteration will give the best results. This is the subject of current research. The recent report [421] discusses an adaptive strategy for symmetric eigenvalue problems.

7.8 Lanczos Method

Z. Bai and D. Day

The non-Hermitian Lanczos method is an oblique projection method (see §3.2) for solving the NHEP,

$$Ax = \lambda x \quad \text{and} \quad y^* A = \lambda y^*, \quad (7.32)$$

where A is a non-Hermitian matrix. For complex symmetric matrices ($A = A^T$ but $A \neq A^*$), a special Lanczos method is presented §7.11.

With two starting vectors q_1 and p_1 , the Lanczos method builds a pair of biorthogonal bases for the Krylov subspaces $\mathcal{K}^j(A, q_1)$ and $\mathcal{K}^j(A^*, p_1)$, provided that the matrix-vector multiplications Az and A^*z for an arbitrary vector z are available. The inner loop uses two three-term recurrences. These recurrences use less memory and fewer memory references than the corresponding recurrences in the Arnoldi method discussed in §7.5. The Lanczos method provides approximations for both right and left eigenvectors. When estimating errors and condition numbers of the computed eigenpairs, it is crucial that both the left and right eigenvectors be available. However, there are risks of breakdown and numerical instability with the method, since it does not work with orthogonal transformations. This section defines the basic Lanczos method and its main properties, and presents algorithmic techniques that enhance the method's numerical stability and accuracy.

7.8.1 Algorithm

The non-Hermitian Lanczos method as presented in Algorithm 7.13 below is a two-sided iterative algorithm with starting vectors p_1 and q_1 . It can be viewed as biorthogonalizing, via a two-sided Gram–Schmidt procedure, the two Krylov sequences

$$\{q_1, Aq_1, A^2q_1, \dots\} \quad \text{and} \quad \{p_1, A^*p_1, (A^*)^2p_1, \dots\}.$$

The two sequences of vectors $\{q_i\}$ and $\{p_i\}$ are generated using the three-term recurrences:

$$\beta_{j+1} q_{j+1} = Aq_j - \alpha_j q_j - \gamma_j q_{j-1}, \quad (7.33)$$

$$\bar{\gamma}_{j+1} p_{j+1} = A^* p_j - \bar{\alpha}_j p_j - \bar{\beta}_j p_{j-1}. \quad (7.34)$$

The vectors $\{q_i\}$ and $\{p_i\}$ are called *Lanczos vectors*, span $\mathcal{K}^j(A, q_1)$ and $\mathcal{K}^j(A^*, p_1)$, respectively, and are biorthogonal, namely, $p_k^* q_\ell = 0$ if $k \neq \ell$ and $w_k^* v_k = 1$. In matrix notation, at the j th step, the Lanczos method generates two $n \times j$ matrices Q_j and P_j :

$$Q_j = [q_1, q_2, \dots, q_j], \quad P_j = [p_1, p_2, \dots, p_j]$$

and

$$T_j = \begin{bmatrix} \alpha_1 & \gamma_2 & & & \\ \beta_2 & \alpha_2 & \gamma_3 & & \\ & \beta_3 & \alpha_3 & \ddots & \\ & & \ddots & \ddots & \gamma_j \\ & & & \beta_j & \alpha_j \end{bmatrix}.$$

The computed quantities satisfy the governing relations, called *Lanczos factorizations*:

$$AQ_j = Q_j T_j + \beta_{j+1} q_{j+1} e_j^*, \quad (7.35)$$

$$A^* P_j = P_j T_j^* + \bar{\gamma}_{j+1} p_{j+1} e_j^*, \quad (7.36)$$

$$P_j^* Q_j = I_j. \quad (7.37)$$

In addition, $P_j^* q_{j+1} = 0$ and $p_{j+1}^* Q_j = 0$. The relation (7.37) shows that the Lanczos vectors (bases) are biorthonormal. But note that neither Q_j nor P_j is unitary. In the Lanczos bases, the matrix A is represented by a non-Hermitian tridiagonal matrix,

$$P_j^* A Q_j = T_j. \quad (7.38)$$

At any step j , we may compute eigensolutions of T_j ,

$$T_j z_i^{(j)} = \theta_i^{(j)} z_i^{(j)} \quad \text{and} \quad (w_i^{(j)})^* T_j = \theta_i^{(j)} (w_i^{(j)})^*.$$

Eigenvalues of A are approximated by the eigenvalues $\theta_i^{(j)}$ of T_j , which are called Ritz values.

To each Ritz value $\theta_i^{(j)}$ there correspond right and left Ritz vectors,

$$x_i^{(j)} = Q_j z_i^{(j)} \quad \text{and} \quad y_i^{(j)} = P_j w_i^{(j)}. \quad (7.39)$$

The convergence of the Ritz values and vectors to eigenvalues and eigenvectors of the original matrix A can be evaluated by comparing the norms of the residuals

$$r_i^{(j)} = Ax_i^{(j)} - \theta_i^{(j)} x_i^{(j)}, \quad (7.40)$$

$$(s_i^{(j)})^* = (y_i^{(j)})^* A - \theta_i^{(j)} (y_i^{(j)})^* \quad (7.41)$$

to the norms of $x_i^{(j)}$ and $y_i^{(j)}$, respectively. Moreover, by equation (7.35), the right residual vector becomes

$$r_i^{(j)} = \beta_{j+1} q_{j+1} (e_j^* z_i^{(j)}), \quad (7.42)$$

and by equation (7.36), the left residual vector becomes

$$(s_i^{(j)})^* = \gamma_{j+1} p_{j+1}^* ((w_i^{(j)})^* e_j). \quad (7.43)$$

Therefore, as in the Hermitian case (see §4.4), the residual norms are available without explicitly computing the Ritz vectors $x_i^{(j)}$ and $y_i^{(j)}$, though $\|x_i^{(j)}\|_2$ and $\|y_i^{(j)}\|_2$ are unavailable. See §7.8.2 for a more detailed discussion of this topic.

ALGORITHM 7.13: Lanczos Method for NHEP

```

(1) choose starting vectors  $q_1$  and  $p_1$  such that  $p_1^* q_1 = 1$ 
(2)  $r = Aq_1$ 
(3)  $s = A^* p_1$ 
(4) for  $j = 1, 2, \dots$ , until convergence,
    (5)  $\alpha_j = p_j^* r$ 
    (6)  $r := r - q_j \alpha_j$ 
    (7)  $s := s - p_j \bar{\alpha}_j$ 
    (8) if ( $\|r\| = 0$  and/or  $\|s\| = 0$ ), stop
    (9)  $\omega_j = r^* s$ 
    (10) if ( $\omega_j = 0$ ), stop
    (11)  $\beta_{j+1} = |\omega_j|^{1/2}$ 
    (12)  $\gamma_{j+1} = \bar{\omega}_j / \beta_{j+1}$ 
    (13)  $q_{j+1} = r / \beta_{j+1}$ 
    (14)  $p_{j+1} = s / \bar{\gamma}_{j+1}$ 
    (15) compute eigentriplets  $(\theta_i^{(j)}, z_i^{(j)}, w_i^{(j)})$  of  $T_j$ 
    (16) test for convergence
    (17) reorthogonalize if necessary
    (18)  $r = Aq_{j+1}$ 
    (19)  $s = A^* p_{j+1}$ 
    (20)  $r := r - q_j \gamma_{j+1}$ 
    (21)  $s := s - p_j \bar{\beta}_{j+1}$ 
(22) end for
(23) compute approximate eigenvectors  $x_i^{(j)}$  and  $y_i^{(j)}$ 

```

We will now comment on certain steps of Algorithm 7.13:

- (1) The initial starting vectors p_1 and q_1 are best selected by the user to embody any available knowledge concerning A 's wanted eigenvectors. In the absence of such knowledge, one can choose q_1 with randomly distributed entries and let $p_1 = q_1$.
- (2), (3), (18), (19) The matrix-vector multiplication routines to multiply A and A^* by an arbitrary vector are required in these steps. This is usually the computational bottleneck. See the discussion of convergence properties below for implementation notes in the shift-and-invert case.
- (8) This is one of two cases in which the method may break down. In fact, this is a desirable breakdown. If r is null, then the Lanczos vectors $\{q_1, q_2, \dots, q_j\}$ span a (right) invariant subspace of A ; each Ritz value and corresponding right Ritz vector is an exact eigenvalue and eigenvector of A . The Lanczos algorithm may

be continued if necessary by taking as q_{j+1} any vector such that $P_j^* q_{j+1} = 0$ and setting $\beta_{j+1} = 0$. Similar actions can be taken when s or both r and s vanish.

In practice, an exact null vector is rare. It does happen that the norms of r and/or s are tiny. A tolerance value to detect a tiny $\|r\|_2$ compared to $\|Q\|_2 \|T\|_2$ or a tiny $\|s\|_2$ compared to $\|P\|_2 \|T\|_2$ should be given. A default tolerance value is a small multiple of ϵ , the machine precision.

- (10) If $\omega_j = r^* s = 0$ before either r or s vanishes, the method has an essential breakdown. In most cases we may continue finding new vectors in the Krylov subspaces $\mathcal{K}^{j+k}(A, r)$ and $\mathcal{K}^{j+k}(A^*, s)$ for some integer $k > 0$, and add a block outside the three diagonals of T_j ; this so-called look-ahead procedure is described in [178] and implemented in QMRPACK (see §7.8.3 below). If, however, our starting vectors q_1 and p_1 have different minimal polynomials (or, say, p_1 and q_1 are composites of different set of eigenvectors), even this does not help, and we have a *mismatch*, also called *incurable breakdown*. See [354] for a further discussion. A different treatment of the breakdown is discussed in §7.9.

In practice, exact breakdown is rare. Near breakdowns occur more often; i.e., ω_j is nonzero but extremely small in absolute value. Near breakdowns cause stagnation and instability. Any criterion for detecting a near breakdown either must stop too early in some situations or stops too late in other cases. A reasonable compromise criterion for detecting near breakdowns in an eigenvalue problem is to stop if $|\omega_j| \leq \sqrt{\epsilon} \|r\|_2 \|s\|_2$.

- (15) The cost of computing eigendecomposition of the tridiagonal matrix T_j by the QR algorithm (see §7.3) is approximately $30j^3$ floating point operations per iteration, and the cumulative cost for steps 1 to j is $15j^4$ floating point operations. Solving the eigenproblem periodically, say at every 10 steps, reduces this cost.

No stable algorithm has been found for an eigenvalue problem that approximates the eigenvalues of a general tridiagonal in j^2 floating point operations, though recently conditionally stable algorithms such as [94, 189] have been proposed. No software implementation of the Lanczos algorithm known to the authors employs a fast conditionally stable eigensolver.

- (16) Computation halts once bases Q_j and P_j have been determined so that eigenvalues $\theta_i^{(j)}$ of T_j (7.38) approximate all the desired eigenvalues of A with sufficiently small residuals, which are calculated according to the equations (7.42) and (7.43). Convergence properties are discussed in more detail in §7.8.2.

If there is no re-biorthogonalization (see [105]), then in finite precision arithmetic after a Ritz value converges to an eigenvalue of A , copies of this Ritz value will appear at later Lanczos steps. For example, a cluster of Ritz values of the reduced tridiagonal matrix, T_j , may approximate a single eigenvalue of the original matrix A . A spurious value [93] is a simple Ritz value that is also an eigenvalue of the matrix of order $j - 1$ obtained by deleting the first row and column from T_j . Such spurious values should be discarded from consideration. Eigenvalues of T_j which are not spurious are identified as approximations to eigenvalues of the original matrix A and are tested for convergence.

- (17) As in the Hermitian case, in the presence of finite precision arithmetic, the biorthogonality of computed Lanczos vectors $\{q_i\}$ and $\{p_i\}$ deteriorates. One

may use the two-sided modified Gram–Schmidt (TSMGS) process [354] to reorthogonalize the $(j+1)$ st Lanczos vectors;

```

for  $i = 1, 2, \dots, j$ 
   $q_{j+1} = q_{j+1} - q_i(p_i^* q_{j+1})$ 
   $p_{j+1} = p_{j+1} - p_i(q_i^* p_{j+1})$ 
end for

```

Applying the TSMGS process at each step is very costly and becomes a computational bottleneck. In [105], an efficient alternative scheme is proposed. This topic is revisited in §7.9.

7.8.2 Convergence Properties

In this section, we discuss convergence properties in detail. First the SI is reviewed in the context of the Lanczos algorithm. Next a convergence theory of two-sided algorithms for nonsymmetric eigenvalue problems is outlined with reference to other sections of this work. Finally techniques are reviewed to cope with the unavailability of normalized residuals.

The Ritz values tend to converge first to eigenvalues at the boundary of the convex hull of the spectrum. To compute interior eigenvalues it is often worthwhile to apply a spectral transformation to A . The strategy is to accelerate the convergence rate by mapping the desired eigenvalues in the complex plane to the exterior of the transformed spectrum. The standard approach is to apply the Lanczos algorithm to the shifted and inverted matrix

$$C = (A - \sigma I)^{-1} \quad (7.44)$$

Eigenvalues in the complex plane nearest the shift σ converge first. Note that the LU factorization,

$$A - \sigma I = LU, \quad (7.45)$$

can be used to compute both $Cq_j = U^{-1}(L^{-1}q)$ and $C^*p_j = L^{-*}(U^{-*}p)$ as required in the Lanczos algorithm. See §10.3.

An inherent difficulty of a NHEP is that the known bounds on the distance from $\theta_i^{(j)}$ to an eigenvalue of A are not practical to compute [425, 256]. Another complication is that, to be of use, a convergence theory for two-sided algorithms must account for the fact that the left and right Krylov subspaces simultaneously approximate eigenvalues of A . These two problems are resolved by bounding the distance, $\|F\|$, to the nearest matrix, $A - F$, with eigentriplet $(\theta_i^{(j)}, x_i^{(j)}, y_i^{(j)})$. Eigenvalue sensitivities (condition numbers) are also available. Recall that residual vectors $r_i^{(j)}$ and $s_i^{(j)}$ as defined in (7.40) and (7.41). Next observe that the biorthogonality condition (7.37) implies that $(s_i^{(j)})^* x_i^{(j)} = (y_i^{(j)})^* r_i^{(j)} = 0$. Together these relations imply that

$$F = \frac{r_i^{(j)}(x_i^{(j)})^*}{\|x_i^{(j)}\|_2^2} + \frac{y_i^{(j)}(s_i^{(j)})^*}{\|y_i^{(j)}\|_2^2} \quad (7.46)$$

is the desired perturbation such that

$$(A - F)x_i^{(j)} = x_i^{(j)}\theta_i^{(j)}, \quad (7.47)$$

$$(y_i^{(j)})^*(A - F) = \theta_i^{(j)}(y_i^{(j)})^*, \quad (7.48)$$

and $\|F\|_F^2 = \|r_i^{(j)}\|_2^2/\|x_i^{(j)}\|_2^2 + \|s_i^{(j)}\|_2^2/\|y_i^{(j)}\|_2^2$. See §7.13 for further discussion on how to derive estimated error bounds on the accuracy of approximate eigenvalues and eigenvectors to the exact eigenvalues and eigenvectors of A .

There is a subtle yet critical complication concerning testing the Ritz vectors for convergence. The approximate eigenvectors of the original matrix A are only calculated after the test in step (16) suggests the convergence of Ritz values $\theta_i^{(j)}$ to the desired eigenvalues of A . If the Lanczos vectors are stored, then the bases Q_j and P_j are used to get the approximate eigenvectors. If the Lanczos vectors are not stored, then there are two possibilities [93]: one either reruns the three-term recurrences to generate the requested eigenvectors, or applies inverse iteration (see §7.4). The complication is that the decision whether or not to accept a Ritz vector as an approximate eigenvector cannot be based on these unnormalized residuals. After $x_i^{(j)} = Q_j z_i^{(j)}$ and $y_i^{(j)} = P_j w_i^{(j)}$ are computed, it is necessary to compare the normalized residuals

$$\frac{\|r_i^{(j)}\|_2}{\|x_i^{(j)}\|_2} \quad \text{and} \quad \frac{\|s_i^{(j)}\|_2}{\|y_i^{(j)}\|_2}$$

to user-specified tolerance. Normalized residuals can be larger in norm than the unnormalized residuals. It may be necessary to reject the computed Ritz vectors, continue the Lanczos algorithm, and then recompute the Ritz vectors. The Ritz vector $x_i^{(j)}$ is rejected if $\|Q_j z_i^{(j)}\|_2$ is significantly smaller than $\|z_i^{(j)}\|_2$. There are two common causes for this shrinkage. The first is that due to the loss of biorthogonality, $\theta_i^{(j)}$ is a spurious eigenvalue [93]. This case is addressed either by re-biorthogonalization [105] or by not computing the Ritz vectors that correspond to spurious modes [93].

The second common cause of shrinkage is best explained in terms of implementations of the Lanczos algorithm in which q_j and p_j are scaled to have unit Euclidean length [179, 354, 105]. The inner products

$$\omega_i = \frac{p_j^* q_j}{\|p_j\|_2 \|q_j\|_2}$$

bound the condition numbers of the matrices of Lanczos vectors [354]. In [29] it was shown that

$$\text{cond}(Q_j) = \|Q_j\|_2 \|Q_j^\dagger\|_2 \leq \sum_{i=1}^j |\omega_i|^{-1},$$

where Q_j^\dagger denotes the generalized inverse of Q_j and the bound also applies to $\text{cond}(P_j)$. The right Krylov subspace is nearly invariant if $\|r_j\|_2$ is tiny relative to the product of the tridiagonal norm (which is easy to estimate) and $\|Q_j\|_2$. In this scaling, the bound $\|Q_j\|_2 \leq \sqrt{j}$ simplifies the detection of invariant subspaces.

It is common for $\{|\omega_i|\}$ to decrease nearly monotonically and by many orders of magnitude over hundreds of Lanczos steps [179, 105]. And in this case the unnormalized residuals are poor estimates for Ritz values that do not appear until after $|\omega_i|$ has decreased by several orders of magnitude. More precisely, for eigenvectors $z_i^{(j)}$ with the special property that $z_i^{(j)}(1:k)$ is negligible, the lower bound

$$\|Q_j z_i^{(j)}\|_2 \geq \max_{i>k} |\omega_i| \|z_i^{(j)}\|_2 / \sqrt{j - (k+1)}$$

is likely to be nearly an equality, and if $\max_{i>k} |\omega_i|$ is extremely small, then forming $Q_j z_i^{(j)}$ is discouraged.

Multiple Eigenvalues. Multiple eigenvalues can be *defective* (having only a single eigenvector and a chain of principal vectors) or *derogatory* (having several linearly independent eigenvectors) or both. A normal (or more generally, a diagonalizable) matrix has only derogatory multiple eigenvalues. The eigenvalue problem for defective matrices is ill posed. As mentioned above in the discussion of convergence properties, perturbation (read approximation) scatters the defective eigenvalue into a cluster of poorly conditioned eigenvalues.

The Lanczos algorithm has the theoretical advantage that the characteristic polynomials of the tridiagonal matrices $\{T_j\}$ approximate the *minimal polynomial* of A [234]. This means that in exact arithmetic the Lanczos algorithm computes a complete chain of principal vectors for a defective multiple eigenvalue if run for at least as many steps as the length of the chain.

7.8.3 Software Availability

A FORTRAN implementation of the non-Hermitian Lanczos procedure with look-ahead to cure breakdowns is available in QMRPACK [180]. The corresponding routine names are DULAL for double precision real and ZULAL for double precision complex. These routines first run the look-ahead Lanczos algorithm without re-biorthogonalization for a number of steps. They will then prompt the user for the number of eigenvalues to compute. In addition, the routine will also find the common eigenvalues in two successive runs for the identification test as described at the step (16) of Algorithm 7.13.

A set of MATLAB routines for implementing the adaptive block Lanczos method (ABLE) described in §7.9 can be used to implement Algorithm 7.13 by defining the block size as one at the initial step. Optional re-biorthogonalization schemes are available. An adaptive block size scheme is used for the treatment of (near) breakdown and/or multiple or closely clustered eigenvalues of interest.

For more information about this software, including how to access it, see the book's homepage, ETHOME.

7.8.4 Notes and References

Lanczos's original 1950 paper studied the eigenvalue problem $Ax = \lambda x$ for a general matrix A [285].⁴ Cullum and Willoughby [93] demonstrated the success of the non-symmetric Lanczos method with no reorthogonalization for a number of nonsymmetric eigenproblems from applications. Special devices were proposed for solving the eigenvalue problem of the reduced tridiagonal matrices and to identify good and spurious Ritz values.

Taylor [434] and Parlett, Taylor, and Liu [364] were the first to attempt to fix the breakdown problem. The terminology "look-ahead" was coined in their papers. A new study of the look-ahead scheme in order to cure breakdown started with the work of Freund, Gutknecht, and Nachtigal [178]. Ye also proposed a scheme to cure

⁴Historical note: It appears that Lanczos did not know Krylov's work. Ostrowski informed Lanczos about the early parallel work of Krylov.

the breakdown by choosing a new starting vector to generate another Krylov subspace that includes the old one in an appropriate way [465]. The basic idea is incorporated in the ABLE method proposed in 1995 by Bai, Day, and Ye [29] (see §7.9). Theoretical studies of the breakdown phenomena and connections with the underlying polynomial theory can be found in papers by Gutknecht [213, 214] and Brezinski, Redivo Zaglia, and Sadok [65].

Since the Lanczos method is an oblique projection method, the general convergence analysis of projection methods by Saad [384] and Jia [243] is applicable, with significant limitations in practical usage.

An analogue of Paige's theory on the relationship between the loss of orthogonality among the computed Lanczos vectors and the convergence of Ritz values in the symmetric Lanczos method is extended to the nonsymmetric Lanczos method by Bai [26]. Day has published an error analysis of the nonsymmetric Lanczos process in finite precision arithmetic [104, 105]. A number of well-known results on the symmetric Lanczos method are extended to the nonsymmetric case. For example, a nonsymmetric counterpart of Simon's partial reorthogonalization scheme for symmetric Lanczos [403] is presented. A partial reorthogonalization scheme was also published recently by van der Veen and Vuik [443].

7.9 Block Lanczos Methods

Z. Bai and D. Day

The block Lanczos method, to be described in this section, is a variation of the Lanczos procedure as presented in §7.8. It uses block versions of the three-term recursions (7.34) and (7.33). As a general thrust, block algorithms substitute matrix block multiplies and block solvers for matrix-vector products and simple solvers in unblocked algorithms. In other words, higher level BLAS are used in the inner loop of the block algorithms. This decreases the I/O costs essentially by a factor of the block size. In addition, the block algorithms are generally more robust and efficient for matrices with multiple or closely clustered eigenvalues. The adaptive block method, called ABLE, presented in this section is designed to overcome possible breakdowns and to eliminate the causes of slow convergence by either the loss of biorthogonality of the computed Lanczos vectors or a block size that is smaller than the size of the cluster of desired eigenvalues. The ABLE method was first presented in [29] by Bai, Day, and Ye.

In §7.10, a band Lanczos method is presented. It is an alternative formulation of a block Lanczos method. In this variant, in contrast to the methods described in this section, the sizes of the right and left blocks of Lanczos vectors need not necessarily be the same. In particular, this method can be applied when the right and left starting blocks have different sizes, a situation that often arises in reduced-order modeling of linear dynamical systems, a topic closely related to matrix eigenvalue problems.

7.9.1 Basic Algorithm

Given an n by n matrix A and initial n by p block vectors P_1 and Q_1 such that $P_1^* Q_1 = I$, the block Lanczos method generates sequences of $n \times p$ right and left Lanczos block vectors $\{Q_i\}$ and $\{P_i\}$. These vectors are the biorthonormal bases of

the Krylov subspaces $\mathcal{K}^j(A, Q_1)$ and $\mathcal{K}^j(A^*, P_1)$. Specifically, after j steps, the Lanczos procedure determines a block-tridiagonal matrix

$$T_{[j]} = \begin{bmatrix} A_1 & B_2 & & \\ C_2 & A_2 & \ddots & \\ \ddots & \ddots & B_j & \\ & C_j & A_j & \end{bmatrix},$$

and matrices of right and left Lanczos vectors

$$Q_{[j]} = [Q_1, Q_2, \dots, Q_j] \quad \text{and} \quad P_{[j]} = [P_1, P_2, \dots, P_j]$$

that satisfy the biorthonormality condition

$$P_{[j]}^* Q_{[j]} = I. \quad (7.49)$$

The block Lanczos method uses three-term recurrences

$$Q_{j+1}C_{j+1} = AQ_j - Q_jA_j - Q_{j-1}B_j, \quad (7.50)$$

$$P_{j+1}B_{j+1}^* = A^*P_j - P_jA_j^* - P_{j-1}C_j^* \quad (7.51)$$

for $j = 1, 2, \dots$, where $Q_0 = P_0 = 0$. In matrix notation, these quantities satisfy the governing relations

$$AQ_{[j]} = Q_{[j]}T_{[j]} + Q_{j+1}C_{j+1}E_j^*, \quad (7.52)$$

$$A^*P_{[j]} = P_{[j]}T_{[j]}^* + P_{j+1}B_{j+1}^*E_j^*, \quad (7.53)$$

where E_j is a jp by p matrix of which the bottom square is an identity matrix and zeros elsewhere.

To use the Lanczos method for approximating eigenvalues and eigenvectors of A , we solve the eigenvalue problem of the $jp \times jp$ block-tridiagonal matrix $T_{[j]}$. Each eigentriplet $(\theta_i^{(j)}, z_i^{(j)}, w_i^{(j)})$ of $T_{[j]}$,

$$T_{[j]}z_i^{(j)} = \theta_i^{(j)}z_i^{(j)} \quad \text{and} \quad (w_i^{(j)})^*T_{[j]} = \theta_i^{(j)}(w_i^{(j)})^*$$

determines a Ritz triplet $(\theta_i^{(j)}, x_i^{(j)}, y_i^{(j)})$, where $x_i^{(j)} = Q_{[j]}z_i^{(j)}$ and $y_i^{(j)} = P_{[j]}w_i^{(j)}$. In general, Ritz triplets approximate outer eigentriples of A first.

To assess the approximation, let $r_i^{(j)}$ and $s_i^{(j)}$ denote the corresponding right and left residual vectors:

$$r_i^{(j)} = Ax_i^{(j)} - \theta_i^{(j)}x_i^{(j)}, \quad (7.54)$$

$$(s_i^{(j)})^* = (y_i^{(j)})^*A - \theta_i^{(j)}(y_i^{(j)})^*. \quad (7.55)$$

Substitute (7.52) and (7.53), respectively, and there appears

$$r_i^{(j)} = Q_{j+1}C_{j+1}(E_j^*z_i^{(j)}), \quad (7.56)$$

$$(s_i^{(j)})^* = ((w_i^{(j)})^*E_j)B_{j+1}P_{j+1}^*. \quad (7.57)$$

A Ritz value $\theta_i^{(j)}$ is considered to have converged if both residual norms are sufficiently small. Similarly, the residuals can also be used to determine a backward error bound for the triplet, as in the case for the standard non-Hermitian Lanczos algorithm (see §7.8).

A simple blocked version of the basic non-Hermitian Lanczos method is presented in Algorithm 7.14.

ALGORITHM 7.14: Block Lanczos Method for NHEP

```

(1)      choose  $n \times p$  starting blocks  $P_1$  and  $Q_1$  such that  $P_1^* Q_1 = I$ 
(2)       $R = A Q_1$ 
(3)       $S = A^* P_1$ 
(4)      for  $j = 1, 2, \dots$ , until convergence
        (5)           $A_j = P_j^* R$ 
        (6)           $R := R - Q_j A_j$ 
        (7)           $S := S - P_j A_j^*$ 
        (8)          QR factorize  $R = Q_{j+1} C_{j+1}$ 
        (9)          QR factorize  $S = P_{j+1} B_{j+1}^*$ 
        (10)         if  $R$  and/or  $S$  are rank deficient, stop
        (11)         compute the SVD:  $P_{j+1}^* Q_{j+1} = U_j \Sigma_j V_j^*$ 
        (12)         if  $\Sigma_j$  is (nearly) singular, stop
        (13)          $B_{j+1} := B_{j+1} U_j \Sigma_j^{1/2}$ 
        (14)          $C_{j+1} := \Sigma_j^{1/2} V_j^* C_{j+1}$ 
        (15)          $Q_{j+1} := Q_{j+1} V_j \Sigma_j^{-1/2}$ 
        (16)          $P_{j+1} := P_{j+1} U_j \Sigma_j^{-1/2}$ 
        (17)         compute eigentriplets  $(\theta_i^{(j)}, z_i^{(j)}, w_i^{(j)})$  of  $T_{[j]}$ 
        (18)         test for convergence
        (19)         reorthogonalize if necessary
        (20)          $R = A Q_{j+1}$ 
        (21)          $S = A^* P_{j+1}$ 
        (22)          $R := R - Q_j B_{j+1}$ 
        (23)          $S := S - P_j C_{j+1}^*$ 
(24)      end for
(25)      compute approximate eigenvectors  $x_i^{(j)}$  and  $y_i^{(j)}$ .

```

We now comment on some steps of Algorithm 7.14.

- (1) The starting vectors Q_1 and P_1 are best selected by the user to embody any available knowledge concerning A 's wanted eigenvectors. In the absence of such knowledge, choose Q_1 to be a real orthonormalized random $n \times p$ matrix and let $P_1 = Q_1$.

The block size p should be chosen to be larger than or equal to the multiplicity of any wanted eigenvalue. If the multiplicities are unknown, typical values of p are 2, 3, and 6.

- (2), (3), (20), (21) The user is required to furnish subroutines to calculate the products AZ and A^*Z for an arbitrary $n \times p$ matrix Z . This gives the user a chance to

calculate these products with only one pass over the data structure defining A and A^* , with possible improvements in efficiency.

(8), (9), (10) First, the QR factorizations of R^* and S are computed. If R^* and S are both of full rank, then Q_{j+1} and P_{j+1} are n by p matrices such that $Q_{j+1}^* Q_{j+1} = P_{j+1}^* P_{j+1} = I$, and both B_{j+1}^* and C_{j+1} are p by p right upper triangular matrices. If either R or S is not of full rank, then an invariant subspace has been revealed. Continuing the recurrence in the rank-deficient case is discussed in §7.9.2.

- (12) If Σ_j is singular or nearly singular, that is, if there is a zero or a very tiny singular value, then there is a *breakdown*. Proper action must be taken to continue. See §7.9.2 for possible treatments.
- (17), (18) The eigentriplets of $T_{[j]}$ can be computed using the method discussed in §7.3. The convergence of Ritz values can be tested by the norms of residuals (7.56) and (7.57). For more details on accessing the accuracy of the approximation, see §7.8.2 and §7.13 and [29].

When j becomes large, solving the eigenproblem of $T_{[j]}$ becomes time consuming. A simple way to reduce the cost is to do this periodically, say every five steps.

- (19) As in the unblocked Lanczos method, in the presence of finite precision arithmetic, the biorthogonality of the block Lanczos vectors $\{Q_i\}$ and $\{P_i\}$ deteriorates. The TSMGS process can be used to biorthogonalize Q_{j+1} and P_{j+1} in place against the previous Lanczos vectors $Q_{[j]} = [Q_1, Q_2, \dots, Q_j]$ and $P_{[j]} = [P_1, P_2, \dots, P_j]$.

```

PROCEDURE TSMGS
  for i = 1, 2, ..., j
     $Q_{j+1} := Q_{j+1} - Q_i(P_i^* Q_{j+1})$ 
     $P_{j+1} := P_{j+1} - P_i(Q_i^* P_{j+1})$ 
  end

```

- (25) This is recommended only if the biorthogonality of the Lanczos vectors is well maintained. The approximate eigenvectors $x_i^{(j)}$ and $y_i^{(j)}$ corresponding to converged Ritz values $\theta_i^{(j)}$ are computed and the residuals (7.54) and (7.55) are checked again relative to $\|x_i^{(j)}\|_2$ and $\|y_i^{(j)}\|_2$. Note that the norms can be greater than the estimates computed at step (17). This is a side-effect of the ill-conditioning of the Lanczos basis vectors.

7.9.2 An Adaptively Blocked Lanczos Method

Algorithm 7.14 breaks down if $P_{j+1}^* Q_{j+1}$ is singular. Moreover, the computed Ritz triplets can converge slowly due to either the loss of biorthogonality among the computed Lanczos vectors $Q_{[j]}$ and $P_{[j]}$ or a block size that is smaller than the size of the cluster of the desired eigenvalues. ABLE attempts to avoid breakdowns and eliminates these causes of slow convergence by adaptively changing the block size and maintaining the *full biorthogonality* of the Lanczos vectors, i.e., $\|P_{[j]}^* Q_{[j]} - I\| \approx \epsilon_M$, or *semi-biorthogonality*, i.e., $\|P_{[j]}^* Q_{[j]} - I\| \approx \sqrt{\epsilon_M}$.

Algorithm 7.15 is a pseudocode for ABLE. The method can be executed in different increasingly complex levels indicated by the following logical parameters:

fullbo uses the two-sided Gram–Schmidt process to maintain full biorthogonality at each Lanczos step.

semibo monitors the loss of biorthogonality and corrects the loss of biorthogonality at certain steps.

treatbd cures the breakdown by increasing the block size.

group adapts the block size to the order of a cluster of converged Ritz values.

There are four levels of implementation of ABLE:

Level 1: All logical flags are **false** (F).

This is the most simple block Lanczos algorithm as in Algorithm 7.14. Essentially, the three-term recurrences (7.50) and (7.51) are used and it does not store the computed Lanczos vectors. Only eigenvalues are computed. The user specifies the parameter **tolbd** for detecting breakdown. The default value for **tolbd** is $10n\epsilon_M$, where ϵ_M denotes the machine precision.

Level 2: *fullbo = true* (T) and other logical flags are **false** (F).

This version maintains full biorthogonality. Each new pair of Lanczos vectors computed by the three-term recurrences is re-biorthogonalized against all previous Lanczos vectors by TSMGS. Eigentriplets are computed for Levels ≥ 2 .

Level 3: *semibo = true* (T) and other logical flags are **false** (F).

This level attempts to maintain semi-biorthogonality. It is less expensive in both floating point operations and memory references than full biorthogonality.

Level 4: *fullbo = true* (T) or *semibo = true* (T) and *treatbd = true* (T) and/or *group = true* (T).

This version attempts to cure breakdowns by increasing the block size and/or by adapting the block size to be at least the order of any detected cluster of converged Ritz values. The user specifies the parameters **tolbd** and **tolcl** for declaring breakdown and/or grouping a cluster of Ritz values. The default values for **tolcl** and **tolbd** are $\epsilon_M^{1/2}$.

The user may select an implementation of ABLE with either full biorthogonality (Level 2) or semi-biorthogonality (Level 3), but not both. Level 4 is implemented on top of Levels 2 or 3 (full or semi-biorthogonality). The default values for the logical flags are all **true** except *fullbo = false*.

At Level 1, after a Ritz value converges to an eigenvalue of A , copies of this Ritz value appear at later Lanczos steps. For example, a cluster of Ritz values of the reduced tridiagonal matrix, $T_{[j]}$, may approximate a single eigenvalue of the original matrix A . A heuristic device to detect the spurious Ritz values is proposed in [93] and discussed in §7.8. Such phenomena are not anticipated in the higher levels of ABLE.

ALGORITHM 7.15: ABLE for NHEP

```

(1) choose  $n \times p$  starting blocks  $P_1$  and  $Q_1$  such that  $P_1^* Q_1 = I$ 
(2)  $R = A Q_1$ 
(3)  $S = A^* P_1$ 
(4) for  $j = 1, 2, \dots, j_{\max}$  until convergence,
    (5)  $A_j = P_j^* R$ 
    (6)  $R := R - Q_j A_j$ 
    (7)  $S := S - P_j A_j^*$ 
    (8) QR factorize  $R = Q_{j+1} C_{j+1}$ 
    (9) QR factorize  $S = P_{j+1} B_{j+1}^*$ 
    (10) if  $R$  or  $S$  is rank deficient and ( $fullbo = T$  or  $semibo = T$ )
        re-biorthogonalize (TSMGS)
    (11) end if
    (12) if  $\omega_j = \min(\text{diag}(\Sigma_j)) < tolbd$ ,
        breakdown := T
    (13) compute the SVD:  $P_{j+1}^* Q_{j+1} = U_j \Sigma_j V_j^*$ 
    (14) if  $\omega_j = \min(\text{diag}(\Sigma_j)) < tolbd$ ,
        breakdown := T
    (15) if  $treatbd = F$ , method fails, stop
    (16) end if
    (17) if group = T and ( $fullbo = T$  or  $semibo = T$ ),
        detect the clusters of converged Ritz values
    (18) end if
    (19) if (breakdown = T and treatbd = T) or group = T
        cure breakdown or adapt to clusters
    (20) end if
    (21) if (breakdown = T and treatbd = T) or group = T
        cure breakdown or adapt to clusters
    (22) end if
    (23) end if
    (24)  $B_{j+1} := B_{j+1} U_j \Sigma_j^{1/2}$ 
    (25)  $C_{j+1} := \Sigma_j^{1/2} V_j^* C_{j+1}$ 
    (26)  $Q_{j+1} := Q_{j+1} V_j \Sigma_j^{-1/2}$ 
    (27)  $P_{j+1} := P_{j+1} U_j \Sigma_j^{-1/2}$ 
    (28) compute eigentriplets  $(\theta_i^{(j)}, z_i^{(j)}, w_i^{(j)})$  of  $T_{[j]}$ 
    (29) test for convergence
    (30) if fullbo = T
        maintain full biorthogonality
    (31) else if semibo = T,
        monitor biorthogonality loss and correct if necessary
    (32) end if
    (33)  $R = A Q_{j+1}$ 
    (34)  $S = A^* S_{j+1}$ 
    (35)  $R := R - Q_j B_{j+1}$ 
    (36)  $S := S - P_j C_{j+1}^*$ 
    (37) end for
    (38) compute approximate eigenvectors  $x_i^{(j)}$  and  $y_i^{(j)}$ 

```

We now comment on some steps of Algorithm 7.15:

- (1) The comment on step (1) of Algorithm 7.14 applies here as well.
- (2), (3), (35), (36) The comment on steps (2), (3) and (20), (21) of Algorithm 7.14 applies here equally well.

(6), (7) The following step for maintaining local biorthogonality with the block vectors Q_j and P_j may be inserted after step (7):

$$\begin{aligned} R &:= R - Q_j(P_j^* R) \\ S &:= S - P_j(Q_j^* S) \end{aligned}$$

- (10) If R or S is rank deficient, then Q_{j+1} and P_{j+1} are biorthogonalized in place against the previous Lanczos vectors $Q_{[j]}$ and $P_{[j]}$ using TSMGS.
- (14) If the smallest singular value of $P_{j+1}^* Q_{j+1}$ is less than the prescribed tolerance value tolbd , the *breakdown* flag is switched on. δ_d denotes the number of the singular values less than tolbd . If a breakdown occurs and no treatment is specified by the user (i.e., *treatbd = false*), then the method fails.
- (18) The order, δ_c , of the largest group of clustered values from converged Ritz values $\{\theta_i^{(j)}\}$ is determined in this step. Two Ritz values $\theta_i^{(j)}$ and $\theta_k^{(j)}$ are regarded as clustered if $|\theta_i^{(j)} - \theta_k^{(j)}| \leq \text{tolcl} \cdot |\theta_i^{(j)}|$, where tolcl is the user-prescribed tolerance.
- (21) If breakdown occurs or if the order of a cluster of converged Ritz values exceeds the current block size, $\delta_c > p_j$, Q_{j+1} and P_{j+1} can be augmented:

$$Q_{j+1} := \begin{bmatrix} Q_{j+1} & \widehat{Q} \end{bmatrix}, \quad P_{j+1} := \begin{bmatrix} P_{j+1} & \widehat{P} \end{bmatrix},$$

such that $P_{j+1}^* Q_{j+1}$ is numerically nonsingular, where \widehat{Q} and \widehat{P} are $n \times \delta$, δ is determined by $\delta = \min(\max(\delta_c, \delta_d), p_{\max} - p_j)$, where p_j is the block size before argument and p_{\max} is the prescribed maximum block size. The new block size is $p_j + \delta$. TSMGS is then invoked to biorthogonalize Q_{j+1} and P_{j+1} against previous Lanczos vectors stored in $Q_{[j]}$ and $P_{[j]}$. This is the so-called adaptive blocking scheme.

No feasible method is known to select \widehat{Q} and \widehat{P} that are guaranteed to lift the smallest singular values of $P_{j+1}^* Q_{j+1}$ above tolbd , except in the case of $\text{tolbd} = 0$, i.e., the exact breakdown. One may choose \widehat{Q} and \widehat{P} as random vectors.

- (28), (29) If semi-biorthogonality is maintained, we solve the eigenproblem of $T_{[j]}$ after each re-biorthogonalization.

The formulas (7.56) and (7.57) can be used to detect convergence. Since we expect too many steps in the Lanczos iteration, at Level 1, we use the following convergence test:

$$\min\{\|r_i^{(j)}\|, \|(s_i^{(j)})^*\|\} \leq \text{tolconv} \cdot \|A\|_1,$$

At the higher levels we may use the semiquadratic convergence test:

$$\min\left\{\|r_i^{(j)}\|, \|(s_i^{(j)})^*\|, \eta_i^{(j)}\right\} \leq \text{tolconv} \cdot \|A\|_1,$$

where

$$\eta_i^{(j)} = \frac{\|r_i^{(j)}\| \|(s_i^{(j)})^*\|_2}{\text{gap}(\theta_i^{(j)}, T_{[j]})};$$

$\text{gap}(\theta_i^{(j)}, T_{[j]})$ defines the gap between $\theta_i^{(j)}$ and other Ritz values. Specifically, $\text{gap}(\theta_i^{(j)}, T_{[j]}) = \min_{k \neq i} |\theta_i^{(j)} - \theta_k^{(j)}|$. In the blocked case, the gap is defined between the clusters.

In [29] it is shown that under mild conditions, for a Ritz value $\theta_i^{(j)}$, there is an eigenvalue of A , such that $|\lambda - \theta_i^{(j)}|$ is proportional to the product of the condition number and the quantity $\eta_i^{(j)}$. However, for ill-posed problems (i.e., large condition number), small residuals (backward errors) do not imply high eigenvalue accuracy (small forward error). In this case, the above semiquadratic convergence criterion is optimistic. In any case, the right and left eigenvectors can be used to approximate eigenvalue condition numbers. This detects ill-conditioning in an eigenvalue problem (see §7.13).

- (30) If full biorthogonality is required (Level 2), simply use the TSMGS procedure (see (10)) to biorthogonalize Q_{j+1} and P_{j+1} against all previous Lanczos vectors stored in $Q_{[j]}$ and $P_{[j]}$. If semi-biorthogonality is required (Level 3), the loss of biorthogonality is monitored and, if necessary, TSMGS is invoked to maintain semi-biorthogonality.

The biorthogonality of the $(j+1)$ st Lanczos vectors Q_{j+1} and P_{j+1} to the previous Lanczos vectors is measured by the quantity

$$d_{j+1} = \max \left\{ \frac{\|P_{[j]}^* Q_{j+1}\|_1}{\|P_{[j]}\|_1 \|Q_{j+1}\|_1}, \frac{\|Q_{[j]}^* P_{j+1}\|_1}{\|Q_{[j]}\|_1 \|P_{j+1}\|_1} \right\}.$$

An efficient procedure was developed in [29] to efficiently simulate this quantity at $O(n)$ cost.

- (40) The comment to step (25) of Algorithm 7.14 also applies here.

Storage Requirements and Floating Point Operations. The following table summarizes the storage requirements for the different levels of ABLE. For clarity, we assume that the block size is a constant, p . To obtain an upper bound for the storage requirements in the variable block case, replace p by the maximum allowed block sizes, p_{\max} .

Level	Function	Storage requirement
1	basic three-term recurrences	$\text{matrix} + 6pn + 3p^2j^2 + 4pj$
2	full biorthogonality	additional $2jpn$
3	semi-biorthogonality	additional $2jpn + 4p^2j$
4	treat breakdown or cluster	no additional requirement

Note that matrix denotes the required storage for computing the product AZ and A^*Z .

To maintain full or semi-biorthogonality, it is necessary to store all computed Lanczos vectors $\{Q_i\}$ and $\{P_i\}$. The user must allocate memory for two rectangular arrays of dimension n by m for this purpose. If memory is limited, the best remedy is to restart as described in [207, 110], but this is not available in the current version of ABLE.

Now, let us summarize the cost of floating point operations performed per Lanczos step. Lower order floating point operation costs are ignored. The following table is a summary of the different types of operations required in the implementation of ABLE.

Function	Matrix-m-vectors product	m-inner product (X^*Y)	m-saxpy ($(Y + Xa)$)	m-scaling ((Xa))	QRD
Basic three-term <i>fullbo</i> <i>semibo</i>	2	6 $2j$ 2	8 $2j$	2	2

Here the m-inner product, m-saxpy, and m-scaling denote the generalizations of the corresponding Level 1 BLAS inner product, saxpy and scaling operations to multiple-vector operations. As with the storage requirements, for simplicity we assume that the block size is constant. The multiple vectors are of dimension $n \times p$. The m-inner product, m-saxpy, and m-scaling involve $2p^2n$, $2p(p+1)n$ and $2p^2n$ floating point operations, respectively. QR decomposition (QRD) of an $n \times p$ ($p \ll n$) matrix costs $2p^2n$ floating point operations. Note that for particular problems and data structures, certain operations can be performed more efficiently.

Additional floating point operations are required for the following occasional events:

- If the correction occurs at iteration j to maintain semi-biorthogonality, then this costs an additional $(4j + 2)$ m-inner products and $(4j + 2)$ m-saxpy.
- If the procedure to adapt the block size to the larger clustering of converged Ritz values (*group*) and/or cure the breakdown (*treatbd*) is executed, then the cost increases by $(2j + 5)$ m-inner products and $(2j + 4)$ m-saxpy of $n \times \delta$ multiple vectors.
- If the Lanczos vectors in a block are *rank deficient*, then the cost increases by $2j$ m-inner products and $2j$ m-saxpy for the TSMGS procedure.
- It costs about $30(jp)^3$ to compute all eigentriplets of the block-tridiagonal matrix $T_{[j]}$.

7.9.3 Software Availability

See the book's homepage, ETHOME, for a complete MATLAB implementation of the ABLE method described in this section.

7.9.4 Notes and References

A block version of the Lanczos method in the symmetric case appeared in the works of Cullum and Donath [89] and Golub and Underwood [197]. The so-called band Lanczos algorithm proposed by Ruhe is an implementation variation of the block Lanczos method [375]. The implementation of the block-symmetric Lanczos method by Grimes, Lewis, and Simon represents the state of the art [206]; this software is proprietary.

A block version of the nonsymmetric Lanczos method is straightforward, but has many pitfalls awaiting the unwary implementor. Bai studied a simple implementation of the block Lanczos method [27]. Aliaga et al. generalized Ruhe's band Lanczos implementation to the nonsymmetric case [5]. This is presented in §7.10.

The ABLE method for nonsymmetric eigenvalue problems proposed in this section was presented by Bai, Day, and Ye [29]. ABLE also implements a block version of the algorithm presented in [104, 105], to monitor the biorthogonality loss and maintain semi-biorthogonality among the computed Lanczos vectors. A so-called s -step biorthogonal Lanczos method was proposed by Kim and Chronopoulos [262] for better data locality on parallel vector computers.

7.10 Band Lanczos Method

R. Freund

The standard non-Hermitian Lanczos algorithm as presented in §7.8 uses the Krylov subspaces induced by the matrix A and a pair of single right and left starting vectors b and c , to produce approximate solutions of the NHEP,

$$Ax = \lambda x. \quad (7.58)$$

Here, A is a square, in general non-Hermitian matrix.

There are situations where the use of blocks of right and left starting vectors, instead of a pair of single starting vectors, is preferable. One such case is eigenvalue computations for matrices with multiple or closely clustered eigenvalues. Another important application is reduced-order modeling of linear dynamical systems. Here, the right and left starting blocks are given as part of the problem, as is described in more detail in §7.10.4 below. Finally, the use of blocks of starting vectors is also beneficial whenever computing matrix-matrix products AV and A^TW , where V and W are blocks of vectors, is cheaper than sequentially computing matrix-vector products Av and A^Tw for all the columns of V and W . Block Lanczos methods for blocks of equal size were discussed in §7.9.

In this section, we describe the non-Hermitian band Lanczos method, which extends the standard non-Hermitian Lanczos algorithm for single starting vectors to blocks of m right and p left starting vectors,

$$b_1, b_2, \dots, b_m \quad \text{and} \quad c_1, c_2, \dots, c_p. \quad (7.59)$$

The matrix A and the vectors (7.59) are allowed to be real or complex. However, even when they are complex, we will state the algorithm in terms of A and A^T , rather than A and A^H , since this way, we can avoid unnecessary complex conjugations in the recurrence relations used in the algorithm. We stress that both formulations are equivalent.

The matrix A and the starting vectors (7.59) induce the right block Krylov sequence

$$b_1, b_2, \dots, b_m, Ab_1, Ab_2, \dots, Ab_m, \dots, A^{k-1}b_1, \dots, A^{k-1}b_m, \dots, \quad (7.60)$$

and the left block Krylov sequence

$$c_1, c_2, \dots, c_p, A^Tc_1, A^Tc_2, \dots, A^Tc_p, \dots, (A^T)^{k-1}c_1, \dots, (A^T)^{k-1}c_p, \dots \quad (7.61)$$

The goal of the band Lanczos algorithm is to construct suitable right and left Lanczos vectors,

$$v_1, v_2, \dots, v_j \quad \text{and} \quad w_1, w_2, \dots, w_j, \quad (7.62)$$

that build bases for the subspaces spanned by the first j linearly independent vectors of the block Krylov sequences (7.60) and (7.61), respectively.

The non-Hermitian band Lanczos method discussed in this section can also be viewed as an extension of the Hermitian band Lanczos method described in §4.6 to general square non-Hermitian matrices. For the special case of right and left starting blocks of the same size, i.e., $m = p$, the band Lanczos method is also related to the block Lanczos method described in §7.9. However, the band Lanczos method is more general in that it can handle the case of arbitrary block sizes $m, p \geq 1$. Even for the special case $m = p$, there are advantages of the band method over the block Lanczos method; see §7.10.5 below.

7.10.1 Deflation

As in the case of the Hermitian band Lanczos method, the use of multiple starting vectors necessitates a suitable *deflation* procedure to delete linearly and almost linearly dependent vectors in the block Krylov sequences. We refer to §4.6.1 for a discussion of deflation in the Hermitian case.

In the non-Hermitian case, deflation in general occurs independently in the right and left block Krylov sequences. An *exact deflation* in the right block Krylov sequence (7.60) means that a vector, say $A^i b_j$, in the sequence (7.60) is linearly dependent on vectors to the left of $A^i b_j$ in (7.60) and that this vector and all its A -multiples are removed from (7.60). Similarly, an *exact deflation* in the left block Krylov sequence (7.61) means that a vector in the sequence (7.61) is linearly dependent on previous Krylov vectors in (7.61) and that this vector and all its A^T -multiples are removed from (7.61). After m exact deflations in the right block Krylov sequence (7.60) have occurred, the remaining nondeflated vectors of (7.60) span an A -invariant subspace. Furthermore, the right Lanczos vectors in (7.62) build a suitable basis for this A -invariant subspace, and all the eigenvalues of the Lanczos matrix $T_j^{(pr)}$ defined in §7.10.2 below are also eigenvalues of A . Similarly, after p exact deflations in the left block Krylov sequence (7.61) have occurred, the remaining nondeflated vectors of (7.61) span an A^T -invariant subspace. The left Lanczos vectors in (7.62) build a suitable basis for this A^T -invariant subspace and all the eigenvalues of the Lanczos matrix $T_j^{(pr)}$ are also eigenvalues of A .

Of course, in finite-precision arithmetic, it is impossible to distinguish between exactly linearly dependent and almost linearly dependent vectors. Therefore, in practice, almost linearly dependent vectors also have to be detected and deleted. In the sequel, we will refer to the process of detecting and deleting linearly dependent and almost linearly dependent vectors as *deflation*.

7.10.2 Basic Properties

After j iterations, the algorithm has generated the vectors (7.62). It will be convenient to introduce the notation

$$V_j = [v_1 \ v_2 \ \cdots \ v_j] \quad \text{and} \quad W_j = [w_1 \ w_2 \ \cdots \ w_j] \quad (7.63)$$

for the matrices whose columns are just the right and left Lanczos vectors (7.62), respectively. The vectors (7.62) are constructed to be biorthogonal. Using the notation (7.63), the biorthogonality can be stated compactly as follows:

$$W_j^T V_j = D_j = \text{diag}(\delta_1, \delta_2, \dots, \delta_j). \quad (7.64)$$

In order to enforce biorthogonality of the next Lanczos vectors, the algorithm involves division by the diagonal entries, δ_k , in (7.64). As a result, the algorithm has to be stopped as soon as

$$\delta_k = w_k^T v_k = 0, \quad \text{but} \quad v_k, w_k \neq 0, \quad (7.65)$$

occurs. The situation (7.65) is called a *breakdown* of the algorithm. While breakdowns can be remedied by incorporating so-called *look-ahead* into the algorithm, here, for simplicity, we discuss only the band Lanczos algorithm without look-ahead.

After j iterations, in addition to (7.62), the algorithm has constructed the vectors

$$\hat{v}_{j+1}, \hat{v}_{j+2}, \dots, \hat{v}_{j+m_c} \quad \text{and} \quad \hat{w}_{j+1}, \hat{w}_{j+2}, \dots, \hat{w}_{j+p_c}. \quad (7.66)$$

The vectors $\hat{v}_{j+1}, \hat{v}_{j+2}, \dots, \hat{v}_{j+m_c}$ are the candidates for the next m_c right Lanczos vectors, $v_{j+1}, v_{j+2}, \dots, v_{j+m_c}$, and the vectors $\hat{w}_{j+1}, \hat{w}_{j+2}, \dots, \hat{w}_{j+p_c}$ are the candidates for the next p_c left Lanczos vectors, $w_{j+1}, w_{j+2}, \dots, w_{j+p_c}$. Here, m_c is the number of right starting vectors, m , minus the number of deflations in the right block Krylov sequence that have occurred during the first j iterations, and p_c is the number of left starting vectors, p , minus the number of deflations in the left block Krylov sequence that have occurred during the first j iterations. The vectors (7.66) are constructed so that they satisfy the biorthogonality relations

$$\begin{aligned} W_j^T \hat{v}_k &= 0 \quad \text{for all } k = j+1, j+2, \dots, j+m_c, \\ V_j^T \hat{w}_k &= 0 \quad \text{for all } k = j+1, j+2, \dots, j+p_c. \end{aligned} \quad (7.67)$$

The algorithm has a very simple built-in deflation procedure based on the vectors (7.66). In fact, an exact deflation in the right block Krylov sequence at iteration $j+1$ is equivalent to $\hat{v}_{j+1} = 0$. Therefore, in the algorithm, one checks if $\|\hat{v}_{j+1}\|_2$ is smaller than some suitable deflation tolerance. If yes, the vector \hat{v}_{j+1} is deflated and m_c is reduced by 1. Otherwise, \hat{v}_{j+1} is normalized to become the next right Lanczos vector v_{j+1} . Similarly, an exact deflation in the left block Krylov sequence at iteration $j+1$ is equivalent to $\hat{w}_{j+1} = 0$. In the algorithm, one checks if $\|\hat{w}_{j+1}\|_2$ is smaller than the deflation tolerance. If yes, the vector \hat{w}_{j+1} is deflated and p_c is reduced by 1. Otherwise, \hat{w}_{j+1} is normalized to become the next left Lanczos vector w_{j+1} .

The recurrences that are used in the algorithm to generate the vectors (7.62) and (7.66) can be summarized compactly as follows:

$$\begin{aligned} AV_j &= V_j T_j + [\begin{array}{cccccc} 0 & \cdots & 0 & \hat{v}_{j+1} & \hat{v}_{j+2} & \cdots & \hat{v}_{j+m_c} \end{array}] + \hat{V}_j^{(dl)}, \\ A^T W_j &= W_j \tilde{T}_j + [\begin{array}{cccccc} 0 & \cdots & 0 & \hat{w}_{j+1} & \hat{w}_{j+2} & \cdots & \hat{w}_{j+p_c} \end{array}] + \hat{W}_j^{(dl)}. \end{aligned} \quad (7.68)$$

Here, T_j and \tilde{T}_j are $j \times j$ matrices whose entries are chosen so that the biorthogonality conditions (7.64) and (7.67) are satisfied. The matrix $\hat{V}_j^{(dl)}$ in (7.68) contains mostly zero columns, together with the \hat{v}_k vectors that have been deflated during the first j iterations. The matrix $\hat{W}_j^{(dl)}$ in (7.68) contains mostly zero columns, together with the \hat{w}_k vectors that have been deflated during the first j iterations. We remark that $m - m_c$ is the number of deflated \hat{v}_k vectors and that $p - p_c$ is the number of deflated \hat{w}_k vectors. It turns out that biorthogonality only has to be explicitly enforced among $m_c + p_c + 1$ consecutive Lanczos vectors and, once deflation has occurred, against

$p - p_c$ fixed earlier left Lanczos vectors, respectively, $m - m_c$ fixed earlier right Lanczos vectors. As a result, the matrices T_j and \tilde{T}_j are “essentially” banded. More precisely, T_j has lower bandwidth $m_c + 1$ and upper bandwidth $p_c + 1$, where the lower bandwidth is reduced by 1 every time a \hat{v}_k vector is deflated, and the upper bandwidth is reduced by 1 every time a \hat{w}_k vector is deflated. In addition, each deflation of a \hat{w}_k vector causes T_j to have nonzero elements in a fixed row outside and to the right of the banded part. More precisely, the row index of each such row caused by deflation of a \hat{w}_k vector is given by $k - p_c(k)$, where k is the number of the iteration at which the deflation has occurred and $p_c(k)$ is the corresponding value of p_c at that iteration. The matrix T_j can thus be written as

$$T_j = T_j^{(b)} + T_j^{(d)}, \quad (7.69)$$

where $T_j^{(b)}$ is a banded matrix and $T_j^{(d)}$ contains horizontal “spikes” above the band of T_j due to deflation of \hat{w}_k vectors. Similarly,

$$\tilde{T}_j = \tilde{T}_j^{(b)} + \tilde{T}_j^{(d)},$$

where the banded part $\tilde{T}_j^{(b)}$ has lower bandwidth $p_c + 1$ and upper bandwidth $m_c + 1$, and $\tilde{T}_j^{(d)}$ contains horizontal “spikes” above the band of \tilde{T}_j due to deflation of \hat{v}_k vectors.

The entries of the matrices T_j and \tilde{T}_j are not independent of each other. More precisely, setting

$$T_j^{(pr)} = T_j + D_j^{-1} (\tilde{T}_j^{(d)})^T D_j \quad \text{and} \quad \tilde{T}_j^{(pr)} = \tilde{T}_j + D_j^{-1} (T_j^{(d)})^T D_j, \quad (7.70)$$

we have

$$T_j^{(pr)} = D_j^{-1} (\tilde{T}_j^{(pr)})^T D_j, \quad (7.71)$$

where D_j is the diagonal matrix given by (7.64). Inserting (7.69) into the definition of $T_j^{(pr)}$ in (7.70), we obtain the relation

$$T_j^{(pr)} = T_j^{(b)} + T_j^{(d)} + D_j^{-1} (\tilde{T}_j^{(d)})^T D_j,$$

which shows that $T_j^{(pr)}$ consists of the banded part $T_j^{(b)}$, horizontal spikes due to deflation of \hat{w}_k vectors above the banded part, and vertical spikes due to deflation of \hat{v}_k vectors below the banded part.

For example, consider the case of $m = 3$ right and $p = 5$ left starting vectors. Assume that during the first $j = 15$ iterations, deflations of \hat{w}_k vectors have occurred at iterations $k = 8$, $k = 11$, and $k = 13$, and deflations of \hat{v}_k vectors have occurred at iterations $k = 7$ and $k = 12$. In this case, the matrix $T_{15}^{(pr)}$ has the following sparsity structure:

$$T_{15}^{(\text{pr})} = \begin{bmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & d \\ * & * & * & * & * & d \\ * & * & * & * & * & d \\ * & * & * & * & * & d \\ * & * & * & * & * & d \\ \tilde{d} & * & * & * & * & d \\ \tilde{d} & * & * & * & * & d \\ \tilde{d} & * & * & * & * & d \\ \tilde{d} & * & * & * & * & d \\ \tilde{d} & * & * & * & * & d \\ \tilde{d} & * & * & * & * & d \\ \tilde{d} & * & * & * & * & d \\ \tilde{d} & * & * & * & * & d \\ \tilde{d} & * & * & * & * & d \end{bmatrix}.$$

Here, the *'s denote potentially nonzero entries within the banded part, $T_{15}^{(\text{b})}$; the d's denote potentially nonzero entries due to the deflations of \widehat{w}_k vectors at iterations $k = 8$, $k = 11$, and $k = 13$; and the \tilde{d} 's denote potentially nonzero entries due to the deflations of \widehat{w}_k vectors at iterations $k = 7$ and $k = 12$. Note that the deflations have reduced the initial lower bandwidth $m + 1 = 4$ to $m_c + 1 = 2$ at iteration $j = 15$ and the initial upper bandwidth $p + 1 = 6$ to $p_c + 1 = 3$ at iteration $j = 15$.

After j iterations of the band Lanczos algorithm, approximate eigensolutions for the NHEP (7.58) are obtained via an oblique projection of the matrix A onto the subspace spanned by the columns of V_j and orthogonal to the subspace spanned by the columns of W_j . More precisely, this means that we are seeking approximate eigenvectors of (7.58) of the form $x_i^{(j)} = V_j z_i^{(j)}$ and that, after inserting this ansatz for $x_i^{(j)}$ into (7.58), we multiply the resulting relation from the left by W_j^T . This yields the generalized eigenvalue problem

$$W_j^T A V_j z_i^{(j)} = \theta_i^{(j)} W_j^T V_j z_i^{(j)}, \quad i = 1, 2, \dots, j. \quad (7.72)$$

Using the biorthogonality relations (7.64) and (7.67), it is easy to verify that the matrix $T_j^{(\text{pr})}$ defined in (7.70) satisfies

$$T_j^{(\text{pr})} = (W_j^T V_j)^{-1} W_j^T A V_j. \quad (7.73)$$

By (7.73), the generalized eigenvalue problem (7.72) is equivalent to the standard eigenvalue problem

$$T_j^{(\text{pr})} z_i^{(j)} = \theta_i^{(j)} z_i^{(j)}, \quad i = 1, 2, \dots, j.$$

We stress that, in the algorithm, we use the formula in (7.70) to obtain the entries of $T_j^{(\text{pr})}$, rather than (7.73).

The band Lanczos algorithm terminates as soon as $m_c = 0$ or $p_c = 0$ is reached. In the case $m_c = 0$, m deflations of \widehat{w}_k vectors have occurred, and thus the right block Krylov sequence (7.60) is exhausted. In the case $p_c = 0$, p deflations of \widehat{w}_k vectors have occurred, and thus the left block Krylov sequence (7.61) is exhausted.

First consider termination due to $m_c = 0$. Then, the relation for the right Lanczos vectors in (7.68) can be rewritten as follows:

$$AV_j - V_j T_j^{(\text{pr})} = P_j \widehat{V}_j^{(\text{dl})}. \quad (7.74)$$

Here, the matrix

$$P_j = I - V_j D_j^{-1} W_j^T, \quad \text{where } D_j = W_j^T V_j, \quad (7.75)$$

represents the oblique projection characterized by $P_j V_j = 0$ and $P_j x = x$ for all x in the null space of W_j^T . Now let $\theta_i^{(j)}$ and $z_i^{(j)}$ be any of the eigenpairs of $T_j^{(\text{pr})}$, and assume that $z_i^{(j)}$ is normalized so that $\|z_i^{(j)}\|_2 = 1$. Recall that the pair $\theta_i^{(j)}$ and $x_i^{(j)} = V_j z_i^{(j)}$ is used as an approximate eigensolution of A . From (7.74), it follows that the residual of this approximate eigensolution can be bounded as follows:

$$\left\| Ax_i^{(j)} - \theta_i^{(j)} x_i^{(j)} \right\|_2 = \left\| P_j \widehat{V}_j^{(\text{dl})} z_i^{(j)} \right\|_2 \leq \|P_j\|_2 \cdot \left\| \widehat{V}_j^{(\text{dl})} \right\|_2. \quad (7.76)$$

In particular, if only exact deflation is performed, then $\widehat{V}_j^{(\text{dl})} = 0$ and (7.76) shows that each eigenvalue $\theta_i^{(j)}$ of $T_j^{(\text{pr})}$ is indeed an eigenvalue of A .

Similarly, in the case of termination due to $p_c = 0$, the relation for the left Lanczos vectors in (7.68) can be rewritten as follows:

$$A^T W_j - W_j \widetilde{T}_j^{(\text{pr})} = P_j^T \widehat{W}_j^{(\text{dl})}. \quad (7.77)$$

Here, P_j is again the matrix defined in (7.75). Now let $\theta_i^{(j)}$ and $\tilde{z}_i^{(j)}$ be any of the eigenpairs of $(T_j^{(\text{pr})})^T$, and assume that $\tilde{z}_i^{(j)}$ is normalized such that $\|D_j^{-1} \tilde{z}_i^{(j)}\|_2 = 1$. Note that the complex conjugate of $\tilde{z}_i^{(j)}$ is a left eigenvector of $T_j^{(\text{pr})}$. The pair $\theta_i^{(j)}$ and $y_i^{(j)} = W_j D_j^{-1} \tilde{z}_i^{(j)}$ then represents an approximate eigensolution of A^T . From (7.77), it follows that the residual of this approximate eigensolution can be bounded as follows:

$$\left\| A^T y_i^{(j)} - \theta_i^{(j)} y_i^{(j)} \right\| = \left\| P_j^T \widehat{W}_j^{(\text{dl})} D_j^{-1} \tilde{z}_i^{(j)} \right\|_2 \leq \|P_j^T\|_2 \cdot \left\| \widehat{W}_j^{(\text{dl})} \right\|_2. \quad (7.78)$$

In particular, if only exact deflation is performed, then $\widehat{W}_j^{(\text{dl})} = 0$ and (7.78) shows that each eigenvalue $\theta_i^{(j)}$ of $T_j^{(\text{pr})}$ is indeed an eigenvalue of A^T and thus of A .

7.10.3 Algorithm

A complete statement of the band Lanczos algorithm is as follows.

ALGORITHM 7.16: Band Lanczos Method for NHEP

- (1) set $\hat{v}_k = b_k$ for $k = 1, 2, \dots, m$
 set $\hat{w}_k = c_k$ for $k = 1, 2, \dots, p$
 set $m_c = m$, $p_c = p$, and $\mathcal{I}_v = \mathcal{I}_w = \emptyset$
- (2) for $j = 1, 2, \dots$, until convergence or $m_c = 0$ or $p_c = 0$ do:
 - (3) compute $\|\hat{v}_j\|_2$
 - (4) decide if \hat{v}_j should be deflated, if yes, do the following:
 - (a) if $j - m_c > 0$, set $\mathcal{I}_w = \mathcal{I}_w \cup \{j - m_c\}$
 - (b) set $m_c = m_c - 1$. if $m_c = 0$, set $j = j - 1$ and stop
 - (c) for $k = j, j + 1, \dots, j + m_c - 1$, set $\hat{v}_k = \hat{v}_{k+1}$
 - (d) return to step (3)
 - (5) compute $\|\hat{w}_j\|_2$
 - (6) decide if \hat{w}_j should be deflated, if yes, do the following:
 - (a) if $j - p_c > 0$, set $\mathcal{I}_v = \mathcal{I}_v \cup \{j - p_c\}$
 - (b) set $p_c = p_c - 1$. if $p_c = 0$, set $j = j - 1$ and stop
 - (c) for $k = j, j + 1, \dots, j + p_c - 1$, set $\hat{w}_k = \hat{w}_{k+1}$
 - (d) return to step (5)
 - (7) set $t_{j,j-m_c} = \|\hat{v}_j\|_2$ and $\tilde{t}_{j,j-p_c} = \|\hat{w}_j\|_2$
 normalize $v_j = \hat{v}_j / t_{j,j-m_c}$ and $w_j = \hat{w}_j / \tilde{t}_{j,j-p_c}$
 - (8) compute $\delta_j = w_j^T v_j$. if $\delta_j = 0$, stop
 - (9) compute $\hat{v}_{j+m_c} = Av_j$
 - (10) set $k_v = \max\{1, j - p_c\}$
 set $\mathcal{I}_v^{(e)} = \mathcal{I}_v \cup \{k_v, k_v + 1, \dots, j - 1\}$
 for $k \in \mathcal{I}_v^{(e)}$ (in ascending order), set
 $t_{kj} = w_k^T \hat{v}_{j+m_c} / \delta_k$ and $\hat{v}_{j+m_c} = \hat{v}_{j+m_c} - v_k t_{kj}$
 - (11) compute $\hat{w}_{j+p_c} = A^T w_j$
 - (12) set $k_w = \max\{1, j - m_c\}$
 set $\mathcal{I}_w^{(e)} = \mathcal{I}_w \cup \{k_w, k_w + 1, \dots, j - 1\}$
 for $k \in \mathcal{I}_w^{(e)}$ (in ascending order), set
 $\tilde{t}_{kj} = \hat{w}_k^T v_k / \delta_k$ and $\hat{w}_{j+p_c} = \hat{w}_{j+p_c} - w_k \tilde{t}_{kj}$
 - (13) for $k = j + 1, j + 2, \dots, j + m_c$, set
 $t_{j,k-m_c} = w_j^T \hat{v}_k / \delta_j$ and $\hat{v}_k = \hat{v}_k - v_j t_{j,k-m_c}$
 - (14) for $k = j + 1, j + 2, \dots, j + p_c$, set
 $\tilde{t}_{j,k-p_c} = \hat{w}_k^T v_j / \delta_j$ and $\hat{w}_k = \hat{w}_k - w_j \tilde{t}_{j,k-p_c}$
 - (15) for $k \in \mathcal{I}_w$, set $t_{jk} = \tilde{t}_{kj} \delta_k / \delta_j$
 - (16) set $T_j^{(\text{pr})} = [t_{ik}]_{i,k=1,2,\dots,j}$
 - (17) test for convergence
 - (18) end for

We stress that the non-Hermitian band Lanczos method can be implemented by directly following the statement in Algorithm 7.16, together with the further details of some of the steps given below. To keep the length of the statement to one page, in Algorithm 7.16, all potentially nonzero entries of the banded parts of T_j and \tilde{T}_j are computed as inner products. However, only roughly half of these entries need to be

explicitly computed as inner products. The remaining entries can be obtained via the relation

$$t_{ik} = \tilde{t}_{ki} \delta_k / \delta_i, \quad (7.79)$$

which follows from the connection (7.71) of $T_j^{(\text{pr})}$ and $\tilde{T}_j^{(\text{pr})}$. In particular, in the following discussion of steps (10), (12), (13), and (14), we give formulas for the entries of T_j and \tilde{T}_j that use (7.79) whenever possible. Employing these formulas would minimize the number of inner products, but it would sacrifice some numerical stability.

Next, we discuss some of the steps of Algorithm 7.16 in more detail.

- (4) Generally, the decision if \hat{v}_j needs to be deflated should be based on checking if

$$\|\hat{v}_j\|_2 \leq \text{dtol}, \quad (7.80)$$

where dtol is a suitably chosen small deflation tolerance. The vector \hat{v}_j is then deflated if (7.80) is satisfied. In this case, the value $j - m_c$, if positive, is added to the index set \mathcal{I}_w , which contains the indices of the nonzero rows in the part $\tilde{T}_j^{(\text{d})}$ of \tilde{T}_j , and the current right block size is updated to $m_c = m_c - 1$. If $m_c = 0$, then the right block Krylov sequence (7.60) is exhausted, and the algorithm terminates naturally. If $m_c > 0$, the vector \hat{v}_j is deleted, the index $k + 1$ of each of the remaining right candidate vectors \hat{v}_{k+1} is reset to k , and, finally, the algorithm returns to step (3). If (7.80) is not satisfied, then no deflation is necessary and the algorithm proceeds with step (5).

- (6) In analogy to (7.80), the decision if \hat{w}_j needs to be deflated is based on checking if

$$\|\hat{w}_j\|_2 \leq \text{dtol}. \quad (7.81)$$

The vector \hat{w}_j is then deflated if (7.81) is satisfied. In this case, the value $j - p_c$, if positive, is added to the index set \mathcal{I}_v , which contains the indices of the nonzero rows in the part $T_j^{(\text{d})}$ of T_j , and the current left block size is updated to $p_c = p_c - 1$. If $p_c = 0$, then the left block Krylov sequence (7.61) is exhausted, and the algorithm terminates naturally. If $p_c > 0$, the vector \hat{w}_j is deleted, the index $k + 1$ of each of the remaining left candidate vectors \hat{w}_{k+1} is reset to k , and, finally, the algorithm returns to step (5). If (7.81) is not satisfied, then no deflation is necessary and the algorithm proceeds with step (7).

- (7) Both vectors \hat{v}_j and \hat{w}_j have passed the deflation check and are now normalized to become the next right and left Lanczos vectors v_j and w_j , respectively. The normalization is such that

$$\|v_j\|_2 = \|w_j\|_2 = 1 \quad \text{for all } j.$$

- (8) Here, we compute δ_j and check for breakdown. If $\delta_j = 0$, then look-ahead would be needed in order to continue the algorithm.
- (9) In this step, we advance the right block Krylov sequence by computing a new candidate vector, \hat{v}_{j+m_c} , as the A -multiple of the latest right Lanczos vector v_j .
- (10) The vector \hat{v}_{j+m_c} is biorthogonalized against the left Lanczos vectors w_k , $k \in \mathcal{I}_v^{(\text{e})}$. Note that a TSMGS procedure is used to do this biorthogonalization. One of

the inner products required in the computation of t_{kj} can be saved by employing (7.79). More precisely, the following formula for t_{kj} should be used:

$$t_{kj} = \begin{cases} \tilde{t}_{jk} \delta_j / \delta_k & \text{if } k = j - p_c, \\ w_k^T \hat{v}_{j+m_c} / \delta_k & \text{otherwise.} \end{cases} \quad (7.82)$$

Note that the necessary entry \tilde{t}_{jk} in (7.82) is available from step (7).

- (11) In this step, we advance the left block Krylov sequence by computing a new candidate vector, \hat{w}_{j+p_c} , as the A^T -multiple of the latest left Lanczos vector w_j .
- (12) The vector \hat{w}_{j+p_c} is biorthogonalized against the right Lanczos vectors v_k , $k \in \mathcal{I}_w^{(e)}$. Note that this biorthogonalization is done by means of a TSMGS procedure. Again, to save one inner product, the following formula for \tilde{t}_{kj} should be used:

$$\tilde{t}_{kj} = \begin{cases} t_{jk} \delta_j / \delta_k & \text{if } k = j - m_c, \\ \hat{w}_{j+p_c}^T v_k / \delta_k & \text{otherwise.} \end{cases}$$

- (13) The right candidate vectors, $\hat{v}_{j+1}, \hat{v}_{j+2}, \dots, \hat{v}_{j+m_c}$, are biorthogonalized against the latest left Lanczos vector w_j . To save inner products, the following formula for $t_{j,k-m_c}$ could be used:

$$t_{j,k-m_c} = \begin{cases} w_j^T \hat{v}_k / \delta_j & \text{if } k - m_c \leq 0 \text{ or } k - m_c = j, \\ \tilde{t}_{k-m_c,j} \delta_{k-m_c} / \delta_j & \text{otherwise.} \end{cases}$$

However, the use of this formula sacrifices some numerical stability.⁵

- (14) The left candidate vectors, $\hat{w}_{j+1}, \hat{w}_{j+2}, \dots, \hat{w}_{j+p_c}$, are biorthogonalized against the latest right Lanczos vector v_j . To save inner products, the following formula for $\tilde{t}_{j,k-p_c}$ could be used:

$$\tilde{t}_{j,k-p_c} = \begin{cases} \hat{w}_k^T v_j / \delta_j & \text{if } k - p_c \leq 0, \\ t_{k-p_c,j} \delta_{k-p_c} / \delta_j & \text{otherwise.} \end{cases}$$

However, the use of this formula sacrifices some numerical stability.

- (15) The entries t_{jk} computed in this step are the potentially nonzero entries in the j th row of $T_j^{(\text{pr})}$ due to the vertical spikes caused by the deflation of \hat{v}_k vectors. Note that again formula (7.79) is employed.
- (16) All the potentially nonzero elements in the j th row and the j th column of $T_j^{(\text{pr})}$ have now been computed and they are added to the matrix $T_{j-1}^{(\text{pr})}$ of the previous iteration $j - 1$, to yield the current matrix $T_j^{(\text{pr})}$. Here, we use the convention that entries t_{ik} that are not explicitly defined in Algorithm 7.16 are set to be

⁵A version of the algorithm that enhances numerical stability at the cost of some extra inner products is described in R. W. Freund, Using Krylov subspace methods with inexact deflation for reduced-order model. Bell Labs Numerical Analysis Manuscript, August 2000.

zero. We remark that in the initial iterations, i.e., as long as $j \leq m_c$, respectively $j \leq p_c$, Algorithm 7.16 also produces entries t_{jk} , respectively \tilde{t}_{jk} , with negative indices $k \leq 0$. These entries arise from the biorthogonalization of the starting vectors, and they are not part of the matrix $T_j^{(\text{pr})}$. In particular, these entries are not needed when Algorithm 7.16 is used for eigenvalue computations. However, they are crucial when Algorithm 7.16 is applied to reduced-order modeling, as we will discuss in §7.10.4 below.

- (17) For eigenvalue computations, one tests for convergence by computing the eigenvalues $\theta_i^{(j)}$, $i = 1, 2, \dots, j$, of the $j \times j$ matrix $T_j^{(\text{pr})}$. The algorithm is stopped if some of the $\theta_i^{(j)}$'s are good enough approximations to the desired eigenvalues of A . For reduced-order modeling, the algorithm is stopped if the j th-order model generated by the algorithm is a good enough approximation of the original linear dynamical system.

7.10.4 Application to Reduced-Order Modeling

For eigenvalue computations, one is usually free to choose the right and left starting vectors for the band Lanczos method. However, there are other important applications where the starting vectors are given as part of the problem. One such application is reduced-order modeling of time-invariant linear dynamical systems with multiple inputs and multiple outputs; see, e.g., [176] for a recent survey. Such systems are characterized by matrix-valued transfer functions of the form

$$H(s) = C^T (I - sA)^{-1} B, \quad s \in \mathcal{C}. \quad (7.83)$$

Here, A is a square, in general non-Hermitian matrix, and

$$B = [b_1 \ b_2 \ \cdots \ b_m] \quad \text{and} \quad C = [c_1 \ c_2 \ \cdots \ c_p] \quad (7.84)$$

are rectangular matrices with m and p columns, respectively. Moreover, m is the number of inputs, p is the number of outputs, and m and p are different in general. The band Lanczos method (applied to A and with the columns of the matrices (7.84), B and C , as right and left starting vectors) can be used to generate a reduced-order model of the linear dynamical system described by (7.83).

For reduced-order modeling, the entries t_{jk} and \tilde{t}_{jk} with negative indices $k \leq 0$ are also used. More precisely, in this case, step (16) in Algorithm 7.16 is augmented by the following six lines:

```

if  $j \leq m_c$ ,
    set  $\rho_{jk} = t_{j,k-m}$  for all  $k$  with  $j - m_c + m \leq k \leq m$ ,
    and set  $\rho = [\rho_{ik}]_{i=1,2,\dots,j, k=1,2,\dots,m}$ 
if  $j \leq p_c$ ,
    set  $\eta_{jk} = \tilde{t}_{j,k-p}$  for all  $k$  with  $j - p_c + p \leq k \leq p$ ,
    and set  $\eta = [\eta_{ik}]_{i=1,2,\dots,j, k=1,2,\dots,p}$ 

```

Here again we use the convention that entries ρ_{ik} and η_{ik} that are not explicitly defined in Algorithm 7.16 are set to be zero.

The matrix ρ is upper triangular and of size $m_1 \times m$. Here, m_1 is defined as the value of $m_c = m_c(j)$ at iteration j of Algorithm 7.16 for which $j = m_c$ is reached.

It turns out that m_1 is just the value of m minus the number of right initial vectors b_1, b_2, \dots, b_m that have been deflated. In particular, $m_1 \leq m$, and $m_1 = m$ if and only if none of the right starting vectors has been deflated. The matrix η is upper triangular and of size $p_1 \times p$. Here, p_1 is defined as the value of $p_c = p_c(j)$ at the iteration j of Algorithm 7.16 for which $j = p_c$ is reached. The number p_1 is the value of p minus the number of left initial vectors c_1, c_2, \dots, c_p that have been deflated. In particular, $p_1 \leq p$, and $p_1 = p$ if and only if none of the left starting vectors has been deflated. The entries of ρ are the coefficients used to turn the right starting vectors into the first m_1 right Lanczos vectors, and the entries of η are the coefficients used to turn the left starting vectors into the first p_1 left Lanczos vectors.

Now let $j \geq \max\{m_1, p_1\}$, and let $T_j^{(\text{pr})}$, ρ , and η be the matrices generated after j iterations of Algorithm 7.16. These three matrices then define a j th-order reduced model of the original transfer function (7.83) as follows:

$$H_j(s) = \eta_j^T D_j \left(I_j - s T_j^{(\text{pr})} \right)^{-1} \rho_j, \quad s \in \mathcal{C}. \quad (7.85)$$

Here,

$$\rho_j = \begin{bmatrix} \rho \\ 0 \end{bmatrix} \in \mathcal{C}^{j \times m} \quad \text{and} \quad \eta_j = \begin{bmatrix} \eta \\ 0 \end{bmatrix} \in \mathcal{C}^{j \times p}$$

are the matrices ρ and η with $j - m_1$, respectively $j - p_1$, rows of zeros added. The reduced-order model (7.85) can be shown to be characterized as a certain matrix-Padé approximation of the original transfer function (7.83); see [176] and the references given there.

7.10.5 Variants

The traditional approach to extending Krylov subspace techniques for single starting vectors to multiple starting vectors is to design a block version of the Krylov subspace algorithm. A block version of the non-Hermitian Lanczos process was first proposed in [260, 261]. However, this block Lanczos algorithm assumes that the right and left starting blocks have the same size, i.e., $m = p$, and it does not have deflation or look-ahead. It is easy to see that *any* block version of the non-Hermitian Lanczos process requires all right and left blocks to have the same size. In particular, any block version is restricted to the special case when $m = p$ and when possible deflations occur simultaneously in the right and left block Krylov sequences, so that $m_c = p_c$ throughout the algorithm.

The non-Hermitian band Lanczos algorithm described in this section, however, suffers from none of these restrictions. In particular, it can be used for starting blocks of arbitrary sizes $m, p \geq 1$, and it allows deflations that occur independently in the right and left block Krylov sequences.

The version of the band Lanczos method stated in Algorithm 7.16 first appeared in this form in [175]. A somewhat different version, which also includes a look-ahead procedure to remedy possible breakdowns, is described in [5].

The version of the band Lanczos method stated as Algorithm 7.16 performs all multiplications with A and A^T as matrix-vector products with single vectors. However, at any stage of the algorithm, it is also possible to precompute the next m_c matrix-vector products with A , which will be needed in the next m_c iterations, as a single

matrix-matrix product AV with a block V of m_c vectors. Indeed, instead of performing step (9) in Algorithm 7.16, one computes the matrix-matrix product

$$A [\ v_j \ \ \widehat{v}_{j+1} \ \ \widehat{v}_{j+2} \ \ \cdots \ \ \widehat{v}_{j+m_c-1} \].$$

This gives us the vector $\widehat{v}_{j+m_c} = Av_j$, which is used in the remaining steps of iteration j , as well as the vectors

$$Av_{j+1}, A\widehat{v}_{j+2}, \dots, A\widehat{v}_{j+m_c-1}. \quad (7.86)$$

In the following $m_c - 1$ iterations, we will need the vectors

$$Av_{j+1}, Av_{j+2}, \dots, Av_{j+m'_c-1}. \quad (7.87)$$

Here, m'_c is defined as m_c minus the number of deflations of \widehat{v}_k vectors that will have occurred during these following $m_c - 1$ iterations. Using the procedure outlined in §4.6.4, one can easily obtain the vectors (7.87) as suitable linear combinations of the precomputed vectors (7.86). Similarly, at any stage of the algorithm, it is possible to precompute the next p_c matrix-vector products with A^T , which will be needed in the next p_c iterations, as a single matrix-matrix product A^TW with a block W of p_c vectors. Indeed, instead of performing step (11) in Algorithm 7.16, one computes the matrix-matrix product

$$A^T [\ w_j \ \ \widehat{w}_{j+1} \ \ \widehat{w}_{j+2} \ \ \cdots \ \ \widehat{w}_{j+p_c-1} \]$$

and then proceeds analogously as for the vectors (7.86).

7.11 Lanczos Method for Complex Symmetric Eigenproblems

R. Freund

In this section, we consider the complex symmetric eigenvalue problem

$$Ax = \lambda x, \quad (7.88)$$

where $A = A^T \in \mathcal{C}^{n \times n}$.

7.11.1 Properties of Complex Symmetric Matrices

Complex symmetry is a purely algebraic property, and it has no effect on the spectrum of the matrix. Indeed, for any given set of n numbers,

$$\lambda_1, \lambda_2, \dots, \lambda_n \in \mathcal{C}, \quad (7.89)$$

there exists a complex symmetric $n \times n$ matrix A whose eigenvalues are just the prescribed numbers (7.89); see, e.g., [233, Theorem 4.4.9].

A complex symmetric matrix may not even be diagonalizable. For example, consider the complex symmetric matrix

$$A = \begin{bmatrix} 2i & 1 \\ 1 & 0 \end{bmatrix}, \quad \text{where } i = \sqrt{-1}. \quad (7.90)$$

The only eigenvalue of this matrix is $\lambda = i$, with algebraic multiplicity 2 but geometric multiplicity 1. In fact, the Jordan normal form of A is as follows:

$$Z^{-1}AZ = \begin{bmatrix} i & 1 \\ 0 & i \end{bmatrix}, \quad \text{where} \quad Z = \begin{bmatrix} i & 1 \\ 1 & 0 \end{bmatrix}.$$

Thus, A is not diagonalizable.

If a complex symmetric matrix A is diagonalizable, then it has an eigendecomposition that reflects the complex symmetry; see, e.g., [233, Theorem 4.4.13]. More precisely, a complex symmetric matrix A is diagonalizable if and only if its eigenvector matrix, Z , can be chosen such that

$$Z^T A Z = \Lambda = \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_n) \quad \text{and} \quad Z^T Z = I_n. \quad (7.91)$$

A matrix Z with n columns that satisfies $Z^T Z = I_n$ is called *complex orthogonal*. The complex orthogonality of Z in (7.91) reflects the complex symmetry of A .

We remark that the eigendecomposition (7.91) is the suitable adaptation of the corresponding decomposition for Hermitian matrices. Recall that for any matrix $A = A^*$, the eigenvector matrix Z can always be chosen to be unitary:

$$Z^* A Z = \Lambda = \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_n) \quad \text{and} \quad Z^* Z = I_n. \quad (7.92)$$

The unitarity of Z in (7.92) reflects the fact that A is Hermitian.

The reason why an eigendecomposition (7.91) does not always exist is that there are complex vectors z with

$$z^T z = 0, \quad \text{but} \quad z \neq 0. \quad (7.93)$$

Indeed, suppose A has an eigenvalue with a one-dimensional eigenspace and the vector z spanning that space satisfies (7.93). Then one of the columns of any eigenvector matrix Z of A would be of the form $z_k = \gamma z$, where $\gamma \neq 0$ is a scalar. Then, by (7.93), $z_k^T z_k = 0$, while the complex orthogonality condition, $Z^T Z = I_n$, in (7.91) would imply $z_k^T z_k = 1$. Note that for example (7.90), the vector $z = [i \ 1]^T$ spans the one-dimensional eigenspace associated with $\lambda = i$ and it satisfies (7.93).

7.11.2 Properties of the Algorithm

While the complex symmetry of A has no effect on the eigenvalues of A , this particular structure can be exploited to halve the work and storage requirements of the general non-Hermitian Lanczos method described in §7.8. Indeed, while the non-Hermitian Lanczos method involves one matrix-vector product with A and one with A^* at each iteration, the complex symmetric Lanczos method only requires one matrix-vector product with A at each iteration.

After j iterations, the complex symmetric Lanczos method has generated j Lanczos vectors,

$$v_1, v_2, \dots, v_j, \quad (7.94)$$

that span the j th Krylov subspace $\mathcal{K}^j(A, b)$ induced by the complex symmetric matrix A and any nonzero starting vector b . The vectors (7.94) are constructed to be complex orthogonal:

$$V_j^T V_j = I_j, \quad \text{where} \quad V_j = [v_1 \ v_2 \ \cdots \ v_j]. \quad (7.95)$$

Note that, in view of the eigendecomposition (7.91) of diagonalizable complex symmetric matrices A , the complex orthogonality (7.95) of the Lanczos vectors is natural.

The complex symmetric Lanczos algorithm computes the vectors (7.94) by means of three-term recurrences that can be summarized as follows:

$$AV_j = V_j T_j + [\begin{array}{cccc} 0 & \cdots & 0 & \hat{v}_{j+1} \end{array}]. \quad (7.96)$$

Here,

$$T_j = T_j^T = \begin{bmatrix} \alpha_1 & \beta_2 & & \\ \beta_2 & \alpha_2 & \ddots & \\ \ddots & \ddots & \ddots & \beta_j \\ \beta_j & \alpha_j & & \end{bmatrix} \quad (7.97)$$

is a complex symmetric tridiagonal matrix whose entries are the coefficients of the three-term recurrences. The vector \hat{v}_{j+1} is the candidate for the next Lanczos vector, v_{j+1} . It is constructed so that the orthogonality condition

$$V_j^T \hat{v}_{j+1} = 0 \quad (7.98)$$

is satisfied, and it only remains to be normalized so that $v_{j+1}^T v_{j+1} = 1$. However, it cannot be excluded that

$$\hat{v}_{j+1}^T \hat{v}_{j+1} = 0, \quad \text{but} \quad \hat{v}_{j+1} \neq 0. \quad (7.99)$$

If (7.99) occurs, then a next vector v_{j+1} cannot be obtained by simply normalizing \hat{v}_{j+1} , as it would require division by zero. Therefore, (7.99) is called a *breakdown* of the complex symmetric Lanczos algorithm. Breakdowns can be remedied by incorporating look-ahead into the algorithm. Here, for simplicity, we restrict ourselves to the complex symmetric Lanczos algorithm without look-ahead, and we simply stop the algorithm in case a breakdown (7.99) is encountered.

After j iterations of the complex symmetric Lanczos algorithm, approximate eigensolutions for the complex symmetric eigenvalue problem (7.88) are obtained by computing eigensolutions of T_j ,

$$T_j z_i^{(j)} = \theta_i^{(j)} z_i^{(j)}, \quad i = 1, 2, \dots, j. \quad (7.100)$$

Each value $\theta_i^{(j)}$ and its Ritz vector, $x_i^{(j)} = V_j z_i^{(j)}$, yield an approximate eigenpair of A . Note that T_j is the complex orthogonal projection of A onto the space spanned by the Lanczos basis matrix V_j , i.e.,

$$T_j = V_j^T A V_j. \quad (7.101)$$

Indeed, the relation follows by multiplying (7.96) from the left by V_j^T and by using the orthogonality relations (7.95) and (7.98). Of course, in the complex symmetric Lanczos algorithm, the matrix T_j is not computed via the relation (7.101). Instead, the symmetric tridiagonal structure in the definition (7.97) is exploited and only the diagonal and subdiagonal entries of T_j are explicitly generated.

It should be pointed out that V_j is complex orthogonal, but not unitary, which may have effects for the numerical stability.

7.11.3 Algorithm

A complete statement of the complex symmetric Lanczos algorithm (without look-ahead) is as follows.

ALGORITHM 7.17: Complex Symmetric Lanczos Method

- (1) set $\hat{v}_1 = b$ and $v_0 = 0$
- (2) **for** $j = 1, 2, \dots$, until convergence or $\beta_j = 0$ **do**:
- (3) **if** $\hat{v}_j = 0$, set $j = j - 1$ and **stop**
- (4) compute $\beta_j = \sqrt{\hat{v}_j^T \hat{v}_j}$
- (5) **if** $\beta_j = 0$, set $j = j - 1$ and **stop**: breakdown
- (6) normalize $v_j = \hat{v}_j / \beta_j$
- (7) compute $\hat{v}_{j+1} = Av_j$
- (8) subtract $\hat{v}_{j+1} = \hat{v}_{j+1} - v_{j-1}\beta_j$
- (9) compute $\alpha_j = v_j^T \hat{v}_{j+1}$
- (10) subtract $\hat{v}_{j+1} = \hat{v}_{j+1} - v_j\alpha_j$
- (11) test for convergence
- (12) **end for**

Next, we comment on a few of the steps of Algorithm 7.17.

- (3) If $\hat{v}_j = 0$ occurs, then the algorithm has fully exhausted the Krylov sequence generated by A and b and thus termination is natural. In fact, in this case, the Lanczos vectors generated so far span an A -invariant subspace, and all eigenvalues of the Lanczos tridiagonal matrix are also eigenvalues of A .
- (5) In practice, one also needs to stop if a so-called *near breakdown*, i.e., $\beta_j \approx 0$, occurs. A look-ahead version of the algorithm remedies both exact breakdowns, i.e., $\beta_j = 0$, and near breakdowns; see, e.g., [178, 180].
- (11) To test for convergence, the eigenvalues $\theta_i^{(j)}$, $i = 1, 2, \dots, j$, of the complex symmetric tridiagonal matrix T_j are computed, and the algorithm is stopped if some of the $\theta_i^{(j)}$'s are good enough approximations to the desired eigenvalues of A .

7.11.4 Solving the Reduced Eigenvalue Problems

Recall that Algorithm 7.17 produces a complex symmetric tridiagonal matrix T_j and that, in order to obtain approximate eigenvalues of A , at least some of the eigenvalues of T_j need to be computed. Using the QR algorithm for the task of computing all eigenvalues of T_j does not exploit the tridiagonal structure of T_j . Indeed, after one step of the QR algorithm, in general, the tridiagonal structure of T_j will have been destroyed and one obtains a full upper Hessenberg matrix, just as in the case of QR for general NHEPs. Cullum and Willoughby [91, 92, 94] developed a QL procedure for complex symmetric tridiagonal matrices that exploits this special structure. The algorithm is based on factorizations of complex symmetric tridiagonal matrices into a complex orthogonal matrix Q and a lower triangular matrix L . However, since Q is not unitary in general, carefully chosen heuristics are needed in order to monitor and

maintain numerical stability; we refer the reader to [94] for a detailed description of the QL procedure.

If the complex symmetric Lanczos method is run with look-ahead, then the complex symmetric tridiagonal structure of the Lanczos matrix T_j will be destroyed as soon as a look-ahead step occurs. The matrix V_j of Lanczos vectors is then no longer complex orthogonal, but instead $D_j = V_j^T V_j$ is a complex symmetric block-diagonal matrix, where the sizes of the diagonal blocks correspond to the lengths of the look-ahead steps. In particular, a 1×1 “block” occurs whenever a regular Lanczos step (without look-ahead) is possible, and each “true” block of size bigger than 1 corresponds to a look-ahead step. Furthermore, instead of (7.101), one now has the relation

$$D_j T_j = V_j^T A V_j,$$

which shows that $D_j T_j$ is complex symmetric. In addition, the matrix $D_j T_j$ is block tridiagonal with diagonal blocks of the same size as D_j . Therefore, in the look-ahead case, instead of the standard eigenvalue problem (7.100), one could solve the equivalent complex symmetric generalized eigenvalue problem

$$D_j T_j z_i^{(j)} = \theta_i^{(j)} D_j z_i^{(j)}, \quad i = 1, 2, \dots, j. \quad (7.102)$$

Here, both matrices $D_j T_j$ and D_j are complex symmetric, $D_j T_j$ is block tridiagonal, and D_j is block diagonal. However, it is not clear how to exploit this special structure when solving (7.102).

7.11.5 Software Availability

A FORTRAN implementation of Algorithm 7.17, even with the option to perform look-ahead if necessary, is included in the software package QMRPACK [180]. The corresponding routines in QMRPACK are CSLAL (for single precision complex) and ZSLAL (for double precision complex). The optional look-ahead in these routines is based on the look-ahead strategy described in [178].

Algorithm 7.17 can also be used for the iterative solution of complex symmetric linear systems. In [173], a QMR iterative solver based on Algorithm 7.17 is described. Some further theoretical results on complex symmetric matrices can also be found in [173]. QMRPACK also contains implementations of the complex symmetric QMR iterative solver based on Algorithm 7.17.

For more information about this software, including how to access it, see the book’s homepage, ETHOME.

7.11.6 Notes and References

A detailed discussion of basic properties of complex symmetric matrices can be found in [85] and [233, Section 4.4]. It is also possible to adapt other eigenvalue algorithms to complex symmetric matrices. For example, a Jacobi-type method for complex symmetric matrices is described in [13].

7.12 Jacobi–Davidson Methods

G. Sleijpen and H. van der Vorst

7.12.1 Generalization of Hermitian Case

Similar to the Lanczos methods and the Arnoldi method, the Jacobi–Davidson method constructs a subspace onto which the given eigenproblem is projected. The subspace is constructed with approximate shift-and-invert steps, instead of forming a Krylov subspace. In §4.7 we have explained the method in detail, and the generalization to the non-Hermitian case for the basic algorithm, described in Algorithm 4.13 (p. 91), is quite straightforward. In fact, the changes are:

1. The construction of the matrix M has to take into account that M is non-Hermitian, hence the corresponding action in (7)–(9) has to be replaced by

$$\begin{aligned} \text{for } i = 1, \dots, m-1 \\ M_{i,m} = v_i^* v_m^A, \quad M_{m,i} = v_m^* v_i^A \\ M_{m,m} = v_m^* v_m^A \end{aligned}$$

2. In (10) a routine has to be selected for the non-Hermitian dense matrix M

If the correction equation (in step (15) of the algorithm) is solved exactly, then the approximate eigenvalues have quadratic convergence towards the eigenvalues of A .

7.12.2 Schur Form and Restart

If we want to include restarts and deflation then matters become more complicated since non-Hermitian matrices do not have orthonormal eigensystems in general. Since we prefer to work with an orthonormal basis for at least the test subspace, we compute Schur forms of the reduced matrices. Instead of eigenvectors of the matrix A , we compute a partial Schur form $AQ_k = Q_kR_k$, where Q_k is an n by k orthonormal matrix and R_k is k by k upper tridiagonal. A scheme that accommodates for Schur forms is given in Algorithm 7.18. This scheme includes the possibility for restart when the dimension of the current subspace exceeds m_{\max} .

The scheme computes k_{\max} eigenvalues close to a target τ in the complex plane. Here we have to be necessarily inexact, since the eigenvalues of a general non-Hermitian matrix are not ordered in the complex plane. Which Ritz values are delivered as close eigenvalues depends, among other factors, on the angles of the corresponding eigenvectors. Usually, the selection from Ritz values is appropriate if τ is chosen somewhere at the exterior of the distribution of eigenvalues. If we want eigenvalues of M close to some interior point then the scheme may be much less satisfactory, and we propose to work in such cases with harmonic Ritz values. A scheme for interior eigenvalues will be presented in §7.12.3.

ALGORITHM 7.18: Jacobi–Davidson Method for k_{\max} Exterior Eigenvalues for NHEP

```

(1)    $t = v_0, k = 0, m = 0; Q = [], R = []$ 
(2)   while  $k < k_{\max}$ 
(3)     for  $i = 1, \dots, m$ 
(4)        $t = t - (v_i^* t)v_i$ 
(5)        $m = m + 1, v_m = t/\|t\|_2, v_m^A = Av_m$ 
(6)       for  $i = 1, \dots, m-1$ 
(7)          $M_{i,m} = v_i^* v_m^A, M_{m,i} = v_m^* v_i^A$ 
(8)          $M_{m,m} = v_m^* v_m^A$ 
(9)         Make a Schur decomposition  $M = STS^*$ , such that
            $|T_{i,i} - \tau| \leq |T_{i+1,i+1} - \tau|$ 
(10)         $u = Vs_1, u^A = V^As_1, \theta = T_{1,1}, r = u^A - \theta u, \tilde{a} = Q^*r, \tilde{r} = r - Q\tilde{a}$ 
(11)        while  $\|\tilde{r}\|_2 \leq \epsilon_M$ 
(12)           $R = \begin{bmatrix} R & \tilde{a} \\ 0 & \theta \end{bmatrix}, Q = [Q, u], k = k + 1$ 
(13)          if  $k = k_{\max}$  then stop
(14)           $m = m - 1$ 
(15)          for  $i = 1, \dots, m$ 
(16)             $v_i = Vs_{i+1}, v_i^A = V^As_{i+1}, s_i = e_i$ 
(17)             $M = \text{lower } m \text{ by } m \text{ block of } T$ 
(18)             $u = v_1, \theta = M_{1,1}, r = v_1^A - \theta u, \tilde{a} = Q^*r, \tilde{r} = r - Q\tilde{a}$ 
(19)        end while
(20)        if  $m \geq m_{\max}$  then
(21)          for  $i = 2, \dots, m_{\min}$ 
(22)             $v_i = Vs_i, v_i^A = V^As_i$ 
(23)             $M = \text{leading } m_{\min} \text{ by } m_{\min} \text{ block of } T$ 
(24)             $v_1 = u, v_1^A = u^A, m = m_{\min}$ 
(25)        end if
(26)         $\tilde{Q} = [Q, u]$ 
(27)        Solve  $t \perp \tilde{Q}$  (approximately) from:
            $(I - \tilde{Q}\tilde{Q}^*)(A - \theta I)(I - \tilde{Q}\tilde{Q}^*)t = -\tilde{r}$ 
(28)    end while

```

To apply this algorithm we need to specify a starting vector v_0 , a tolerance ϵ , a target value τ , and a number k_{\max} that specifies how many eigenpairs near τ should be computed. The value of m_{\max} denotes the maximum dimension of the search subspace. If it is exceeded, then a restart takes place with a subspace of specified dimension m_{\min} .

On completion the k_{\max} eigenvalues close to τ are delivered, and the corresponding reduced Schur form $AQ = QR$, where Q is n by k_{\max} orthogonal and R is k_{\max} by k_{\max} upper triangular. The eigenvalues are on the diagonal of R . The computed Schur form satisfies $\|Aq_j - QR e_j\|_2 \leq j\epsilon$, where q_j is the j th column of Q .

We will now discuss the components of the algorithm.

- (1) Initialization phase.
- (3)–(4) The new vector t is made orthogonal with respect to the current search subspace by means of modified Gram–Schmidt. This can be replaced, for improved

numerical stability, by the template, for iterated modified Gram–Schmidt, given in Algorithm 4.14.

If $m = 0$, then this is an empty loop.

If the angle between the new vector t and the search subspace is very small, then the resulting vector v_j has little significance and the method practically stagnates. A random vector t helps to overcome this stagnation. A more sophisticated strategy can be found in [190].

- (6)–(8) We compute the last column and row of the dense matrix $M \equiv V^*AV = V^*V^A$ (of order m); $V^A \equiv AV$. The matrix V denotes the n by m matrix with columns v_j , V^A likewise.
- (9) The Schur reduction for the m by m matrix M can be solved by a standard solver for dense eigenproblems. We have chosen to compute the standard Ritz values, which makes the algorithm suitable for computing k_{\max} exterior eigenvalues of A close to a specified τ . If eigenvalues in the interior part of the spectrum have to be computed, then the usage of harmonic Ritz values is advocated; see §7.12.3.

In each step, the Schur form has to be sorted such that $T_{1,1}$ is closest to τ . Only if $m \geq m_{\max}$ does the sorting of the Schur form have to be such that all of the m_{\max} leading diagonal elements of T are closest to τ . For ease of presentation we sorted all diagonal elements here.

For a template of an algorithm for the sorting of a Schur form, see [448, 449, 33, 171, Chap. 6B].

S is the m by m matrix with columns s_j .

- (11) The stopping criterion is to accept a Schur vector approximation as soon as the norm of the residual (for the normalized Schur vector approximation) is below ϵ . This means that we accept inaccuracies in the order of ϵ in the computed eigenvalues, and inaccuracies (in angle) in the Schur vectors of $O(\epsilon)$ (provided that the concerned eigenvalue is simple and well separated from the others). For a more quantitative analysis and proportionality constants, see §7.13.

Detection of all wanted eigenvalues cannot be guaranteed; see note (13) for Algorithm 4.17 (p. 97).

If the matrix is real, then all eigenpairs are either real or appear in complex conjugate pairs. If a complex eigenpair has been detected, then its conjugate is known and can be deflated as well. This makes the algorithm more efficient.

- (15)–(16) After acceptance of a Ritz pair, we continue the search for a next pair, with the remaining Ritz vectors as a basis for the initial search space.
- (20) We restart as soon as the dimension of the search space for the current Schur vector exceeds m_{\max} . The process is restarted with the subspace spanned by the m_{\min} Ritz vectors corresponding to the Ritz values closest to the target value τ .
- (27) We have collected the locked (computed) Schur vectors in Q , and the matrix \tilde{Q} is Q expanded with the current Schur vector approximation u . This is done in order to obtain a more compact formulation; the correction equation is equivalent to

the one in (4.50). The new correction t has to be orthogonal to the columns of \tilde{Q} as well as to u .

Of course, the correction equation can be solved by any suitable process, for instance a preconditioned Krylov subspace method that is designed to solve unsymmetric systems. Because of the occurrence of \tilde{Q} one has to be careful with the usage of preconditioners for the matrix $A - \theta I$. The inclusion of preconditioners can be achieved following the same principles as for the single vector Jacobi–Davidson algorithm; see Algorithm 4.18 for a template. Make sure that the starting vector t_0 for an iterative solver satisfies the orthogonality constraints $\tilde{Q}^* t_0 = 0$. Note that significant savings per step can be made in Algorithm 4.18 if K is kept the same for a (few) Jacobi–Davidson iterations. In that case columns of \tilde{Q} can be saved from previous steps. Also, the matrix \mathcal{M} can be updated from previous steps, along with its \mathcal{LU} decomposition.

It is not necessary to solve the correction equation very accurately. A strategy, often used for inexact Newton methods [113], here also works well: increase the accuracy with the Jacobi–Davidson iteration step, and, for instance, solve the correction equation with a residual reduction of $2^{-\ell}$ in the ℓ th Jacobi–Davidson iteration (ℓ is reset to 0 when a Schur vector is detected).

In particular, in the first few initial steps, the approximate eigenvalue θ may be very inaccurate, and then it does not make sense to solve the correction equation accurately. In this stage it can be more effective to temporarily replace θ by τ or to take $t = -r$ for the expansion of the search subspace [335, 172].

For deflation see §8.4.2, with Z_j replaced by Q_j and B by I . For the full theoretical background of this method, as well as details on the deflation technique with Schur vectors, see [172].

7.12.3 Computing Interior Eigenvalues

For restart purposes, specially if one is heading for interior eigenvalues, the harmonic Ritz vectors have been advocated for symmetric matrices [331]; see also §4.7.4.

The concept of harmonic Ritz values [349] is easily generalized for unsymmetric matrices [411]. As we have seen, the Jacobi–Davidson methods generate basis vectors v_1, \dots, v_m for a subspace \mathcal{V}_m . For the projection of A onto this subspace we have to compute the vectors $w_j \equiv Av_j$. The inverses of the Ritz values of A^{-1} , with respect to the subspace spanned by the w_j , are called the *harmonic Ritz values*. The harmonic Ritz values can be computed without inverting A , since a harmonic Ritz pair $(\tilde{\theta}_j^{(m)}, \tilde{u}_j^{(m)})$ satisfies

$$A\tilde{u}_j^{(m)} - \tilde{\theta}_j^{(m)}\tilde{u}_j^{(m)} \perp \mathcal{W}_m \equiv \text{span}\{w_1, \dots, w_m\} \quad (7.103)$$

for $\tilde{u}_j^{(m)} \in \mathcal{V}_m$ and $\tilde{u}_j^{(m)} \neq 0$. This implies that the harmonic Ritz values are the eigenvalues of the pencil $(W_m^*AV_m, W_m^*V_m)$:

$$W_m^*AV_m s_j^{(m)} - \tilde{\theta}_j^{(m)} W_m^*V_m s_j^{(m)} = 0.$$

The exterior standard Ritz values usually converge to exterior eigenvalues of A . Likewise, the interior harmonic Ritz values for the shifted matrix $A - \tau I$ usually converge towards eigenvalues $\lambda \neq \tau$ closest to the shift τ . Fortunately, the search subspaces

\mathcal{V}_m for the shifted matrix and the unshifted matrix coincide, which facilitates the computation of harmonic Ritz pairs. For reasons of stability we construct both V_m and W_m to orthonormal: W_m is such that $(A - \tau I)V_m = W_m M_m^A$, where M_m^A is m by m upper triangular.

In the resulting scheme we compute a (partial) Schur decomposition rather than a partial eigendecomposition. That is, we wish to compute vectors q_1, \dots, q_k , such that $AQ_k = Q_kR_k$, with $Q_k^*Q_k = I_k$, and R_k is a k by k upper triangular matrix. The diagonal elements of R_k represent eigenvalues of A , and the corresponding eigenvectors of A can be computed from Q_k and R_k .

An algorithm for Jacobi–Davidson, with partial reduction to Schur form and based on harmonic Ritz values and vectors, is given in Algorithm 7.19. The algorithm includes restart and deflation techniques. It can be used for the computation of a number of eigenvalues close to τ .

To apply this algorithm we need to specify a starting vector v_0 , a tolerance ϵ , a target value τ , and a number k_{\max} that specifies how many eigenpairs near τ should be computed. The value of m_{\max} denotes the maximum dimension of the search subspace. If it is exceeded, then a restart takes place with a subspace of specified dimension m_{\min} .

On completion the k_{\max} eigenvalues close to τ are delivered, and the corresponding reduced Schur form $AQ = QR$, where Q is n by k_{\max} orthogonal and R is k_{\max} by k_{\max} upper triangular. The eigenvalues are on the diagonal of R . The computed Schur form satisfies $\|Aq_j - QRe_j\|_2 \leq j\epsilon$, where q_j is the j th column of Q .

We will comment on some parts of the algorithm in view of our discussions in previous subsections.

(1) Initialization phase.

(3)–(4) The newly computed correction is made orthogonal with respect to the current search subspace by means of modified Gram–Schmidt. We have chosen to store also the matrix $V^A \equiv AV - \tau V$; v_m^A is the expansion vector for this matrix. The expansion vector for W is obtained by making v_m^A orthogonal with respect to the space of detected Schur vectors and with respect to the current test subspace by means of modified Gram–Schmidt. The Gram–Schmidt steps can be replaced, for improved numerical stability, by the template given in Algorithm 4.14.

V denotes the n by m matrix with columns v_j ; likewise W and V^A .

(8)–(9) The values $M_{i,j}$ represent elements of the square m by m matrix $M \equiv W^*V$. The values $M_{i,j}^A$ represent elements of the m by m upper triangular $M^A \equiv W^*V^A$.

(Note that $V^A = WM^A + QF$, if $F \equiv Q^*V^A$. Therefore, V^A can be reconstructed from W , M^A , Q , and F . In particular, r can be computed from these quantities. Instead of storing the n -dimensional matrix V^A , it suffices to store the k by m matrix F (of elements q_i^*w , computed in (3)–(4)). This approach saves memory space. However, for avoiding instabilities, the deflation procedure needs special attention.)

(14) At this point the QZ decomposition (generalized Schur form) of the matrix pencil (M^A, M) has to be computed: $M^A S^R = S^L T^A$ and $M S^R = S^L T$, where S^R and S^L are unitary and T^A and T are upper triangular. This can be done with a suitable routine for dense matrix pencils from LAPACK.

ALGORITHM 7.19: Jacobi–Davidson Method for k_{\max} Interior Eigenvalues for NHEP

```

(1)    $t = v_0, k = 0, m = 0; Q = [], R = []$ 
(2)   while  $k < k_{\max}$ 
(3)     for  $i = 1, \dots, m$ 
(4)        $t = t - (v_i^* t)v_i$ 
(5)        $m = m + 1, v_m = t/\|t\|_2, v_m^A = Av_m - \tau v_m, w = v_m^A$ 
(6)       for  $i = 1, \dots, k$ 
(7)          $w = w - (q_i^* w)q_i$ 
(8)       for  $i = 1, \dots, m-1$ 
(9)          $M_{i,m}^A = w_i^* w, w = w - M_{i,m}^A w_i$ 
(10)         $M_{m,m}^A = \|w\|_2, w_m = w/M_{m,m}^A$ 
(11)        for  $i = 1, \dots, m-1$ 
(12)           $M_{i,m} = w_i^* v_m, M_{m,i} = w_m^* v_i$ 
(13)           $M_{m,m} = w_m^* v_m$ 
(14)        Make a QZ decomposition  $M^A S^R = S^L T^A, M S^R = S^L T$ ,  

           such that:  $|T_{i,i}^A/T_{i,i}| \leq |T_{i+1,i+1}^A/T_{i+1,i+1}|$ 
(15)         $u = V s_1^R, u^A = V^A s_1^R, \vartheta = \overline{T_{1,1}} \cdot T_{1,1}^A$ 
(16)         $r = u^A - \vartheta u, \tilde{a} = Q^* r, \tilde{r} = r - Q \tilde{a}$ 
(17)        while  $\|\tilde{r}\|_2 \leq \epsilon$ 
(18)           $R = \begin{bmatrix} R & \tilde{a} \\ 0 & \vartheta + \tau \end{bmatrix}, Q = [Q, u], k = k + 1$ 
(19)          if  $k = k_{\max}$  then stop
(20)           $m = m - 1$ 
(21)          for  $i = 1, \dots, m$ 
(22)             $v_i = V s_{i+1}^R, v_i^A = V^A s_{i+1}^R, w_i = W s_{i+1}^L, s_i^R = s_i^L = e_i$ 
(23)             $M^A, M$  is the lower  $m$  by  $m$  block of  $T^A, T$ , resp.
(24)             $u = v_1, u^A = v_1^A, \vartheta = \overline{M_{1,1}} \cdot M_{1,1}^A$ 
(25)             $r = u^A - \vartheta u, \tilde{a} = Q^* r, \tilde{r} = r - Q \tilde{a}$ 
(26)        end while
(27)        if  $m \geq m_{\max}$  then
(28)          for  $i = 2, \dots, m_{\min}$ 
(29)             $v_i = V s_i^R, v_i^A = V^A s_i^R, w_i = W s_i^L$ 
(30)             $M^A, M$  is the leading  $m_{\min}$  by  $m_{\min}$  block of  $T^A, T$ , resp.
(31)             $v_1 = u, v_1^A = u^A, w_1 = W s_1^L, m = m_{\min}$ 
(32)        end if
(33)         $\theta = \vartheta + \tau, \tilde{Q} = [Q, u]$ 
(34)        Solve  $t \perp \tilde{Q}$  (approximately) from:  

            $(I - \tilde{Q} \tilde{Q}^*)(A - \theta I)(I - \tilde{Q} \tilde{Q}^*)t = -\tilde{r}$ 
(35)    end while

```

In each step, the Schur form has to be sorted such that $|T_{1,1}^A/T_{1,1}|$ is smallest. Only if $m \geq m_{\max}$ does the sorting of the Schur form have to be such that all of the m_{\max} leading diagonal elements of T^A and T represent the m_{\max} harmonic Ritz values closest to 0. For ease of presentation we sorted all diagonal elements here.

For an algorithm of the sorting of a generalized Schur form, see [33, 448, 449] and [171, Chap. 6B].

The matrix S^R is m by m with columns s_j^R ; likewise S^L .

- (15) The value of θ needs some attention. We have chosen to compute the Rayleigh quotient ϑ (instead of the harmonic Ritz value) corresponding to the harmonic Ritz vector u (see [412]). The Rayleigh quotient follows from the requirement that $(A - \tau I)u - \vartheta u \perp u$ instead of $\perp W$; then $r \perp u$. $\overline{T_{1,1}}$ denotes the complex conjugate of $T_{1,1}$.
- (17) The stopping criterion is to accept a Schur vector approximation as soon as the norm of the residual (for the normalized Schur vector approximation) is below ϵ . This means that we accept inaccuracies in the order of ϵ in the computed eigenvalues, and inaccuracies (in angle) in the eigenvectors of $O(\epsilon)$ (provided that the concerned eigenvalue is simple and well separated from the others and the left and right eigenvectors have a small angle).

Detection of all wanted eigenvalues cannot be guaranteed; see note (13) for Algorithm 4.17 (p. 97).

- (20) This is a restart after acceptance of an approximate eigenpair.
- (27) At this point we have a restart when the dimension of the subspace exceeds m_{\max} . After a restart the Jacobi–Davidson iterations are resumed with a subspace of dimension m_{\min} .
- (33) The deflation with computed eigenvectors is represented by the factors with Q . The matrix Q has the converged eigenvectors as its columns. If a left preconditioner K is available for the operator $A - \theta I$, then with a Krylov solver similar reductions are realizable as in the situation for exterior eigenvalues. A template for the efficient handling of the left-preconditioned operator is given in Algorithm 4.18.

7.12.4 Software Availability

There are MATLAB and FORTRAN implementations of the algorithms described in this section. For the information about this software, including how to access it, see the book's homepage, ETHOME.

7.12.5 Numerical Example

We discuss the results for a small example that can be easily repeated. The unsymmetric matrix A is of dimension $n = 100$ and it is tridiagonal. The diagonal entries are $a_{i,i} \equiv -2$, the codiagonal entries are $a_{i,i-1} \equiv 1$, $a_{i,i+1} \equiv 1.2$.

We use Algorithms 7.18 and 7.19 for the computation of the $k_{\max} = 10$ eigenvalues closest to the target value $\tau = -2 + 0.1i$. Since these eigenvalues are located in the interior of the spectrum, we expect some advantage from Algorithm 7.19, which is designed for interior eigenvalues. Indeed, as we will see, the usage of harmonic Ritz values leads to an advantage here.

We have carried out the experiments in MATLAB. The input parameters have been chosen as follows. The starting vector v has been chosen with random entries (with

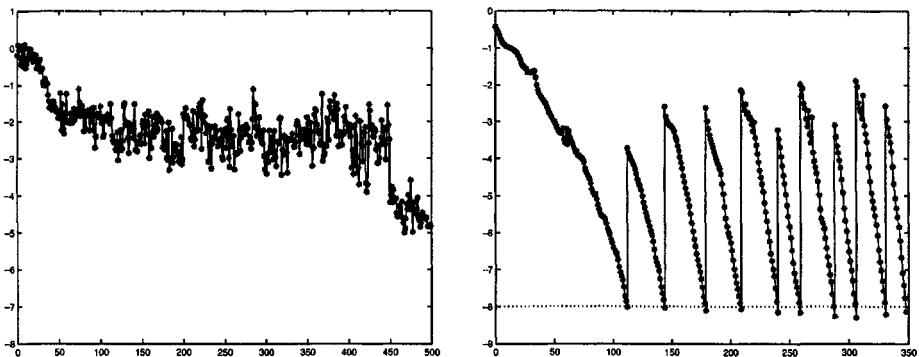


Figure 7.5: Jacobi–Davidson for exterior eigenvalues (left side) and interior eigenvalues (right side).

`seed = 0` in MATLAB). The tolerance is $\epsilon = 10^{-8}$. The subspace dimension parameters are $m_{\min} = 10$, $m_{\max} = 15$. The correction equations in (27) of Algorithm 7.18 and (34) of Algorithm 7.19 have been solved approximately with five steps of GMRES.

We show graphically in Figure 7.5 the norm of the residual vector as a function of the iteration number. Each time when the norm is less than ϵ , then we have determined an eigenvalue in acceptable approximation, and the iteration is continued with deflation for the next eigenvalue. The left picture represents the results obtained with Algorithm 7.18, and we see that there is no convergence detected within 500 Jacobi–Davidson steps. In the right picture we see the results for Algorithm 7.19, and now 10 eigenvalues have been discovered within 350 iterations.

7.13 Stability and Accuracy Assessments

Z. Bai and R. Li

In this section, we discuss the tools to assess the accuracy of computed eigenvalues and corresponding eigenvectors. We only assume the availability of residual vectors for computed eigenvalues and eigenvectors, which are usually available upon exiting a successful computation or are of marginal cost to compute afterwards for the iterative methods discussed in this chapter.

For the treatment of error estimation of the computed eigenvalues, eigenvectors, and invariant subspaces of dense NHEPs, see Chapter 4 of the *LAPACK Users’ Guide* [12].

Residual Vectors. Let $\tilde{\lambda}$ denote a computed eigenvalue of A , and let \tilde{x} be its corresponding computed eigenvector, i.e.,

$$A\tilde{x} \approx \tilde{\lambda}\tilde{x}.$$

Sometimes, its corresponding left computed eigenvector \tilde{y} is also available:

$$\tilde{y}^* A \approx \tilde{\lambda} \tilde{y}^*.$$

For simplicity, we normalize the computed eigenvectors so that $\|\tilde{x}\|_2 = 1$ and $\|\tilde{y}\|_2 = 1$. The residual vectors corresponding to the computed eigenvalue $\tilde{\lambda}$ and right and left computed eigenvectors \tilde{x} and \tilde{y} are defined as

$$r = A\tilde{x} - \tilde{\lambda}\tilde{x} \quad \text{and} \quad s^* = \tilde{y}^*A - \tilde{\lambda}\tilde{y}^*,$$

respectively. We are interested in knowing the accuracy of the computed eigenvalue $\tilde{\lambda}$ and eigenvectors \tilde{x} and \tilde{y} in terms of these residuals.

Transfer Residual Errors to Backward Errors. It turns out that the computed eigenvalue and eigenvector(s) can always be interpreted as the exact one of nearby matrices, i.e.,

$$(A + E)\tilde{x} = \tilde{\lambda}\tilde{x},$$

and

$$\tilde{y}^*(A + E) = \tilde{\lambda}\tilde{y}^*$$

if \tilde{y} is available, where the error matrices E are generally small in norm relative to $\|A\|$. Such an interpretation serves two purposes: first, it reflects indirectly how accurately the eigenproblem has been solved; and second, it can be used to derive error bounds for the computed eigenvalues and eigenvectors to be discussed below. Ideally, we would like E to be zero matrices, but this hardly ever happens at all in practice. There are infinitely many error matrices E that satisfy the above equations, we would like to know only the optimal or nearly optimal error matrices in the sense that certain norms (usually the 2-norm $\|\cdot\|_2$ or the Frobenius norm $\|\cdot\|_F$) are minimized among all feasible error matrices. In fact, practical purposes will be served if we can determine upper bounds for the norms of these (nearly) optimal matrices. The following collection of results indeed shows that *if r (and s^* if available) is small, the error matrix E is small, too* [425].

We distinguish two cases.

1. Only \tilde{x} is available but \tilde{y} is not. Then the optimal error matrix E (in both 2-norm and the Frobenius norm) for which $\tilde{\lambda}$ and \tilde{x} are an exact eigenvalue and its corresponding eigenvector of $A + E$, i.e.,

$$(A + E)\tilde{x} = \tilde{\lambda}\tilde{x}, \tag{7.104}$$

satisfies

$$\|E\|_2 = \|E\|_F = \|r\|_2.$$

2. Both \tilde{x} and \tilde{y} are available. Then the optimal error matrices E_2 (in 2-norm) and E_F (in the Frobenius norm) for which $\tilde{\lambda}$, \tilde{x} , and \tilde{y} are an exact eigenvalue and its corresponding eigenvectors of $A + E$, i.e.,

$$(A + E)\tilde{x} = \tilde{\lambda}\tilde{x} \quad \text{and} \quad \tilde{y}^*(A + E) = \tilde{\lambda}\tilde{y}^* \tag{7.105}$$

for $E = E_2, E_F$, satisfy

$$\|E_2\|_2 = \max\{\|r\|_2, \|s^*\|_2\}$$

and

$$\|E_F\|_F = \sqrt{\|r\|_2^2 + \|s^*\|_2^2 - (\tilde{y}^*A\tilde{x} - \tilde{\lambda}\tilde{y}^*\tilde{x})^2}.$$

See [256, 431].

We say the algorithm that delivers the approximate eigenpair $(\tilde{\lambda}, \tilde{x})$ is τ -backward stable for the pair with respect to the norm $\|\cdot\|$ if it is an exact eigenpair for $A + E$ with $\|E\| \leq \tau$; analogously the algorithm that delivers the eigentriplet $(\tilde{\lambda}, \tilde{x}, \tilde{y})$ is τ -backward stable for the triplet with respect to the norm $\|\cdot\|$ if it is an exact eigentriplet for $A + E$ with $\|E\| \leq \tau$. With these in mind, statements can be made about the backward stability of the algorithm which computes the eigenpair $(\tilde{\lambda}, \tilde{x})$ or the eigentriplet $(\tilde{\lambda}, \tilde{x}, \tilde{y})$. Conventionally, an algorithm is called *backward stable* if $\tau = O(\epsilon_M \|A\|)$.

Error Bound for Computed Eigenvalues. From (7.105) and the perturbation theory of eigenvalues and eigenvectors, it can be shown that up to the first order of residual norms $\|r\|_2$ and $\|s^*\|_2$, there is an eigenvalue λ of A satisfying

$$|\lambda - \tilde{\lambda}| \lesssim \frac{1}{|\tilde{y}^* \tilde{x}|} \max \{ \|r\|_2, \|s^*\|_2 \}, \quad (7.106)$$

where " \lesssim " denotes "less than" up to the first order of $\|r\|_2$ and $\|s^*\|_2$. Note that the *individual condition number* c_λ is defined as

$$c_\lambda \equiv \frac{1}{|y^* x|} = \sec \theta(x, y),$$

and x and y are the corresponding eigenvector and left eigenvector of A and are normalized so that $\|x\|_2 = \|y\|_2 = 1$. Since exact x and y usually are not known, for all practical purposes, one could simply estimate this condition number c_λ by $1/|\tilde{y}^* \tilde{x}|$ instead. Notice that $c_\lambda \geq 1$, with equality if A is Hermitian.

If the approximate left eigenvector \tilde{y} is not available, bounds directly on $|\lambda - \tilde{\lambda}|$ do not exist in general. The equation (7.104) would be the only thing available to explain the accuracy of the computed eigenvalues $\tilde{\lambda}$ and \tilde{x} .

Error Bound for Computed Eigenvectors. Instead of presenting available detailed error bounds of computed eigenvectors, which can be complicated and difficult to digest for large sparse eigenproblems, we shall give a brief actual analysis which will bring out the gist of what may contribute to the sensitivity of eigenvectors.

Let us consider the case of having an approximate eigenpair $(\tilde{\lambda}, \tilde{x})$ to an exact pair (λ, x) . Let $A = QTQ^*$ be the Schur decomposition of A (see §2.5, p. 23), where $Q = [x, Q_2]$ is unitary and

$$T = \begin{bmatrix} \lambda & a^* \\ 0 & T_{22} \end{bmatrix}.$$

Then from the residual vector $r = A\tilde{x} - \tilde{\lambda}\tilde{x} = QTQ^*\tilde{x} - \tilde{\lambda}\tilde{x}$, we have

$$Q^*r = TQ^*\tilde{x} - \tilde{\lambda}Q^*\tilde{x} = (T - \tilde{\lambda}I)Q^*\tilde{x}.$$

So the second to the last components of the above equation imply

$$\|r\|_2 = \|Q^*r\|_2 \geq \|(T_{22} - \tilde{\lambda}I)Q_2^*\tilde{x}\|_2 \geq \|(T_{22} - \tilde{\lambda}I)^{-1}\|_2^{-1} \|Q_2^*\tilde{x}\|_2,$$

which shows

$$\sin \theta(x, \tilde{x}) \equiv \|Q_2^*\tilde{x}\|_2 \leq \|(T_{22} - \tilde{\lambda}I)^{-1}\|_2 \|r\|_2 = \frac{\|r\|_2}{\sigma_{\min}(T_{22} - \tilde{\lambda}I)}. \quad (7.107)$$

This inequality reveals another potential factor that may make $\sin \theta(x, \tilde{x})$ large. This occurs when T_{22} is too far from the set of normal matrices, in which case $\|(T_{22} - \tilde{\lambda}I)^{-1}\|_2$, or the reciprocal of the smallest singular value of $T_{22} - \tilde{\lambda}I$, can get huge even though none of the rest of A 's eigenvalues come close to $\tilde{\lambda}$.

For the small and dense eigenproblems, $\sigma_{\min}(T_{22} - \tilde{\lambda}I)$ can be efficiently estimated. This is available in LAPACK [12]. However, for large sparse eigenproblems, since T_{22} is generally not available, the estimation of $\sigma_{\min}(T_{22} - \tilde{\lambda}I)$ is out of the question. We can only get a gross sense about the quality of computed eigenvectors.

Note that $\sigma_{\min}(T_{22} - \tilde{\lambda}I)$ roughly measures the separation of λ from the eigenvalues of T_{22} . We have to say “roughly” because

$$\sigma_{\min}(T_{22} - \tilde{\lambda}I) \leq \min_{\mu \in \lambda(T_{22})} |\mu - \tilde{\lambda}|$$

and the upper bound can be a gross overestimate.

As summarized in [198], the separation of the eigenvalues has a bearing upon eigenvector sensitivity. Indeed, if λ is a nondefective, repeated eigenvalue, then there are an infinite number of possible eigenvector bases for the associated invariant subspace. The preceding analysis merely indicates that this indeterminacy begins to be felt as the eigenvalues coalesce. It is well known that each individual eigenvector associated with eigenvalue clusters is very sensitive to perturbations, and consequently such an eigenvector cannot be accurately computed in general. Fortunately, often for the case of an eigenvalue cluster, it is the entire associated eigenspace that is of practical importance, and that eigenspace can be computed with satisfactory accuracy. Detailed treatment is beyond the scope of this book and the interested reader is referred to the literature, for example, [425].

This page intentionally left blank

Chapter 8

Generalized Non-Hermitian Eigenvalue Problems

8.1 Introduction

This chapter is devoted to the numerical solution of the (*right*) generalized non-Hermitian eigenvalue problem (GNHEP)

$$Ax = \lambda Bx, \quad (8.1)$$

where A and B are general n by n matrices. Occasionally, one may also seek the solution of the *left* GNHEP

$$y^* A = \lambda y^* B. \quad (8.2)$$

The algebraic and analytic theory of the generalized eigenvalue problem is considerably more complicated than the corresponding theory of the standard eigenvalue problem. This has been discussed in §2.6. This chapter covers a variety of numerical techniques for treating different sorts of the generalized eigenvalue problems.

In §8.2, a brief sketch of what is called the QZ algorithm for the generalized eigenvalue problem is presented. This is currently the most powerful method for dense problems and it is an analog of the QR algorithm. However, the QZ method is only suitable for small- to moderate-sized problems because of the requirements of $O(n^3)$ floating point operations and $O(n^2)$ memory locations.

A common approach for a large scale generalized eigenvalue problem is to reduce the problem (8.1) or (8.2) to a standard eigenvalue problem and then apply an iterative method, as described in Chapter 7. We refer to this technique as *reduction to standard form*. This reduction to standard form requires, for each iteration, the solution of a linear system with A or B or a combination of A and B . This will be discussed in more detail in §8.3.

The Jacobi–Davidson method, presented in §8.4, avoids the transformation of $Ax = \lambda Bx$ to a standard eigenproblem. It does not need the exact solution of a linear system, only that a preconditioned iteration be available for a system with the matrix $A - \sigma B$, which offers possibilities to improve overall efficiency.

The rational Krylov algorithm, introduced in §8.5, is a further development of shifted and inverted Arnoldi. It combines the bases from several shifts to find all

eigenvalues in a prescribed region of the complex plane. This is of special interest when one wants to know the response of a linear dynamic system over a large range of frequencies.

In §8.6, a pseudosymmetric Lanczos method is presented for a special type of the generalized eigenvalue problem (8.1), namely, where the matrices A and B are real symmetric, but neither A nor B nor a combination $\alpha A + \beta B$, for real numbers α and β , is positive definite. $A - \lambda B$ is called a *symmetric indefinite matrix pencil*. The symmetry of A and B is an algebraic property and it is not sufficient to ensure any of the special mathematical properties of a definite matrix pencil. Therefore, we cannot immediately use the algorithms described in Chapter 5. The pseudosymmetric Lanczos algorithm described in §8.6 can save about half of the memory requirements and floating point operations. However, without special precautions this technique could be numerically unstable.

In §8.7, a software package called GUPTRI is presented, which is designed for the singular generalized eigenvalue problem, that is, when the characteristic polynomial $p(\lambda) = \det(A - \lambda B)$ is identical to zero for all λ . This happens, for example, when A and B have a common null space. The algorithms described in this section are efficient for small and dense problems. The costs of algorithms are of order $O(n^4)$.

In the final section, 8.8, tools to assess the accuracy of computed eigenvalues and corresponding eigenvectors of the regular GNHEP are discussed.

8.2 Direct Methods

The QZ algorithm is the method of choice for computing the GNHEP (8.1). The algorithm is an analog of the QR algorithm (see §7.3) for the generalized eigenvalue problem. It follows the pattern described for the QR algorithm:

1. A and B are first simultaneously reduced to condensed forms by unitary equivalence transformations. More specifically, A is reduced to upper Hessenberg form and B is reduced to upper triangular form (Schur form). The trick is now to reduce A also to upper Schur form, while keeping B in that form. This is accomplished in the next step.
2. The effect of one shifted QR step on AB^{-1} (without forming that matrix product) is simulated by unitary equivalence transformations Q and Z on the matrix pair A and B ; this is the heart of the QZ iteration. If applied iteratively, it reduces A to triangular or quasi-triangular form (that is, with 2 by 2 blocks along the diagonal, in order to avoid complex arithmetic), while preserving the triangular structure of B . At convergence, we have the generalized Schur form of A and B (see §2.6); that is, we have computed orthogonal Q and Z so that QAZ and QBZ are upper triangular.
3. Eigenvalues can be computed from the diagonals of the triangular form. Eigenvectors can be computed as the eigenvectors of the triangular problem and then transformed back with Z to the eigenvectors of the original problem.

This brief sketch has necessarily left a number of important questions untreated. For more details, the reader should consult the literature: the original paper on the QZ algorithm is Moler and Stewart [328]; the QZ algorithm is also described in the textbooks by, for examples, Golub and Van Loan [198] or Demmel [114].

The QZ algorithm leads to all eigenvalues and, optionally, to the right and left eigenvectors. It requires $O(n^3)$ floating point operations and $O(n^2)$ memory locations, where n is the order of A and B . More specifically, it requires about $30n^3$ floating point operations for computing the eigenvalues only. If right eigenvectors are desired, then an additional $16n^3$ are necessary, and likewise for the left eigenvectors. These estimates of work are based on the experience that about two QZ iterations per eigenvalue are sufficient (the convergence properties of QZ are about the same as for QR).

Subroutines that implement QZ algorithms are included in most linear algebra-related software packages. It is used as the `eig(A,B)` command in MATLAB.¹ In LAPACK [12], the following driver routines are available for performing a variety of tasks:

`xGGES`: Computes generalized eigenvalues, the generalized Schur form, and optionally the left and/or right matrices of Schur vectors.

`xGGESX`: `xGGES` plus condition estimates of eigenvalues and deflating subspaces.

`xGGEV`: Generalized eigenvalues, and optionally the left and/or right generalized eigenvectors.

`xGGEVX`: `xGGEV` plus condition estimates for eigenvalues and eigenvectors.

The letter `x` stands for `S` or `D` for real single or double precision data types, or `C` or `Z` for complex single or double precision data types.

8.3 Transformation to Standard Problems

A common approach for the numerical solution of the large sparse generalized eigenvalue problem (8.1) is to transform the problem first to an equivalent standard eigenvalue problem and then to apply an appropriate iterative method as described in Chapter 7. In this section, we will discuss three approaches for the transformation to a standard eigenproblem. The first approach (invert B) is recommended only if the matrix B has a very simple structure and when linear systems of equations with matrices B and B^* can be solved efficiently. The second approach (split-invert B) is suitable when the matrix B is Hermitian positive definite and when the Cholesky decomposition of B can be computed efficiently a priori. For numerical stability, these two approaches require that the matrix B be well-conditioned. The third approach is the shift-and-invert spectral transformation (SI), and this is the most common approach. We recommend using it whenever possible.

Invert B . If B is nonsingular, then the problem (8.1) is equivalent to

$$(B^{-1}A)x = \lambda x \quad \text{and} \quad \tilde{y}^*(B^{-1}A) = \lambda \tilde{y}^*, \quad (8.3)$$

where $\tilde{y} = B^*y$. Alternatively, the problem (8.1) is also equivalent to

$$(AB^{-1})\tilde{x} = \lambda \tilde{x} \quad \text{and} \quad y^*(AB^{-1}) = \lambda y^*, \quad (8.4)$$

¹It has been announced that an LAPACK-based numerics library will be part of the next major release of MATLAB; see *The Newsletter for MATLAB, Simulink and Toolbox Users, Winter 2000* available at <http://www.mathworks.com/company/newsletter/clevescorner/winter2000.cleve.shtml>

where $\tilde{x} = Bx$. If the right eigenvectors are of primary interest, then (8.3) is preferable, since it avoids an additional back transformation.

For an iterative method, as described in Chapter 7, for the reduced standard eigenvalue problem (8.3) or (8.4), it is not necessary to evaluate the product $B^{-1}A$ or AB^{-1} . This is a key observation in the treatment of large sparse A and B . One only needs to evaluate matrix-vector products, like

$$r = (B^{-1}A)q,$$

for a given vector q . For a two-sided Lanczos-type method, the vector

$$s = (B^{-1}A)^*p$$

is also necessary for given vector p . Note that the vector $r = (B^{-1}A)q$ can be computed in two steps:

- (a) form $u = Aq$,
- (b) solve $Bu = u$ for r .

Similarly, the vector $s = (B^{-1}A)^*p$, if necessary, can be evaluated as

- (a)' solve $B^*v = p$ for v ,
- (b)' form $s = A^*v$.

One can exploit sparsity in each of these steps. In general, the iterative methods, with the exception of the Jacobi–Davidson method, require accurate solution of the linear systems at step (b) (and (b)'). If possible, a direct linear system solver, say, using LU factorization of B , is preferable. See §10.3 for the discussions on dense or sparse LU factorization.

The error introduced by this transformation to standard form can be proportional to $\|A\|_2\|B^{-1}\|_2$. If B is ill-conditioned, then the approach is potentially suspect. In that situation, one may consider the SI for the transformation or the usage of the Jacobi–Davidson method discussed below.

Split-and-invert B . In some applications, B is Hermitian positive definite and well-conditioned. In this case, it is recommended to compute first a sparse Cholesky decomposition

$$B = LL^*,$$

with L a lower triangular matrix; see §10.3. The equivalent standard eigenvalue problem is

$$(L^{-1}AL^{-*})\tilde{x} = \lambda\tilde{x} \quad \text{and} \quad \tilde{y}^*(L^{-1}AL^{-*}) = \lambda\tilde{y}^*, \quad (8.5)$$

where $\tilde{x} = L^*x$ and $\tilde{y} = L^*y$. As in Invert B , matrices like $L^{-1}AL^{-*}$ should never be evaluated explicitly, more in particular because they will in general not be sparse. For the application of an iterative method for the standard eigenvalue problem, we only need to provide the efficient evaluation of matrix-vector products, like

$$r = (L^{-1}AL^{-*})q$$

or

$$s = (L^{-1}A^*L^{-*})p,$$

where q and p are given vectors. In practice, the vector $r = (L^{-1}AL^{-*})q$ can be formed in three steps:

- (a) solve $L^*u = q$ for u ,
- (b) form $w = Au$,
- (c) solve $Lr = w$ for r .

The vector $s = (L^{-1}A^*L^{-*})p$, when necessary, can be formed as follows:

- (a)' solve $L^*u = p$ for u ,
- (b)' form $w = A^*u$,
- (c)' solve $Ls = w$ for s .

Since L is a triangular matrix, the solutions of linear systems with L or L^* can be obtained by forward and backward substitutions. Sparsity can be exploited in a straightforward manner in all these steps.

Shift-and-Invert. None of the transformations to standard form, (8.3), (8.4), or (8.5), can be used when both A and B are singular or when B is ill-conditioned. An attractive and popular technique is to apply first the shift to the original problem and then carry out the split-invert. This is the SI, as discussed in §3.3. Specifically, let σ be a user-selected shift such that the matrix $A - \sigma B$ is nonsingular; then the original problem (8.1) can be transformed to

$$Cx = \mu x \quad \text{and} \quad z^*C = \mu z^*, \quad (8.6)$$

where

$$C = (A - \sigma B)^{-1}B, \quad \mu = \frac{1}{\lambda - \sigma}, \quad \text{and} \quad z = (A - \sigma B)^*y.$$

We see that the eigenvalues λ of the problem (8.1) closest to the shift σ are mapped as the exterior eigenvalues of the reduced standard eigenvalue problem (8.6), that is, to the eigenvalues of largest magnitude, and these are the eigenvalues that are first well approximated by the iterative methods.

In practice, an effective shift selection depends on the user's preferences and on knowledge of the underlying generalized eigenproblem. A good shift not only amplifies the desired eigenvalues, but it also leads to a well-conditioned matrix $A - \sigma B$. This often makes the task of selecting good shifts a challenging one.

For the application of an iterative method for the reduced standard eigenvalue problem (8.6), one needs to evaluate matrix-vector products

$$r = Cq = ((A - \sigma B)^{-1}B)q$$

or

$$s = C^*p = ((A - \sigma B)^{-1}B)^*p$$

for given vectors q and p . For an efficient evaluation, let

$$A - \sigma B = \mathcal{L}\mathcal{U} \quad (8.7)$$

represent some convenient factorization of $A - \sigma B$, where \mathcal{L} and \mathcal{U} are square matrices. Since $A - \sigma B$ is assumed to be nonsingular, the factors \mathcal{L} and \mathcal{U} are also nonsingular. The factorization should be chosen so that the corresponding linear systems of equations with \mathcal{L} , \mathcal{L}^* and/or \mathcal{U} , \mathcal{U}^* can be solved efficiently, and typically, sparse LU

factorizations are used. See §10.3. Of course, one can also select $\mathcal{L} = A - \sigma B$ and $\mathcal{U} = I$ if this leads to convenient linear systems.

With the above factorization, the matrix-vector product $r = Cq$ can be evaluated as follows:

- (a) form $v = Bq$,
- (b) solve $\mathcal{L}w = v$ for w ,
- (c) solve $\mathcal{U}r = w$ for r .

Similar, the matrix-vector product $s = C^*p$ can be evaluated as following three steps:

- (a)' solve $\mathcal{U}^*v = p$ for v ,
- (b)' solve $\mathcal{L}^*w = v$ for w ,
- (c)' form $s = B^*w$.

The SI technique is a powerful tool in the treatment of the generalized eigenvalue problem (8.1). The major problem, which often becomes bottleneck, is to find a convenient factorization (8.7) of $A - \sigma B$ so that the associated linear systems of equations can be solved efficiently. If accurate solution of the linear systems with $A - \sigma B$ becomes too expensive, then one may consider the usage of inexact Cayley transformations (see §11.2), or the Jacobi–Davidson method.

8.4 Jacobi–Davidson Method

G. Sleijpen and H. van der Vorst

8.4.1 Basic Theory

As for the GHEP, we want to avoid transformation of $Ax = \lambda Bx$ to a standard eigenproblem. This would require the solution of some linear system, involving B and/or A , per iteration step. Furthermore, for stability reasons we want to work with orthonormal transformations, and for this reason our goal is to compute Schur vectors for the pencil $A - \lambda B$, rather than eigenvectors. This leads to a generalization of the Jacobi–Davidson algorithm, in which we compute a partial Schur form for the pencil. This algorithm can be interpreted as a subspace iteration variant of the QZ algorithm. A consequence of this approach is that we have to work with search and test spaces that are different.

With $\lambda = \alpha/\beta$, the generalized eigenproblem $Ax = \lambda Bx$ is equivalent to the eigenproblem

$$(\beta A - \alpha B)x = 0, \quad (8.8)$$

and we denote a generalized eigenvalue of the matrix pair $\{A, B\}$ as a pair (α, β) . This approach is preferred, because underflow or overflow for $\lambda = \alpha/\beta$ in finite precision arithmetic may occur when α and/or β are zero or close to zero, in which case the pair is still well defined [328], [425, Chap. VI], [376].

A *partial generalized Schur form* of dimension k for a matrix pair $\{A, B\}$ is the decomposition

$$AQ_k = Z_k R_k^A, \quad BQ_k = Z_k R_k^B, \quad (8.9)$$

where Q_k and Z_k are orthogonal n by k matrices and R_k^A and R_k^B are upper triangular k by k matrices. A column q_i of Q_k is referred to as a *generalized Schur vector*,

and we refer to a pair $((\alpha_i, \beta_i), q_i)$, with $(\alpha_i, \beta_i) = (R_k^A(i, i), R_k^B(i, i))$ as a *generalized Schur pair*. It follows that if $((\alpha, \beta), y)$ is a generalized eigenpair of (R_k^A, R_k^B) then $((\alpha, \beta), Q_k y)$ is a generalized eigenpair of $\{A, B\}$.

We will now describe a Jacobi–Davidson method for the generalized eigenproblem (8.8). From the relations (8.9) we conclude that

$$\beta_i A q_i - \alpha_i B q_i \perp z_i,$$

which suggests that we should follow a Petrov–Galerkin condition for the construction of reduced systems. In each step the approximate eigenvector u is selected from a *search subspace* $\text{span}(V)$. We require that the residual $\eta A u - \zeta B u$ is orthogonal to some other well-chosen *test subspace* $\text{span}(W)$:

$$\eta A u - \zeta B u \perp \text{span}(W). \quad (8.10)$$

Both subspaces are of the same dimension, say m . Equation (8.10) leads to the *projected eigenproblem*

$$(\eta W^* A V - \zeta W^* B V) s = 0. \quad (8.11)$$

The pencil $\eta W^* A V - \zeta W^* B V$ can be reduced by the QZ algorithm (see §8.2) to generalized Schur form (note that this is an m -dimensional problem). This leads to orthogonal m by m matrices S^R and S^L and upper triangular m by m matrices T^A and T^B , such that

$$(S^L)^*(W^* A V) S^R = T^A \quad \text{and} \quad (S^L)^*(W^* B V) S^R = T^B. \quad (8.12)$$

The decomposition can be reordered such that the first column of S^R and the $(1, 1)$ -entries of T^A and T^B represent the wanted Petrov solution [172]. With $s \equiv s_1^R \equiv S^R e_1$ and $\zeta \equiv T_{1,1}^A$, $\eta \equiv T_{1,1}^B$, the *Petrov vector* is defined as $u \equiv V s$ for the associated *generalized Petrov value* (ζ, η) . In our algorithm we will construct orthogonal matrices V and W , so that also $\|u\|_2 = 1$. With the decomposition in (8.12), we construct an approximate partial generalized Schur form (cf. (8.9)): $V S^R$ approximates a Q_k , and $W S^L$ approximates the associated Z_k . Since $\text{span}(Z_k) = \text{span}(A Q_k) = \text{span}(B Q_k)$ (cf. (8.9)), this suggests to choose W such that $\text{span}(W)$ coincides with $\text{span}(\nu_0 A V + \mu_0 B V)$. With the weights ν_0 and μ_0 we can influence the convergence of the Petrov values. If we want eigenpair approximations for eigenvalues λ close to a target τ , then the choice

$$\nu_0 = 1/\sqrt{1+|\tau|^2}, \quad \mu_0 = -\tau\nu_0$$

is very effective [172], especially if we want to compute eigenvalues in the interior of the spectrum of $A - \lambda B$. We will call the Petrov approximations for this choice the *harmonic Petrov eigenpairs*. The Jacobi–Davidson correction equation for the component $t \perp u$ for the pencil $\eta A - \zeta B$ becomes

$$\left(I - \frac{pp^*}{p^*p} \right) (\eta A - \zeta B) (I - uu^*) t = -r, \quad (8.13)$$

with $r \equiv \eta A u - \zeta B u$ and $p \equiv \nu_0 A u + \mu_0 B u$. It can be shown that if (8.13) is solved exactly, the convergence to the generalized eigenvalue will be quadratic; see [408, Theorem 3.2]. Usually, this correction equation is solved only approximately, for instance, with a (preconditioned) iterative solver. The obtained vector t is used for the expansion

v of V and $\nu_0 Av + \mu_0 Bv$ is used for the expansion of W . For both spaces we work with orthonormal bases. Therefore, the new columns are orthonormalized with respect to the current basis by a modified Gram–Schmidt orthogonalization process (see §4.7.1).

It can be shown that, with the above choices for p and W ,

$$p = Ws_1^L = WS^L e_1. \quad (8.14)$$

The relation between the partial generalized Schur form for the given large problem and the complete generalized Schur form for the reduced problem (8.11) via right vectors ($u = Vs_1^R$) is similar to the relation via left vectors ($p = Vs_1^L$). This can also be exploited for restarts.

8.4.2 Deflation and Restart

Similar to the situation for the standard eigenproblem (see §4.7.3 (p. 95) and §7.12.2 (p. 221)), the Jacobi–Davidson process can be enhanced with restart possibilities in order to restrict the dimension of the search subspace. The process can also be combined with deflation in order to find a number of different generalized Schur pairs.

Deflation. The partial generalized Schur form can be obtained in a number of successive steps. Suppose that we have the partial generalized Schur form $AQ_{k-1} = Z_{k-1}R_{k-1}^A$ and $BQ_{k-1} = Z_{k-1}R_{k-1}^B$. We want to expand this partial generalized Schur form with the new right Schur vector q and the left Schur vector z to

$$A \begin{bmatrix} Q_{k-1} & q \end{bmatrix} = \begin{bmatrix} Z_{k-1} & z \end{bmatrix} \begin{bmatrix} R_{k-1}^A & a \\ 0 & \alpha \end{bmatrix}$$

and

$$B \begin{bmatrix} Q_{k-1} & q \end{bmatrix} = \begin{bmatrix} Z_{k-1} & z \end{bmatrix} \begin{bmatrix} R_{k-1}^B & b \\ 0 & \beta \end{bmatrix}.$$

The new generalized Schur pair $((\alpha, \beta), q)$ satisfies

$$Q_{k-1}^* q = 0 \quad \text{and} \quad (\beta A - \alpha B) q - Z_{k-1}(\beta a - \alpha b) = 0,$$

or, since $\beta a - \alpha b = Z_{k-1}^*(\beta A - \alpha B)q$,

$$Q_{k-1}^* q = 0 \quad \text{and} \quad (I - Z_{k-1}Z_{k-1}^*) (\beta A - \alpha B) (I - Q_{k-1}Q_{k-1}^*) q = 0.$$

The vectors a and b can be computed from

$$a = Z_{k-1}^* A q \quad \text{and} \quad b = Z_{k-1}^* B q.$$

Hence, the generalized Schur pair $((\alpha, \beta), q)$ is an eigenpair of the deflated matrix pair

$$\begin{aligned} & ((I - Z_{k-1}Z_{k-1}^*) A (I - Q_{k-1}Q_{k-1}^*)) , \\ & (I - Z_{k-1}Z_{k-1}^*) B (I - Q_{k-1}Q_{k-1}^*) . \end{aligned} \quad (8.15)$$

This eigenproblem can be solved again with the Jacobi–Davidson process that we have outlined in §8.4.1. In that process we construct vectors v_i that are orthogonal to Q_{k-1}

and vectors w_i that are orthogonal to Z_{k-1} . This simplifies the computation of the interaction matrices M^A and M^B , associated with the deflated operators:

$$\begin{cases} M^A \equiv W^* (I - Z_{k-1} Z_{k-1}^*) A (I - Q_{k-1} Q_{k-1}^*) V = W^* AV, \\ M^B \equiv W^* (I - Z_{k-1} Z_{k-1}^*) B (I - Q_{k-1} Q_{k-1}^*) V = W^* BV, \end{cases} \quad (8.16)$$

and M^A and M^B can be simply computed as W^*AV and W^*BV , respectively.

Restart. Suppose that the generalized Schur form (8.12) is ordered with respect to τ such that

$$|T_{1,1}^A/T_{1,1}^B - \tau| \leq |T_{2,2}^A/T_{2,2}^B - \tau| \leq \cdots \leq |T_{m,m}^A/T_{m,m}^B - \tau|,$$

where m is the dimension of $\text{span}(V)$. Then, for $i < m$, the space $\text{span}(Vs_1^R, \dots, Vs_i^R)$ spanned by the first i columns of VS^R contains the i most promising Petrov vectors. The corresponding test subspace is given by $\text{span}(Ws_1^L, \dots, Ws_i^L)$. Therefore, in order to reduce the dimension of the subspaces (“implicit restart”) to m_{\min} , $m_{\min} < m$, the columns $v_{m_{\min}+1}$ through v_m and $w_{m_{\min}+1}$ through w_m can simply be discarded and the Jacobi–Davidson algorithm can be continued with

$$V = [Vs_1^R, \dots, Vs_{m_{\min}}^R] \quad \text{and} \quad W = [Ws_1^L, \dots, Ws_{m_{\min}}^L].$$

8.4.3 Algorithm

The Jacobi–Davidson algorithm is given in Algorithm 8.1. This algorithm attempts to compute the generalized Schur pairs $((\alpha, \beta), q)$, for which the ratio β/α is closest to a specified target value τ in the complex plane. The algorithm includes restart in order to limit the dimension of the search space, and deflation with already converged left and right Schur vectors.

To apply this algorithm we need to specify a starting vector v_0 , a tolerance ϵ , a target value τ , and a number k_{\max} that specifies how many eigenpairs near τ should be computed. The value of m_{\max} specifies the maximum dimension of the search subspace. If it is exceeded then a restart takes place with a subspace of dimension m_{\min} .

On completion the k_{\max} generalized eigenvalues close to τ are delivered, and the corresponding reduced Schur form $AQ = ZR^A$, $BQ = ZR^B$, where Q and Z are n by k_{\max} orthogonal and R^A , R^B are k_{\max} by k_{\max} upper triangular. The generalized eigenvalues are the on-diagonals of R^A and R^B . The computed form satisfies $\|Aq_j - ZR^A e_j\|_2 = O(\epsilon)$, $\|Bq_j - ZR^B e_j\|_2 = O(\epsilon)$, where q_j is the j th column of Q .

The accuracy of the computed reduced Schur form depends on the distance between the target value τ and the eigenvalue $(\alpha_j, \beta_j) \equiv (R_{j,j}^A, R_{j,j}^B)$. If we neglect terms of order machine precision and of order ϵ^2 , then we have that $\|Aq_j - ZR^A e_j\|_2 \leq j\gamma_A \epsilon$, $\|Bq_j - ZR^B e_j\|_2 \leq j\gamma_B \epsilon$, where the constants γ_A and γ_B are given by

$$\gamma_A \equiv \frac{|\mu_0|}{|\nu_0 \alpha_j + \mu_0 \beta_j|} \quad \text{and} \quad \gamma_B \equiv \frac{|\nu_0|}{|\nu_0 \alpha_j + \mu_0 \beta_j|}.$$

If $\mu_0/\nu_0 = -\tau$, as in step (1) of the algorithm, then $\gamma_A = |\tau|/|\alpha_j - \tau\beta_j|$, $\gamma_B = 1/|\alpha_j - \tau\beta_j|$. These values can be large if $\tau \approx \alpha_j/\beta_j$. In practice an accuracy of order ϵ is achieved also if τ is close to detected eigenvalues. The ϵ -accuracy can be

ALGORITHM 8.1: Jacobi–Davidson QZ Method for k_{\max} Interior Eigenvalues Close to τ for GNHEP

(1) $t = v_0, k = 0, \nu_0 = 1/\sqrt{1 + |\tau|^2}, \mu_0 = -\tau\nu_0, m = 0;$
(2) $Q = [], Z = [], S = [], T = []$
(3) **while** $k < k_{\max}$
(4) **for** $i = 1, \dots, m$
(5) $t = t - (v_i^* t)v_i$
(6) $m = m + 1, v_m = t/\|t\|_2, v_m^A = Av_m, v_m^B = Bv_m, w = \nu_0 v_m^A + \mu_0 v_m^B$
(7) **for** $i = 1, \dots, k$
(8) $w = w - (z_i^* w)z_i$
(9) **for** $i = 1, \dots, m - 1$
(10) $w = w - (w_i^* w)w_i$
(11) $w_m = w/\|w\|_2$
(12) **for** $i = 1, \dots, m - 1$
(13) $M_{i,m}^A = w_i^* v_m^A, M_{m,i}^A = w_m^* v_i^A, M_{i,m}^B = w_i^* v_m^B, M_{m,i}^B = w_m^* v_i^B$
(14) $M_{m,m}^A = w_m^* v_m^A, M_{m,m}^B = w_m^* v_m^B$
(15) compute the QZ decomposition: $M^A S^R = S^L T^A, M^B S^R = S^L T^B$
 such that: $|T_{i,i}^A/T_{i,i}^B - \tau| \leq |T_{i+1,i+1}^A/T_{i+1,i+1}^B - \tau|$
(16) $u = Vs_1^R, p = Ws_1^L, u^A = V^A s_1^R, u^B = V^B s_1^R, \zeta = T_{1,1}^A, \eta = T_{1,1}^B$
(17) $r = \eta u^A - \zeta u^B, \tilde{a} = Z^* u^A, \tilde{b} = Z^* u^B, \tilde{r} = r - Z(\eta \tilde{a} - \zeta \tilde{b})$
(18) **while** $\|\tilde{r}\|_2 \leq \epsilon$
(19) $R^A = \begin{bmatrix} R^A & \tilde{a} \\ 0 & \zeta \end{bmatrix}, R^B = \begin{bmatrix} R^B & \tilde{b} \\ 0 & \eta \end{bmatrix}$
(20) $Q = [Q, u], Z = [Z, p], k = k + 1$
(21) **if** $k = k_{\max}$ **then stop**
(22) $m = m - 1$
(23) **for** $i = 1, \dots, m$
(24) $v_i = Vs_{i+1}^R, v_i^A = V^A s_{i+1}^R, v_i^B = V^B s_{i+1}^R,$
(25) $w_i = Ws_{i+1}^L, s_i^R = s_i^L = e_i$
(26) M^A, M^B is the lower m by m block of T^A, T^B , resp.
(27) $u = v_1, p = w_1, u^A = v_1^A, u^B = v_1^B, \zeta = T_{1,1}^A, \eta = T_{1,1}^B$
(28) $r = \eta u^A - \zeta u^B, \tilde{a} = Z^* u^A, \tilde{b} = Z^* u^B, \tilde{r} = r - Z(\eta \tilde{a} - \zeta \tilde{b})$
(29) **end while**
(30) **if** $m \geq m_{\max}$ **then**
(31) **for** $i = 2, \dots, m_{\min}$
(32) $v_i = Vs_i^R, v_i^A = V^A s_i^R, v_i^B = V^B s_i^R, w_i = Ws_i^L$
(33) M^A, M^B is the leading m_{\min} by m_{\min} block of T^A, T^B , resp.
(34) $v_1 = u, v_1^A = u^A, v_1^B = u^B, w_1 = p, m = m_{\min}$
(35) **end if**
(36) $\tilde{Q} = [Q, u], \tilde{Z} = [Z, p]$
(37) solve $t \perp \tilde{Q}$ (approximately) from
 $(I - \tilde{Z}\tilde{Z}^*)(\eta A - \zeta B)(I - \tilde{Q}\tilde{Q}^*)t = -\tilde{r}$
(38) **end while**

guaranteed when an additional refinement step is performed with values for (μ_0, ν_0) as $(\mu_0, \nu_0) = (1, \bar{\tau})$.

We will now explain the successive main phases of the algorithm.

- (1) The initialization phase. The choice for the scalars ν_0 and μ_0 is in particular effective if τ is in the interior of the spectrum. The choice causes a breakdown if τ is an eigenvalue (which can easily be tested).
- (4)–(5) The new vector t is made orthogonal with respect to the current search subspace V by means of modified Gram–Schmidt. Likewise, the vector $w = (\nu_0 A + \mu_0 B)t$ is made orthogonal with respect to the current test subspace W . The two orthogonalization processes can be replaced, for improved numerical stability, by a template as in Algorithm 4.14 (p. 93).

We expand the subspaces V , $V^A \equiv AV$, $V^B \equiv BV$, and W . V denotes the matrix with the current basis vectors v_i for the search subspace as its columns. The other matrices are defined in a similar obvious way.

- (12)–(13) The m th row and column of the matrices $M^A \equiv W^*AV$ and $M^B \equiv W^*BV$ are computed.

Note that the scalars $M_{i,m}^B$ can also be computed from the scalars $M_{i,m}^A$, and the orthogonalization constants of w_i^*w in step (10).

- (15) The QZ decomposition for the pair (M^A, M^B) of m by m matrices can be computed by a suitable routine for dense matrix pencils from LAPACK.

We have chosen to compute the generalized Petrov pairs, which makes the algorithm suitable for computing k_{\max} interior generalized eigenvalues of $\beta A - \alpha B$, for which α/β is close to a specified τ .

For algorithms for reordering the generalized Schur form, see [448, 449, 171].

- (18) The stopping criterion is to accept a generalized eigenpair approximation as soon as the norm of the residual (for the normalized right Schur vector approximation) is below ϵ . This means that we accept inaccuracies in the order of ϵ in the computed generalized eigenvalues, and inaccuracies (in angle) in the Schur vectors of $O(\epsilon)$ (provided that the concerned eigenvalue is simple and well separated from the others).

Detection of all wanted eigenvalues cannot be guaranteed; see note (13) for Algorithm 4.17 (p. 97).

- (22) After acceptance of a Petrov pair, we continue the search for a next pair, with the remaining Petrov vectors as a basis for the initial search space.
- (30) We restart when the dimension of the search space for the current eigenvector exceeds m_{\max} . The process is restarted with the subspaces spanned by the m_{\min} left and right Ritz vectors corresponding to the generalized Ritz pairs closest to the target value τ .
- (36) We have collected the locked (computed) right Schur vectors in Q , and the matrix \tilde{Q} is Q expanded with the current right Schur eigenvector approximation u . Likewise, the converged left Schur vectors have been collected in Z , and this matrix

is expanded with p . This is done in order to obtain a more compact formulation; the correction equation in step (37) of Algorithm 8.1 is equivalent to the one in equation (8.13) for the deflated pair in (8.15). The new correction t has to be orthogonal to the columns of Q as well as to u .

Of course, the correction equation can be solved by any suitable process, for instance, a preconditioned Krylov subspace method that is designed to solve unsymmetric systems. However, because of the different projections, we always need a preconditioner (which may be the identity operator if nothing else is available) that is deflated by the same skew projections so that we obtain a mapping between \tilde{Q}^\perp and itself. Because of the occurrence of \tilde{Q} and \tilde{Z} , one has to be careful with the usage of preconditioners for the matrix $\eta A - \zeta B$. The inclusion of preconditioners can be done as in Algorithm 8.2. Make sure that the starting vector t_0 for an iterative solver satisfies the orthogonality constraints $\tilde{Q}^* t_0 = 0$. Note that significant savings per step can be made in Algorithm 8.2 if K is kept the same for a (few) Jacobi–Davidson iterations. In that case columns of \tilde{Z} can be saved from previous steps. Also the matrix \mathcal{M} can be updated from previous steps, as well as its \mathcal{LU} decomposition.

It is not necessary to solve the correction equation very accurately. A strategy, often used for inexact Newton methods [113], also works well here: increase the accuracy with the Jacobi–Davidson iteration step, for instance, solve the correction equation with a residual reduction of $2^{-\ell}$ in the ℓ th Jacobi–Davidson iteration (ℓ is reset to 0 when a Schur vector is detected).

In particular, in the first few initial steps, the approximate eigenvalue θ may be very inaccurate, and then it does not make sense to solve the correction equation accurately. In this stage it can be more effective to temporarily replace θ by τ or to take $t = -r$ for the expansion of the search subspace [335, 172].

For the full theoretical background of this method, as well as details on the deflation technique with Schur vectors, see [172].

8.4.4 Software Availability

There are MATLAB and FORTRAN implementations of the algorithms described in this section. For information about this software, including how to access it, see the book’s homepage, ETHOME.

8.4.5 Numerical Example

We present the results for a small example that can be easily repeated. We took the example from the collection of test matrices in [28].

We consider the bounded fineline dielectric waveguide generalized eigenproblem BFW782 [28] of order 782. This problem stems from a finite element discretization of the Maxwell equation for propagating modes and magnetic field profiles of a rectangular waveguide filled with dielectric and PEC structures. The resulting matrix A is non-symmetric and the matrix B is positive definite. Of special interest are the generalized eigenvalues (α, β) with positive real part (i.e., $\operatorname{Re}(\alpha/\beta) \geq 0$) and their corresponding eigenvectors.

ALGORITHM 8.2: Approximate Solution of the Deflated Jacobi–Davidson GNHEP Correction Equation

“Solve” with left preconditioner $\tilde{K} \equiv (I - \tilde{Z}\tilde{Z}^*)K(I - \tilde{Q}\tilde{Q}^*)$
 for $\tilde{A} \equiv (I - \tilde{Z}\tilde{Z}^*)(\eta A - \zeta B)(I - \tilde{Q}\tilde{Q}^*)$:

- (1) solve \tilde{Z} from $K\tilde{Z} = \tilde{Z}$,
- (2) compute $\mathcal{M} = \tilde{Q}^*\tilde{Z}$
- (3) decompose $\mathcal{M} = \mathcal{L}\mathcal{U}$
- (4) compute $\tilde{r} \equiv \tilde{K}^{-1}r$ as:
 - (b') solve \hat{r} from $K\hat{r} = r$
 - (c') $\vec{\gamma} = \tilde{Q}^*\hat{r}$
 - (7) solve $\vec{\beta}$ from $\mathcal{L}\vec{\beta} = \vec{\gamma}$
 - (8) solve $\vec{\alpha}$ from $\mathcal{U}\vec{\alpha} = \vec{\beta}$
 - (d') $\tilde{r} = \hat{r} - \tilde{Z}\vec{\alpha}$
- (10) apply Krylov subspace method with start $t_0 = 0$, with operator $\tilde{K}^{-1}\tilde{A}$,
 and right-hand side $-\tilde{r}$, $z = \tilde{K}^{-1}\tilde{A}v$ for given v is computed as:
 - (a) $y = (\eta A - \zeta B)v$
 - (b) solve \hat{y} from $K\hat{y} = y$
 - (c) $\vec{\gamma} = \tilde{Q}^*\hat{y}$
 - (14) solve $\vec{\beta}$ from $\mathcal{L}\vec{\beta} = \vec{\gamma}$
 - (15) solve $\vec{\alpha}$ from $\mathcal{U}\vec{\alpha} = \vec{\beta}$
 - (16) (d) $z = \hat{y} - \tilde{Z}\vec{\alpha}$

For this problem, the parameters were set to $\tau = 2750.0$, $k_{\max} = 5$, and $\epsilon = 10^{-9}$. In the first few steps, until the size of the first residual was smaller than 10^{-6} , we replaced (ζ, η) in the correction equation by $(1, \tau)$ (as explained in note (36)).

The computed generalized eigenvalues, represented as α/β , are given in Table 8.1. With Algorithm 8.1 we discovered all four positive generalized eigenvalues.

The convergence history is plotted in Figure 8.1. We solved the correction equation (1) by simply taking t as $-\tilde{r}$, denoted by GMRES₁; (2) with full GMRES [389] with a maximum of 10 steps, denoted by GMRES₁₀, and (3) with Bi-CGSTAB(2) [409] with a maximum of 100 matrix multiplications (Bi-CGSTAB refers to biconjugate gradient stabilized). We did not use preconditioning ($K = I$). As stopping criterion for the iterative methods for the correction equation, we used a residual reduction of $2^{-\ell}$ in the ℓ th Jacobi–Davidson iteration or on the maximum number of iterations permitted. A summary of the results is given in Table 8.2. We see that the Jacobi–Davidson QZ

Table 8.1: Five generalized eigenvalues of BFW782, computed by Jacobi–Davidson QZ

−1.1373e + 03
5.6467e + 02
1.2634e + 03
2.4843e + 03
2.5233e + 03

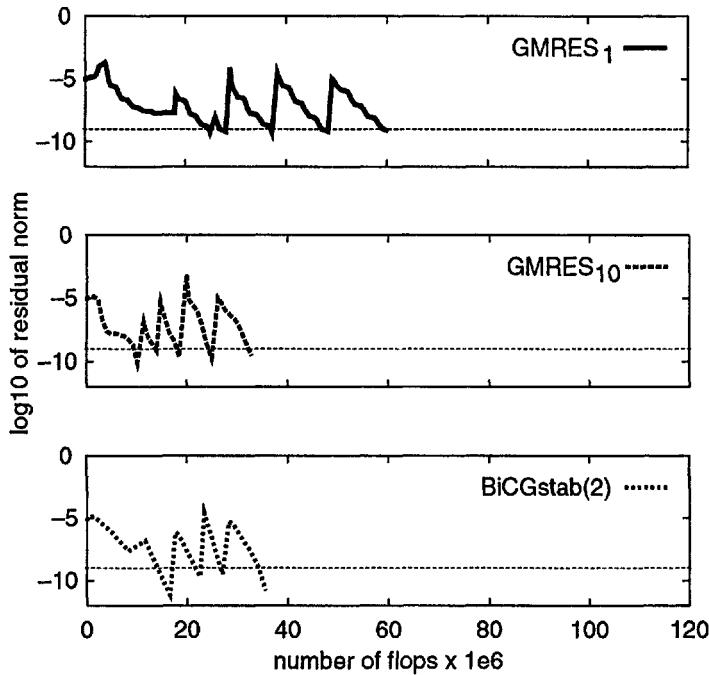


Figure 8.1: Convergence history for BFW782.

Table 8.2: Summary of results for BFW782.

Method for the correction equation	JD iterations	MVs	flops $\times 10^6$
GMRES ₁	143	143	67
GMRES ₁₀	37	233	31.7
Bi-CGSTAB(2)	32	429	38.8

method converges quite nicely for GMRES₁₀ and Bi-CGSTAB(2). It should be noted that although it seems that with Bi-CGSTAB(2) only four generalized eigenvalues are computed, in fact five generalized eigenvalues are computed: the two rightmost generalized eigenvalues, which are relatively close, are found in the same Jacobi–Davidson iteration.

8.5 Rational Krylov Subspace Method

A. Ruhe

The rational Krylov subspace method computes eigenvalues and eigenvectors of a regular pencil,

$$(A - \lambda B)x = 0 . \quad (8.17)$$

It is a generalization of the shift-and-invert Arnoldi algorithm, in which several factorizations with different shifts are used in one run. This way we get good approximations to all the eigenvalues in a union of regions around the shifts chosen; see [378]. A typical application is model reduction of a linear dynamical system, where one wants to get the response over a prescribed range of frequencies; see, e.g., [379] or [184].

Rational Krylov starts as a shifted and inverted Arnoldi iteration with shift σ_1 and starting vector v_1 . It computes an orthonormal basis V_j using the basic recursion,

$$(A - \sigma_1 B)^{-1} B V_j = V_{j+1} H_{j+1,j} . \quad (8.18)$$

(The subscripts indicate sizes of matrices, $H_{j+1,j}$ is $(j+1) \times j$, and V_j is a matrix of j columns each of full length n .) We may compute Ritz values from $H_{j,j}$ and Ritz approximate eigenvectors from V_j in the usual way. Solve the small-sized eigenvalue problem

$$H_{j,j} Z^{(j)} = Z^{(j)} \Theta^{(j)} \quad (8.19)$$

and get the Ritz values $\lambda_i^{(j)} = \sigma_1 + 1/\theta_i^{(j)}$ and Ritz vectors $y_i^{(j)} = V_j z_{:,i}^{(j)}$ for $i = 1, \dots, j$. We get good approximations after a moderate number of steps j to those eigenvalues λ_i that are closest to the shift σ_1 .

Now the idea behind the rational Krylov algorithm is to continue with a *new* shift σ_2 , when we want to get approximations to the eigenvalues close to that new shift. The tricky thing is to avoid throwing away all the information gathered in the basis V_j , computed using the old shift σ_1 . This is indeed possible if we replace the basis V_{j+1} with a new basis W_{j+1} , which spans the same subspace as V_{j+1} but can be interpreted as the orthogonal basis of an Arnoldi recursion (8.18) with the new shift σ_2 . At the same time the trapezoidal (Hessenberg) matrix $H_{j+1,j}$ is replaced by a new matrix of the same form, $\tilde{H}_{j+1,j}$.

Rewrite the recursion (8.18) as

$$B V_j = (A - \sigma_1 B) V_{j+1} H_{j+1,j} .$$

Add a matrix to the left-hand side, so that σ_1 is replaced by the new shift σ_2 in the right-hand side,

$$(\sigma_1 - \sigma_2) B V_{j+1} H_{j+1,j} + B V_j = (A - \sigma_2 B) V_{j+1} H_{j+1,j} ,$$

and note that B is the only large matrix in the left-hand side,

$$B V_{j+1} (I_{j+1,j} + (\sigma_1 - \sigma_2) H_{j+1,j}) = (A - \sigma_2 B) V_{j+1} H_{j+1,j} .$$

The matrix $K_{j+1,j} = I_{j+1,j} + (\sigma_1 - \sigma_2) H_{j+1,j}$ in the left-hand side is trapezoidal with the same pattern of nonzeros as $H_{j+1,j}$, and we may come back to an expression similar to the recursion (8.18) by premultiplying with $(A - \sigma_2 B)^{-1}$ to get

$$(A - \sigma_2 B)^{-1} B V_{j+1} K_{j+1,j} = V_{j+1} H_{j+1,j} .$$

It remains to get rid of the factor $K_{j+1,j}$ on the left-hand side, and this is done in the following way. Make a QR decomposition of $K_{j+1,j}$ and get

$$(A - \sigma_2 B)^{-1} B V_{j+1} Q_{j+1,j+1} \left[\begin{array}{c} R_{j,j} \\ 0 \end{array} \right] = V_{j+1} H_{j+1,j} .$$

Multiply from the right with the inverse of the triangular matrix $R_{j,j}$, which we know is nonsingular if the Hessenberg matrix is unreduced,

$$(A - \sigma_2 B)^{-1} B V_{j+1} Q_{j+1,j} = V_{j+1} H_{j+1,j} R_{j,j}^{-1}. \quad (8.20)$$

Introduce the new orthogonal basis $V_{j+1} Q_{j+1,j+1}$ and note that the pencil (8.17) is represented by the full matrix,

$$L_{j+1,j} = Q_{j+1,j+1}^* H_{j+1,j} R_{j,j}^{-1},$$

in this new basis. We may transform this $L_{j+1,j}$ into trapezoidal (upper Hessenberg) form by applying the Householder algorithm from the bottom upwards (see [377]),

$$L_{j+1,j} = \begin{bmatrix} P_j & 0 \\ 0 & 1 \end{bmatrix} \tilde{H}_{j+1,j} P_j^*. \quad (8.21)$$

Multiply the recursion (8.20) by the orthogonal P_j from the right and get

$$(A - \sigma_2 B)^{-1} B V_{j+1} Q_{j+1,j} P_j = V_{j+1} Q_{j+1} \begin{bmatrix} P_j & 0 \\ 0 & 1 \end{bmatrix} \tilde{H}_{j+1,j}$$

or

$$(A - \sigma_2 B)^{-1} B W_j = W_{j+1} \tilde{H}_{j+1,j},$$

an Arnoldi recursion (8.18) with the new shift σ_2 , the new orthogonal basis,

$$W_{j+1} = V_{j+1} Q_{j+1,j+1} \begin{bmatrix} P_j & 0 \\ 0 & 1 \end{bmatrix},$$

and the new trapezoidal (upper Hessenberg) matrix $\tilde{H}_{j+1,j}$.

We once again stress that all these transformations are effected without actually performing any operations on the large $n \times n$ matrices A and B . We may even accumulate all the Q and P factors and avoid forming W explicitly, thus avoiding all extra work on long vectors. Also note that even if we get an Arnoldi recursion, it does not start at the original starting vector v_1 but at w_1 , which is a linear combination of all the vectors computed during the whole rational Krylov algorithm.

In a practical implementation, this is combined with locking, purging, and implicit restart. First run shifted and inverted Arnoldi with the first shift σ_1 . When an appropriate number of eigenvalues around σ_1 have converged, say after m steps, lock these converged eigenvalues and purge those that are altogether outside the interesting region, leaving a ($j \leq m$)-step Arnoldi recursion (8.18). Then introduce the new shift σ_2 and follow the description in this section to get a new basis W_{j+1} which replaces V_{j+1} . Start at the new shift by operating on the last vector of this new basis

$$r := (A - \sigma_2 B)^{-1} B v_{j+1}$$

and get the next basis vector v_{j+2} in the Arnoldi recursion (8.18) with the new shift σ_2 . Continue until we get convergence for a set of eigenvalues around σ_2 , and repeat the same procedure with new shifts until either all interesting eigenvalues have converged or all the shifts in the prescribed frequency range have been used.

ALGORITHM 8.3: Rational Krylov Subspace Method for GNHEP

- (1) start with σ_1 starting shift, v_1 unit starting vector, basis size $j = 1$
- (2) for $i = 1, 2, \dots$ until convergence
 - (3) expand j -step Arnoldi recursion to m steps:

$$(A - \sigma_i B)^{-1} B V_m = V_{m+1} H_{m+1,m}$$
 - (4) compute Ritz values, lock, and purge
 - (5) determine new shift σ_{i+1} in interesting region
 - (6) factorize $I_{j+1,j} + (\sigma_{i+1} - \sigma_i) H_{j+1,j} = Q_{j+1,j+1} R_{j+1,j}$
 - (7)
$$H_{j+1,j} := \begin{bmatrix} P_j^* & 0 \\ 0 & 1 \end{bmatrix} Q_{j+1,j+1}^* H_{j+1,j} R_{j,j}^{-1} P_j$$
 - (8)
$$V_{j+1} := V_{j+1} Q_{j+1,j+1} \begin{bmatrix} P_j & 0 \\ 0 & 1 \end{bmatrix}$$

Notice that in step (3) we make just $m - j$ operations with the shifted and inverted matrix, the first j basis vectors are those saved from the previous shift σ_{i-1} . The lock, purge, and deflation operations in step (4) are very similar to those for implicitly restarted Arnoldi as described in §7.6; we have actually used a nonhermitian version of thick restart [463]. In step (5) we choose the new shift σ_{i+1} as a value in the complex plane where we are interested in studying the behavior of the matrix pencil (8.17). However, a choice too close to an eigenvalue will result in a nearly singular matrix $A - \sigma_{i+1} B$. Moreover, a new shift at a nearly converged Ritz value will make the upper triangular factor $R_{j,j}$ in step (6) close to singular.

When the second matrix B in the original pencil (8.17) is positive definite, we may choose to make the basis $V_j B$ orthogonal. If then A is also Hermitian, so is $H_{j,j}$, and we may work with tridiagonal matrices; see [322].

This formulation of rational Krylov is different from the one used earlier [378], where the same set of basis vectors V_j is used throughout and the pencil (8.17) is transformed into two Hessenberg matrices. We have found that this new formulation gives a more natural way to continue with a new shift and signal when we risk losing accuracy due to linear dependence.

See the report [379] for a numerical example from model reduction.

8.6 Symmetric Indefinite Lanczos Method

Z. Bai, T. Ericsson, and T. Kowalski

In this section, we present a Lanczos method for solving the generalized eigenvalue problem

$$Ax = \lambda Bx, \quad (8.21)$$

where matrices A and B are real or complex symmetric but neither A nor B nor a combination $\alpha A + \beta B$ for scalars α and β is positive definite. $A - \lambda B$ is called a *symmetric indefinite matrix pencil*.

Such eigenvalue problems come from various applications, such as the linearization of a certain quadratic eigenvalue problem, which often arises in the modeling of damped structural systems; see §9.2.

Formally, the symmetric Lanczos algorithm may be used to compute some eigenpairs, since the matrix $B^{-1}A$ is symmetric with respect to the B inner product.² The three-term recurrence still holds with respect to the B inner product in this more general situation. The algorithm is referred to as a *symmetric indefinite Lanczos method*. The main trouble with this method is that the basis vectors are orthogonal with respect to an indefinite inner product, so there is no assurance that they will be linearly independent. The algorithm could occasionally fail due to a breakdown. Nevertheless, this is an attractive way to solve the problem because of potentially significant savings in memory requirement and floating point operations.

8.6.1 Some Properties of Symmetric Indefinite Matrix Pairs

The symmetry of A and B is purely an algebraic property and is not sufficient to ensure any of the special mathematical properties enjoyed by a definite matrix pencil, such as those discussed in §2.3. In fact, it can be shown that any real square matrix F may be written as $F = AB^{-1}$ or $F = B^{-1}A$, where A and B are suitable symmetric matrices; for example, see [353].

The eigenvalues of a definite matrix pencil are all real, but an indefinite pencil may have complex eigenvalues. For example, when

$$A = \begin{bmatrix} 0 & 10 \\ 10 & 20 \end{bmatrix}, \quad B = \begin{bmatrix} -1 & 0 \\ 0 & 10 \end{bmatrix},$$

the eigenvalues of $\{A, B\}$ are the complex conjugate pair $1 \pm 3i$. A further distinction from the definite case is that an indefinite matrix pencil may not have a complete set of eigenvectors. As an example, consider the pencil

$$\begin{bmatrix} 0 & 1 \\ 1 & 2 \end{bmatrix} - \lambda \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix},$$

which has one eigenvalue $\lambda = 1$ (multiplicity 2) and only one eigenvector, $x = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$.

In §2.3, we know that when B is positive definite we can find a matrix of eigenvectors X and a diagonal matrix of eigenvalues Λ such that $AX = BX\Lambda$ with $X^T BX = I$. The B inner product $(x, y)_B$ forms a true inner product and $(x, x)_B^{1/2}$ is a norm.

When B is indefinite and nonsingular and if $B^{-1}A$ is not defective (i.e., no eigenvectors are missing), we can find a full set of eigenvectors, X . The equation $AX = BX\Lambda$ still holds, and the eigenvectors can be chosen so that $X^T BX = J$, where J is a diagonal matrix with 1 and -1 on the diagonal (note that it is transpose, not conjugate transpose, even though some vectors can be complex). The B inner product $(x, y)_B$ is an indefinite inner product or pseudo-inner product, and $(x, x)_B$ can be used for normalizing purposes. Unlike the positive definite case, there is a set of vectors having pseudolength zero (as measured by $(x, x)_B$). In fact, it is possible for an eigenvector x to satisfy $x^T Ax = x^T Bx = 0$. This implies that the Rayleigh quotient

$$\rho = \frac{x^T Ax}{x^T Bx}$$

²The B inner product between two vectors x and y is defined as $(x, y)_B = y^* Bx$. If B is symmetric and indefinite, then $(x, y)_B$ is a pseudo-inner product. It violates the condition $(x, x)_B \geq 0$ for all x in the definition of a standard inner product.

is undefined. This phenomenon is illustrated by the diagonal matrices

$$A = \text{diag}(-1, 1, 1), \quad B = \text{diag}(1, 1, -1).$$

The vector $x = [1 \ 0 \ 1]^T$ is an eigenvector of $\{A, B\}$ corresponding to the multiple eigenvalue $\lambda = -1$, but $x^T A x = x^T B x = 0$.

The analogy between the definite and the indefinite cases can be taken further. When B is positive definite, \tilde{u} is an approximate eigenvector, and $\tilde{\lambda}$ is an approximate eigenvalue, we have the standard residual bound:

$$|\lambda - \tilde{\lambda}| \leq \frac{\|(A - \tilde{\lambda}B)\tilde{u}\|_{B^{-1}}}{\|\tilde{u}\|_B},$$

where $\|\cdot\|_{B^{-1}}$ is the norm with respect to B^{-1} , i.e., $\|x\|_{B^{-1}} = (x^* B^{-1} x)^{1/2}$ (similarly for the other norm). Also see §5.7. When B is indefinite and nonsingular, $X^T B X = J$, and $B^{-1} A$ is not defective, one can prove that

$$|\lambda - \tilde{\lambda}| \leq \frac{\|(A - \tilde{\lambda}B)\tilde{u}\|_{\tilde{X}X^T}}{\|\tilde{u}\|_{(XX^*)^{-1}}};$$

note that $B^{-1} = XX^*$ in the definite case. A similar bound is given by

$$|\lambda - \tilde{\lambda}| \leq \frac{\|(A - \tilde{\lambda}B)\tilde{u}\|_2}{\|\tilde{u}\|_2} \|B^{-1}\|_2 \|X\|_2 \|X^{-1}\|_2.$$

Even if these bounds are not computable, they imply that a small residual is good. When B is singular there are similar bounds; see [161].

If both A and B are singular, or close to singular, worse problems may occur. Assume that there is a nonzero vector z such that $Az = Bz = 0$; then *any* complex number λ is an eigenvalue. A more general case is illustrated in the example below:

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 0 & 1 \\ 0 & -1 & 1 \\ 1 & 1 & 0 \end{bmatrix}.$$

The characteristic polynomial $\det(A - \lambda B)$ is identically equal to zero, so any complex number is an eigenvalue; we have a singular matrix pencil. The null spaces of A and B are one-dimensional and are spanned by $z_a = [0 \ 0 \ 1]^T$ and $z_b = [-1 \ 1 \ 1]^T$, respectively. Note that intersection between the null spaces is just the zero vector. The singularity of the problem implies that $z_a^T B z_a = 0$ (and $z_b^T A z_b = 0$). To solve a singular problem using numerical methods the singularity must be removed. It is a hopeless task to attack the original problem, as can be seen by perturbing A and B . Set $a_{3,3} = \epsilon$ and add δ to $b_{1,1}$, then $\det(A - \lambda B) = \delta \lambda^3 + \epsilon(\lambda - 1)(\lambda(1 + \delta) - 1)$. So if ϵ or δ is nonzero this is no longer a singular pencil. Taking $\delta = 0$ then for *any* $\epsilon > 0$ we have the eigenvalues 1, 1, and ∞ . If $\delta > 0$ and $\epsilon = 0$ the eigenvalues are 0, 0, and 0. When $\delta = \epsilon \rightarrow 0$ the eigenvalues are roughly 0.56984 and 0.21508 ± 1.30714 . Note that the perturbations can be made *arbitrarily* small. A general discussion about singular pencils can be found in §8.7.

In practice, if we have a singular pencil, $A - \lambda B$ is singular for any λ , and a good LU factorization routine should give a warning (when used in (a) and (b) in the following

§8.6.2). Another sign of a singular pencil is that the eigenvalue routine produces some random eigenvalues at each run on the same problem.

8.6.2 Algorithm

In order to use a Lanczos procedure to solve the generalized eigenproblem (8.21), we first convert it to an equivalent standard eigenvalue problem. Several transformations are possible; the most common ones are listed below.

- (a) If one wants a few of the eigenvalues of $\{A, B\}$ which are largest in magnitude, and if the matrix B is nonsingular, we may multiply by B^{-1} to obtain

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

The matrix $B^{-1}A$ is symmetric with respect to A or B .

- (b) If one wants a few of the eigenvalues closest to a given value σ , we apply the SI to obtain

$$((A - \sigma B)^{-1}B)x = (\lambda - \sigma)^{-1}x.$$

In particular, if the smallest eigenvalues in magnitude are desired, one could choose the shift value $\sigma = 0$. Note that the matrix $(A - \sigma B)^{-1}B$ is symmetric with respect to B or $(A - \sigma B)$.

We will focus on case (b) in the rest of this section and write the transformed problem as

$$H^{-1}Bx = \mu x,$$

where

$$H = A - \sigma B \quad \text{and} \quad \mu = \frac{1}{\lambda - \sigma}.$$

Note that the eigenvalues of $\{A, B\}$ may be complex even when A and B are real, so it may be necessary to use a value of σ which is complex. The Lanczos recursion may still be implemented using real arithmetic even if a complex shift is used; see [362].

A symmetric indefinite Lanczos algorithm template is presented in Algorithm 8.4. The generated Lanczos vectors $\{q_j\}$ and scalars $\{\alpha_j, \beta_j, \omega_j\}$ satisfy the following governing equations:

$$(H^{-1}B)Q_j = Q_j(\Omega_j^{-1}T_j) + \frac{\beta_{j+1}}{\omega_{j+1}}q_{j+1}e_j^T, \quad (8.22)$$

where

$$T_j = \begin{bmatrix} \alpha_1 & \beta_2 & & \\ \beta_2 & \alpha_2 & \ddots & \\ \ddots & \ddots & \ddots & \beta_j \\ & \beta_j & \alpha_j & \end{bmatrix} \quad \text{and} \quad \Omega_j = \text{diag}(\omega_1, \omega_2, \dots, \omega_j).$$

The Lanczos vectors $Q_j = [\begin{array}{cccc} q_1 & q_2 & \cdots & q_j \end{array}]$ are B orthogonal, i.e.,

$$Q_j^T B Q_j = \Omega_j, \quad Q_j^T B q_{j+1} = 0,$$

and are normalized to have unit Euclidean norm, i.e., $\|q_j\|_2 = 1$.

The reduced eigenvalue problem is

$$T_j u_i^{(j)} = \theta_i^{(j)} \Omega_j u_i^{(j)}.$$

It inherits the same numerical difficulties as the original problem. For example, Ritz value $\theta_i^{(j)}$ could be complex, and even defective. In other words, it may belong to a nondiagonal Jordan block of $\Omega_j^{-1} T_j$.

Once $\{\theta_i^{(j)}, u_i^{(j)}\}$ are computed, the right and left Ritz vectors are defined as

$$x_i^{(j)} := Q_j u_i^{(j)} \quad \text{and} \quad y_i^{(j)} := B \bar{Q}_j \bar{u}_i^{(j)} = B \bar{x}_i^{(j)},$$

respectively. The quantities $\{\theta_i^{(j)}, x_i^{(j)}, y_i^{(j)}\}$ are referred to as *Ritz triplets* of the matrix $H^{-1}B$.

The (right) residual vectors $r_i^{(j)}$ corresponding to the Ritz pairs $\{\theta_i^{(j)}, x_i^{(j)}\}$ are

$$r_i^{(j)} = (H^{-1}B)x_i^{(j)} - \theta_i^{(j)}x_i^{(j)} = \gamma_i^{(j)}q_{j+1}, \quad (8.23)$$

where

$$\gamma_i^{(j)} = \frac{\beta_{j+1}}{\omega_{j+1}} (e_j^T u_i^{(j)}).$$

Note that the last equality in (8.23) is obtained by multiplying equation (8.22) on the right by $u_i^{(j)}$ and the definition of $x_i^{(j)}$. Note that the left residual vectors $(s_i^{(j)})^*$ are related to the right residual vectors $r_i^{(j)}$ by $(s_i^{(j)})^* = (r_i^{(j)})^T B$. It follows that

$$\|r_i^{(j)}\|_2 = |\gamma_i^{(j)}| \quad (8.24)$$

and

$$\|s_i^{(j)}\|_2 = |\gamma_i^{(j)}| \|Bq_{j+1}\|_2. \quad (8.25)$$

Note that the vector Bq_{j+1} is directly available in every iteration of the symmetric indefinite Lanczos algorithm. No extra call on B is necessary. Therefore, one of the attractive features of the Lanczos algorithm is that the residual vectors can be calculated without explicitly computing Ritz vectors $x_i^{(j)}$ and $y_i^{(j)}$.

ALGORITHM 8.4: Symmetric Indefinite Lanczos Method

```

(1) choose a starting vector  $q$ 
(2) compute  $\beta_1 = \|q\|_2$ 
(3) scale  $q_1 = q/\beta_1$ 
(4)  $w = Bq_1$ 
(5)  $\omega_1 = w^T q_1$ 
(6) for  $j = 1, 2, \dots$  until convergence
    (7) solve  $Hr = w$  for  $r$ 
    (8)  $r := r - q_{j-1}(\beta_j/\omega_{j-1})$  for  $j > 1$ 
    (9)  $\alpha_j = w^T r$ 
    (10)  $r := r - (\alpha_j/\omega_j)q_j$ 
    (11)  $\tau = \|r\|_2$ 
    (12) if  $\tau = 0$ , stop
    (13) reorthogonalize if needed
    (14)  $q_{j+1} = r/\tau$ 
    (15)  $w = Bq_{j+1}$ 
    (16)  $\omega_{j+1} = w^T q_{j+1}$ 
    (17) if  $\omega_{j+1} = 0$ , stop
    (18)  $\beta_{j+1} = \tau\omega_{j+1}$ 
    (19) solve reduced problem and check for convergence.
    (20) end for
(21) compute approximate eigenvectors  $x_i^{(j)}$ .

```

We now comment on some steps of Algorithm 8.4:

- (1) The ideal initial vector is one which is a linear combination of the eigenvectors corresponding to the desired eigenvalues. If an approximation of such a vector is not available, then a random vector may be used. The initial vector r is often preprocessed by multiplying it by $H^{-1}B$. This preprocessing step was suggested in [161] and is essential when B is singular; see §8.6.4 for details.
- (4) and (15) The matrix-vector multiplication Bq_j should be performed by a user-provided subroutine which can take advantage of the data structure of the matrix B .
- (7) The linear system of equations $Hr = w$ must be solved for r . Let

$$H = \mathcal{L}\mathcal{D}\mathcal{L}^T$$

be the sparse LDL^T factorization (see §10.3) computed before the for-loop. Then the vector r can be efficiently computed in three steps:

$$(7)' \quad \begin{aligned} &\text{solve } \mathcal{L}u = w \text{ for } u \\ &\text{solve } \mathcal{D}v = u \text{ for } v \\ &\text{solve } \mathcal{L}^T r = v \text{ for } r \end{aligned}$$

- (12) When $\|r\|_2 = 0$, it indicates that all of the eigenvalues of the $j \times j$ pencil $T_j - \lambda\Omega_j$ are also the eigenvalues of $H^{-1}B$, and the columns of Q_j span an invariant

subspace. One can either exit the algorithm or continue the algorithm by setting the value of β_j to zero and selecting r to be a random vector which is B orthogonal to the columns of Q_j .

- (13) In finite precision arithmetic, the vectors $\{q_j\}$ are no longer B orthogonal. The schemes for reorthogonalization which are discussed for the standard symmetric Lanczos procedure may be extended to the symmetric indefinite algorithm; see §4.4 for details.
- (14) In the symmetric Lanczos algorithm (see §4.4), the Lanczos vectors are scaled to be B orthonormal, $q_j^T B q_j = 1$. However, in the symmetric indefinite algorithm, the vectors are scaled so that

$$\|q_j\|_2 = 1.$$

There are two reasons for the change in scaling, both of which have to do with the indefiniteness of B . The first reason is that even when A and B are real, scaling the Lanczos vectors so that $q_j^T B q_j = 1$ may involve complex arithmetic. (By keeping track of the sign of $q_j^T B q_j$ it is possible to avoid complex arithmetic but still use $|q_j^T B q_j|^{1/2}$ as a norm. However, this algorithm is less stable.) The second reason is that when B is indefinite, not every subspace has a B -orthonormal basis, and even if it does, it may be highly ill-conditioned [417, 20]. The choice of scaling does not remedy the possibility of an ill-conditioned basis of Lanczos vectors. However, it does provide a convenient means of detecting when the basis is becoming ill-conditioned, specifically, a tiny value of $|\omega_{j+1}|$ [357].

- (17) If $\omega_{j+1} = 0$, the Lanczos procedure suffers a breakdown similar to the breakdown which occurs in the non-Hermitian Lanczos procedure; see §7.8. The same remedies developed for the non-Hermitian Lanczos procedure, such as look-ahead [364, 178] and adaptive blocking [29, 298], can be employed.
- (19) The approximate eigenvalues of the matrix pencil $A - \lambda B$ are determined by solving the reduced $j \times j$ generalized eigenvalue problem

$$T_j u = \theta \Omega_j u.$$

There are several algorithms available for solving this problem. If j is small, one could apply the standard QZ algorithm to $\{T_j, \Omega_j\}$ or the QR algorithm to $\Omega_j^{-1} T_j$. However, neither of these algorithms take advantage of the structure of $\{T_j, \Omega_j\}$. A good survey of algorithms which exploit the symmetry of $\{T_j, \Omega_j\}$ is contained in [357], and some more recent work can be found in [406, 407].

For even moderate values of j (e.g., $j \approx 100$), solving the reduced problem becomes computationally significant. For long Lanczos runs, it is recommended to solve the reduced problem periodically, perhaps once in every five or ten Lanczos iterations.

A proper stopping criterion for the inner loop of the symmetric indefinite Lanczos method is

$$\max \left\{ \|r_i^{(j)}\|_2, \|s_i^{(j)}\|_2 \right\} = |\gamma_i^{(j)}| \cdot \max \{1, \|Bq_{k+1}\|_2\} \leq \epsilon,$$

where ϵ is a user-given tolerance value. Note that Bq_{k+1} is available at the end of a Lanczos run, and its Euclidean norm can be computed at the cost of one

dot product. No extra call on B is necessary for that term. An elaborate testing procedure for the convergence is discussed in the following §8.6.3.

- (21) Compute the approximate eigenvectors only when the corresponding Ritz values have converged according to the test described below.

8.6.3 Stopping Criteria and Accuracy Assessment

Since the eigenproblem of a symmetric indefinite pencil does not have any special mathematical properties, we may refer to techniques developed for general non-Hermitian matrices to assess the quality of the approximate eigentriplets of an indefinite pencil. Nonetheless, we can take advantage of the symmetry to simplify the analysis and subsequently the error bound. The simplification stems from the simple relation between the right and left eigenvectors of $H^{-1}B$. If x is a right eigenvector of $H^{-1}B$, then $y = B\bar{x}$ is the corresponding left eigenvector.

To assess the accuracy of a Ritz triplet $\{\theta_i^{(j)}, x_i^{(j)}, y_i^{(j)}\}$, by the discussion in §7.8, it is known that there is a matrix $E_i^{(j)}$ such that

$$(H^{-1}B - E_i^{(j)})x_i^{(j)} = \theta_i^{(j)}x_i^{(j)} \quad \text{and} \quad (y_i^{(j)})^*(H^{-1}B - E_i^{(j)}) = \theta_i^{(j)}(y_i^{(j)})^*,$$

where

$$\|E_i^{(j)}\|_2 = \max \left\{ \frac{\|r_i^{(j)}\|_2}{\|x_i^{(j)}\|_2}, \frac{\|s_i^{(j)}\|_2}{\|y_i^{(j)}\|_2} \right\}.$$

In fact, $\|E_i^{(j)}\|_2$ is the optimal backward error. By the standard first-order perturbation expansion or the error bound presented in §7.13, there is an eigenvalue μ of $H^{-1}B$, such that

$$\begin{aligned} |\mu - \theta_i^{(j)}| &\lesssim \frac{\|x_i^{(j)}\|_2 \|y_i^{(j)}\|_2}{|(y_i^{(j)})^* x_i^{(j)}|} \cdot \|E_i^{(j)}\|_2 \\ &= \frac{1}{|(y_i^{(j)})^* x_i^{(j)}|} \cdot \max \left\{ \|r_i^{(j)}\|_2 \|y_i^{(j)}\|_2, \|s_i^{(j)}\|_2 \|x_i^{(j)}\|_2 \right\} \\ &= \frac{1}{|(u_i^{(j)})^T \Omega_j u_i^{(j)}|} \cdot |\gamma_i^{(j)}| \cdot \max \left\{ \|B\bar{x}_i^{(j)}\|_2, \|x_i^{(j)}\|_2 \|Bq_{j+1}\|_2 \right\}, \end{aligned}$$

where we have made use of equations (8.24), (8.25), and the equality

$$(y_i^{(j)})^* x_i^{(j)} = (u_i^{(j)})^T Q_j^T B Q_j u_i^{(j)} = (u_i^{(j)})^T \Omega_j u_i^{(j)}.$$

Furthermore, to avoid a bound which explicitly involves the Ritz vectors $x_i^{(j)}$, we note that

$$\|x_i^{(j)}\|_2 = \|Q_j u_i^{(j)}\|_2 \leq \|Q_j\|_F \leq \sqrt{j}$$

and

$$\|y_i^{(j)}\|_2 = \|B Q_j \bar{u}_i^{(j)}\|_2 \leq \|B Q_j\|_F,$$

where it is assumed that $\|u_i^{(j)}\| = 1$. Recall that the columns of Q_j are normalized to 1. The value of

$$\|B Q_j\|_F^2 = \sum_{i=1}^j \|B q_i\|_2^2$$

may be updated in each Lanczos iteration. Therefore the error $|\mu - \theta_i^{(j)}|$ can also be bounded by

$$|\mu - \theta_i^{(j)}| \lesssim \frac{1}{|(u_i^{(j)})^T \Omega_j u_i^{(j)}|} \cdot |\gamma_i^{(j)}| \cdot \max \left\{ \|BQ_j\|_F, \sqrt{j} \|Bq_{j+1}\|_2 \right\},$$

which does not explicitly use the Ritz vector $x_i^{(j)}$. In conclusion, one may use

$$\frac{1}{|(u_i^{(j)})^T \Omega_j u_i^{(j)}|} \cdot |\gamma_i^{(j)}| \cdot \max \left\{ \|BQ_j\|_F, \sqrt{j} \|Bq_{j+1}\|_2 \right\} \quad (8.26)$$

as a provisional error estimate in the inner loop of Algorithm 8.4 (i.e., step (19)). Only when the required number of Ritz values $\theta_i^{(j)}$ have passed this test, and not before, may the Ritz vectors $x_i^{(j)}$ be computed. At that point the more precise factor,

$$\max \left\{ \|B\bar{x}_i^{(j)}\|_2, \|x_i^{(j)}\|_2 \|Bq_{j+1}\|_2 \right\},$$

may be computed at the extra cost of forming $Bx_i^{(j)}$.

The Ritz values $\theta_i^{(j)}$ approximate the eigenvalues of $H^{-1}B$, while the quantities $\sigma + 1/\theta_i^{(j)}$ are approximations to the eigenvalues λ of the original problem (8.21). We can estimate $|\lambda - (\sigma + 1/\theta_i^{(j)})|$ using the bound above and

$$\left| \lambda - \left(\sigma + \frac{1}{\theta_i^{(j)}} \right) \right| = \left| \left(\sigma + \frac{1}{\mu} \right) - \left(\sigma + \frac{1}{\theta_i^{(j)}} \right) \right| = \frac{|\mu - \theta_i^{(j)}|}{|\theta_i^{(j)}\mu|} \approx \frac{|\mu - \theta_i^{(j)}|}{|\theta_i^{(j)}|^2}.$$

8.6.4 Singular B

Special care must be taken when the symmetric indefinite Lanczos procedure is implemented with a singular B matrix. Here we limit our discussion to the case where $H^{-1}B$ is diagonalizable. If $H^{-1}B$ is defective the problem is more complicated and the reader is referred to [161] for details.

When B is singular, $\{A, B\}$ has infinite eigenvalues; the eigenvectors corresponding to the infinite eigenvalues lie in the null space of B , while the eigenvectors corresponding to the eigenvalues of interest (i.e., the finite eigenvalues) belong to the range of $H^{-1}B$. Consider the following example:

$$A = \begin{bmatrix} \alpha & \beta \\ \beta & \gamma \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}.$$

Then

$$H^{-1}B = (A - \sigma B)^{-1}B = \frac{1}{\delta} \begin{bmatrix} \gamma & 0 \\ -\beta & 0 \end{bmatrix},$$

where $\delta = (\alpha - \sigma)\gamma - \beta^2$. The eigenvalue of $H^{-1}B$ of interest is γ/δ and the corresponding eigenvector $\begin{bmatrix} \gamma & -\beta \end{bmatrix}^T$ (the zero eigenvalue corresponds to an infinite eigenvalue of the original problem).

Note that the eigenvector has a component ($-\beta$ in the example) in the null space of B and that the range space of $H^{-1}B$ is independent of σ . It is important that the

Lanczos starting vector q lies in this range space since components outside the space may grow during the Lanczos iteration and contaminate the Ritz vectors.

It is essential that Algorithm 8.4 produce Ritz vectors which lie entirely in the range of $H^{-1}B$. To this end, we recommend a preprocessing step as well as a postprocessing step for Algorithm 8.4.

The initial vector q in Algorithm 8.4 should lie in the range of $H^{-1}B$. For example, one could use the product of $H^{-1}B$ and a random vector. In exact arithmetic, if the initial vector q lies in the range of $H^{-1}B$, then all of the Lanczos vectors $\{q_j\}$ will belong to the range of $H^{-1}B$ [340].

However, in finite precision arithmetic, preprocessing the initial vector q may not be enough to ensure that the Ritz vectors lie in the range of $H^{-1}B$. As the Lanczos iteration progresses, roundoff errors cause the Lanczos vectors to acquire unwanted components in the null space of B . There is no cheap way to eliminate these components from the Lanczos vectors, even though the size of the components may be monitored via an inexpensive recurrence derived in [340]. Fortunately, there is a simple postprocessing scheme for purging the undesired components of Ritz vectors which lie in the null space of B . The idea is to replace the Ritz vector $x_i^{(j)}$ with a multiple of $H^{-1}Bx_i^{(j)}$, which clearly belongs to the range of $H^{-1}B$. In practice, there is no need to explicitly compute the vector $H^{-1}Bx_i^{(j)}$; we may take advantage of quantities which are readily available in the Lanczos procedure to compute a vector parallel to $H^{-1}Bx_i^{(j)}$. From equation (8.23), it follows that

$$H^{-1}Bx_i^{(j)} = \theta_i^{(j)}x_i^{(j)} + \gamma_i^{(j)}q_{j+1}.$$

Hence a purified Ritz vector, $(1/\theta_i^{(j)})H^{-1}Bx_i^{(j)}$, can be cheaply computed by simply updating $x_i^{(j)}$ with a multiple of q_{j+1} :

$$x_i^{(j)} := x_i^{(j)} + \frac{\gamma_i^{(j)}}{\theta_i^{(j)}}q_{j+1}. \quad (8.27)$$

This technique was originally presented in [162] as a mechanism for improving the quality of the Ritz vectors, and it was later shown in [340] to have the purifying effect required here. Numerical experience has shown that the updated vector lies almost entirely in the range of $H^{-1}B$ and generally provides a better eigenvector approximation. This scheme is also recommended for use in the indefinite case.

8.6.5 Software Availability

See the book's homepage, ETHOME, for a prototype MATLAB implementation of the method described in this section.

8.6.6 Numerical Examples

We present several examples which illustrate some of the features of the symmetric indefinite Lanczos procedure described above. In particular, the examples below illustrate three points:

- When B is singular, premultiplying the starting vector by $H^{-1}B$ provides a dramatic improvement in the quality of the computed Ritz pairs.

- When B is singular, the postprocessing step described in equation (8.27) may further improve the Ritz pairs.
- Partial reorthogonalization provides substantial savings over complete reorthogonalization.

In each of our example problems, the symmetric indefinite eigenvalue problem stems from the linearization of a quadratic eigenvalue problem which arises in structural dynamics. The matrices come from one of two finite element models generated by the structural engineering package MSC/NASTRAN [274]. All of the examples were run on a Sun UltraSPARC with a 336 MHz processor using a simple MATLAB implementation of Algorithm 8.4. Ritz values were accepted as “converged” based on the error bound derived in §8.6.3. The bound was found to be pessimistic in most cases, and a reliable tighter bound is still an open research topic.

The first model is an acoustics problem representing a speaker box. Two sets of matrices have been generated from the model. In the first set, the linearized problem $\{A, B\}$ has order 1076. In a smaller second set of matrices, modal reduction has been employed and the associated matrices $\{A, B\}$ have order 668.

The second model represents a shaft on bearing supports with a translational viscous damper attached at the midpoint of the shaft. By controlling the mesh size, two sets of matrices have been produced for this model. With a finer mesh, the order of $\{A, B\}$ is 800, while a coarser mesh yields matrices $\{A, B\}$ with order 160. In each case, the resulting B matrix is singular with this model.

Example 8.6.1. The first example demonstrates the importance of the choice of starting vector when the B matrix is singular. The matrices used in this example come from the shaft model using a fine mesh, and the order of the matrices $\{A, B\}$ is 800. A shift of $\sigma = 100i$ was used and complete reorthogonalization was employed. The following table illustrates one ill effect which may result when the Lanczos vectors do not belong to the range of $H^{-1}B$. The symmetric indefinite Lanczos procedure was executed twice with an initial vector of all ones. In the first run, the initial vector is preprocessed by multiplying it by $H^{-1}B$, while in the second run, no preprocessing is used. The preprocessing step dramatically improves the quality of the Ritz pairs, as measured by the residuals $\|A\tilde{x} - \tilde{\lambda}B\tilde{x}\|$, by ensuring that the Lanczos vectors, and hence the Ritz vectors, belong to the range of $H^{-1}B$.

Ritz value $\tilde{\lambda}$	$\ A\tilde{x} - \tilde{\lambda}B\tilde{x}\ $ (preprocessed initial vector)	$\ A\tilde{x} - \tilde{\lambda}B\tilde{x}\ $ (no preprocessing)
$-4.09e-06 + 5.63e+01$	$1.88e-08$	$3.05e-03$
$-4.15e-06 - 5.63e+01$	$1.45e-07$	$1.10e-02$
$-1.30e-04 + 3.55e+02$	$6.09e-07$	$5.48e-02$
$-1.31e-04 - 3.55e+02$	$5.24e-07$	$9.72e-02$
$-8.58e-04 + 1.00e+03$	$8.57e-07$	$2.85e-01$
$-8.65e-04 - 1.00e+03$	$8.12e-07$	$3.50e-01$

Example 8.6.2. Even when the initial vector is premultiplied by $H^{-1}B$, the Ritz vectors may be further improved by implementing the postprocessing step shown in (8.27). The following example represents the same shaft as in Example 8.6.1, but with a coarser

grid so that the order of $\{A, B\}$ is 160. A shift of $\sigma = 1.e5i$ was used and full orthogonality was maintained. The following table shows that the postprocessing step improves the direct residuals by up to four orders of magnitude.

Ritz value $\tilde{\lambda}$	$\ A\tilde{x} - \tilde{\lambda}B\tilde{x}\ $ (with postprocessing)	$\ A\tilde{x} - \tilde{\lambda}B\tilde{x}\ $ (no postprocessing)
$-1.96e-04 + 1.03e+05i$	$5.04e-08$	$5.04e-08$
$-1.30e-02 + 1.06e+05i$	$4.11e-08$	$4.11e-08$
$-7.16e-03 + 9.28e+04i$	$4.15e-08$	$4.18e-08$
$-7.60e-02 + 1.11e+05i$	$8.41e-08$	$8.51e-05$
$-1.01e-01 + 8.66e+04i$	$1.09e-07$	$5.06e-03$

Example 8.6.3. We adapted the partial reorthogonalization schemes presented in §7.9 for our symmetric indefinite Lanczos implementation. The following table shows that partial reorthogonalization provides substantial, though not dramatic, savings over complete reorthogonalization with comparable accuracy. The time listed in the table is the CPU time spent during the reorthogonalization stage with either the full reorthogonalization strategy or the partial reorthogonalization scheme. The CPU times indicate that complete reorthogonalization is about twice as expensive as partial reorthogonalization in terms of computation.

Model	Speaker	Speaker (reduced model)	Shaft
Order $\{A, B\}$	1076	668	800
Lanczos iterations	91	96	110
Number of reorthogonalizations for semiorthogonality	25	31	40
Reorthogonalization time (partial)	53.7	25.7	46.1
Reorthogonalization time (full)	126.1	53.2	87.0

8.7 Singular Matrix Pencils

B. Kågström

One difference at least in theory between the standard eigenvalue problem $Ax = \lambda x$ and the generalized eigenvalue problem $Ax = \lambda Bx$ discussed in the previous sections is the appearance of infinite eigenvalues whenever B is singular. The standard method for solving dense generalized eigenvalue problems is the QZ algorithm; see §8.2. However, there is more to the story. For example, if A and B have a common null space, then the generalized eigenvalue problem is ill posed in the sense that arbitrary small perturbations may change the eigenvalues completely. In general, the QZ algorithm is not capable of handling this type of problem in a reliable way. It is necessary to apply a regularization technique by allowing a deflation criterion for range or null space separations in finite precision and thereby make it possible to compute the eigenvalues of a nearby problem. Typically, the distance from the regularized nearby problem to the original problem is proportional to the size of the deflation tolerance used.

In this section we give an introduction to the theory, algorithms, and software for solving the most general form of the $Ax = \lambda Bx$ problem, including the case when A

and B are rectangular matrices. The software computes a generalization of the Schur canonical form to matrix pairs $\{A, B\}$ called GUPTRI (generalized upper triangular) form [121, 122]:

$$P^*(A - \lambda B)Q = \begin{bmatrix} A_r - \lambda B_r & * & * \\ 0 & A_{reg} - \lambda B_{reg} & * \\ 0 & 0 & A_l - \lambda B_l \end{bmatrix}, \quad (8.28)$$

with the following properties: it separates the regular and singular parts of the problem. $A_{reg} - \lambda B_{reg}$ is the regular part and it includes all eigenvalues. The blocks $A_r - \lambda B_r$ and $A_l - \lambda B_l$ in (8.28) correspond to the singular part and have a special block structure. Here $*$ denotes arbitrary conforming submatrices.

Before we make a complete description of the GUPTRI form and present algorithms and software for computing it, we need to introduce some new notation and definitions associated with singular problems. Some of the material presented is by definition rather technical and comprehensive. In our effort to simplify the presentation we also provide several examples that illustrate the nature of singular and nearly singular eigenproblems, the difficulties encountered in dealing with them, and how they can be overcome.

8.7.1 Regular Versus Singular Problems

Let us start by considering the 2×2 generalized eigenvalue problem $Ax = \lambda Bx$, where

$$A = \begin{bmatrix} \alpha_1 & 0 \\ 0 & \alpha_2 \end{bmatrix}, \quad B = \begin{bmatrix} \beta_1 & 0 \\ 0 & \beta_2 \end{bmatrix}.$$

The eigenvalues of $A - \lambda B$ are the pairs (α_1, β_1) and (α_2, β_2) with the associated eigenvectors $x_1 = [1 \ 0]^T$ and $x_2 = [0 \ 1]^T$, respectively. If β_i is nonzero, then $\lambda_i = \alpha_i/\beta_i$ is a finite eigenvalue. Otherwise, if β_i is zero, then $\lambda_i = \infty$ is an eigenvalue of the matrix pair $\{A, B\}$. But what happens if, for example, $\alpha_2 = \beta_2 = 0$? Then $\det(A - \lambda B)$ is zero for all λ , which means that we have a singular eigenvalue problem. In this case we have $Ax_2 = Bx_2 = 0$; i.e., A and B have a common null space. We say that x_2 is an eigenvector for an indeterminate eigenvalue $0/0$. Note that the common null space is a sufficient but not necessary condition to have a singular eigenvalue problem.

The most common generalized eigenvalue problems $Ax = \lambda Bx$ are *regular*, i.e., A and B are square matrices and the characteristic polynomial $p_m(\lambda) = \det(A - \lambda B)$ is only vanishing for a finite number of values, where m denotes the degree of the polynomial. The corresponding $A - \lambda B$ is called a regular matrix pencil. The n eigenvalues of a regular pencil are points in the extended complex plane $\mathcal{C} \cup \infty$. The eigenvalues λ_i are defined as the zeros of $p_m(\lambda)$ and $n - m$ additional ∞ eigenvalues.

An alternative representation of a matrix pencil is the *cross product form*: the set of matrices $\beta A - \alpha B$ where $(\alpha, \beta) \in \mathcal{C}^2$. The mapping $(\alpha, \beta) \mapsto \alpha/\beta$ shows the relation between the eigenvalues of $\beta A - \alpha B$ and $A - \lambda B$. For example, zero and infinite eigenvalues are represented as $(0, \beta)$ and $(\alpha, 0)$, respectively, and can be treated as any other points in \mathcal{C}^2 .

If $\det(A - \lambda B)$ (and $\det(\beta A - \alpha B)$) is identically zero for all λ (and pairs (α, β)), then $A - \lambda B$ is called *singular* and $\{A, B\}$ is a singular matrix pair. Singularity of $\{A, B\}$ is signaled by some $\alpha = \beta = 0$. In the presence of roundoff, α and β may be very small.

In these situations, the eigenvalue problem is very ill-conditioned, and some of the other computed nonzero values of α and β may be indeterminate. Such problems are further discussed and illustrated by examples in §8.7.4. Moreover, rectangular matrix pairs are singular and the corresponding $A - \lambda B$ is a singular pencil.

8.7.2 Kronecker Canonical Form

Just as the Jordan canonical form (JCF) describes the invariant subspaces and eigenvalues of a square matrix A in full detail, there is a *Kronecker canonical form* (KCF) which describes the generalized eigenvalues and generalized eigenspaces of a pencil $A - \lambda B$ in full detail. In addition to Jordan blocks for finite and infinite eigenvalues, the Kronecker form contains singular blocks corresponding to minimal indices of a singular pencil (see below).

The KCF of $A - \lambda B$ exhibits the fine structure elements, including elementary divisors (Jordan blocks) and minimal indices (singular blocks), and is defined as follows [187]. Suppose $A, B \in \mathcal{C}^{m \times n}$. Then there exist nonsingular $P \in \mathcal{C}^{m \times m}$ and $Q \in \mathcal{C}^{n \times n}$ such that

$$P^{-1}(A - \lambda B)Q = \tilde{A} - \lambda \tilde{B} \equiv \text{diag}(A_1 - \lambda B_1, \dots, A_b - \lambda B_b), \quad (8.29)$$

where $A_i - \lambda B_i$ is $m_i \times n_i$. We can partition the columns of P and Q into blocks corresponding to the b diagonal blocks of $\tilde{A} - \lambda \tilde{B}$:

$$P = [P_1, \dots, P_b] \quad \text{and} \quad Q = [Q_1, \dots, Q_b],$$

where P_i is $m \times m_i$, and Q_i is $n \times n_i$, such that

$$(A - \lambda B)Q_i = P_i(A_i - \lambda B_i).$$

Each block $M_i \equiv A_i - \lambda B_i$ must be of one of the following forms:

$$J_j(\alpha), \quad N_j, \quad L_j, \quad \text{or} \quad L_j^T.$$

First we consider

$$J_j(\alpha) \equiv \begin{bmatrix} \alpha - \lambda & 1 & & \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ & & & \alpha - \lambda \end{bmatrix} \quad \text{and} \quad N_j \equiv \begin{bmatrix} 1 & -\lambda & & \\ & \ddots & \ddots & \\ & & \ddots & -\lambda \\ & & & 1 \end{bmatrix}. \quad (8.30)$$

$J_j(\alpha)$ is simply a $j \times j$ Jordan block, and α is called a *finite eigenvalue*. A Jordan block $J_j(\alpha)$ is a cyclic transformation. The range is generated by the j th unit vector e_j —the vectors of the Krylov sequence

$$\{e_j, J_j(\alpha)e_j, J_j(\alpha)^2e_j, \dots, J_j(\alpha)^{j-1}e_j\}$$

span the range. From the cyclicity of the transformation we get the vector chain relations

$$(J_j(\alpha) - (\alpha - \lambda)I)e_{k+1} = e_k \quad \text{for } 1 \leq k \leq j-1.$$

The vector e_{k+1} is said to be a *principal vector of grade $k+1$* . The order j is often referred to as the *height* of the chain. The eigenvectors are principal vectors of grade 1.

N_j is a $j \times j$ block corresponding to an *infinite eigenvalue* of multiplicity j :

$$N_j = \begin{bmatrix} 1 & 0 & & \\ & \ddots & \ddots & \\ & & \ddots & 0 \\ & & & 1 \end{bmatrix} - \lambda \begin{bmatrix} 0 & 1 & & \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ & & & 0 \end{bmatrix}.$$

By exchanging the A -part and B -part of N_j we get a $J_j(0)$ block. So if the KCF of $\{A, B\}$ has an N_j block in its KCF, then $\{B, A\}$ has a $J_j(0)$ block in its KCF and vice versa. This fact is utilized in algorithms for computing canonical structure of matrix pencils.

The $J_j(\alpha)$ and N_j blocks together constitute the *regular structure* of the pencil. All the $A_i - \lambda B_i$ are regular blocks if and only if $A - \lambda B$ is a regular pencil. $\lambda(A, B)$ denotes the eigenvalues of the regular part of $A - \lambda B$ (counted with multiplicities) and is called the *spectrum* of $A - \lambda B$.

The other two types of diagonal blocks are

$$L_j \equiv \begin{bmatrix} -\lambda & 1 & & \\ & \ddots & \ddots & \\ & & -\lambda & 1 \end{bmatrix} \text{ and } L_j^T \equiv \begin{bmatrix} -\lambda & & & \\ 1 & \ddots & & \\ & \ddots & -\lambda & \\ & & & 1 \end{bmatrix}. \quad (8.31)$$

The $j \times (j+1)$ block L_j is called a *singular block of right (or column) minimal index j* . It has a one-dimensional right null space, $[1, \lambda, \dots, \lambda^j]^T$, for any λ , i.e.,

$$\begin{bmatrix} -\lambda & 1 & & \\ -\lambda & 1 & & \\ & \ddots & \ddots & \\ & & -\lambda & 1 \end{bmatrix} \begin{bmatrix} 1 \\ \lambda \\ \vdots \\ \lambda^j \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

Similarly, the $(j+1) \times j$ block L_j^T is a *singular block of left (or row) minimal index j* and has a one-dimensional left null space for any λ . The left and right singular blocks together constitute the *singular structure* of the pencil and appear in the KCF if and only if the pencil is singular. The regular and singular structures define the *Kronecker structure* of a singular pencil.

We end this introductory description by briefly pointing to the relationship between structure information of the KCF and the GUPTRI form (8.28). The block $A_r - \lambda B_r$ contains all information about the right singular blocks and $A_l - \lambda B_l$ contains all information about the left singular blocks. The regular part corresponds to $A_{reg} - \lambda B_{reg}$.

8.7.3 Generic and Nongeneric Kronecker Structures

Although the KCF looks quite complicated in the general case, most matrix pencils have a more simple Kronecker structure. If $A - \lambda B$ is $m \times n$, where $m \neq n$, then for

almost all A and B it will have the same KCF, depending only on m and n . This corresponds to the *generic case* when $A - \lambda B$ has full rank for any complex (or real) value of λ . Accordingly, generic rectangular pencils have no regular part. The generic Kronecker structure for $A - \lambda B$ with $d = n - m > 0$ is

$$\text{diag}(L_\ell, \dots, L_\ell, L_{\ell+1}, \dots, L_{\ell+1}), \quad (8.32)$$

where $\ell = \lfloor m/d \rfloor$, the total number of blocks is d , and the number of $L_{\ell+1}$ blocks is $m \bmod d$ (which is 0 when d divides m) [446, 116]. The same statement holds for $d = m - n > 0$ if we replace $L_\ell, L_{\ell+1}$ in (8.31) by $L_\ell^T, L_{\ell+1}^T$. For example, a generic pencil of size 2×3 has an L_2 block as its KCF.

Square pencils are generically regular; i.e., $\det(A - \lambda B) = 0$ if and only if λ is an eigenvalue. Moreover, the most generic regular pencil is diagonalizable with distinct finite eigenvalues. The generic singular pencils of size $n \times n$ have the Kronecker structures [456]:

$$\text{diag}(L_j, L_{n-j-1}^T), \quad j = 0, \dots, n-1. \quad (8.33)$$

Only if a singular $A - \lambda B$ is rank deficient (for some λ), the associated KCF may be more complicated and possibly include a regular part, as well as right and left singular blocks. This situation corresponds to the *nongeneric* (or *degenerate*) case, which is the real challenge from a computational point of view. Degenerate rectangular pencils have several applications in control theory, for example, to compute the controllable subspace and uncontrollable modes of a linear descriptor system [447, 120].

8.7.4 Ill-Conditioning

Singular pencils may or may not have eigenvalues. Indeed, the generic case corresponds to a singular pencil that has no eigenvalues. Below we illustrate this with two 3 by 3 examples:

$$A_1 - \lambda B_1 = \left[\begin{array}{c|cc} 1 & 0 & 0 \\ \hline 0 & 0 & 0 \\ 0 & 0 & -\lambda \end{array} \right], \quad A_2 - \lambda B_2 = \left[\begin{array}{cc|c} 1 & -\lambda & 0 \\ 0 & 0 & 1 \\ 0 & 0 & -\lambda \end{array} \right].$$

Obviously, $\det(A_1 - \lambda B_1) \equiv 0$ and $\det(A_2 - \lambda B_2) \equiv 0$ for all λ . Although both pencils have the same diagonal elements they have very different canonical forms. Indeed, both pencils are in KCF: $A_1 - \lambda B_1 = \text{diag}(N_1, L_0, L_0^T, J_1(0))$ and $A_2 - \lambda B_2 = \text{diag}(L_1, L_1^T)$. From top to bottom, the diagonal blocks of $A_1 - \lambda B_1$ correspond to N_1 , $\text{diag}(L_0, L_0^T)$, and J_1 . So $A_1 - \lambda B_1$ has a regular part of size 2×2 with eigenvalues at zero ($J_1(0)$) and infinity (N_1) and a singular part of size 1×1 corresponding to one L_0 block (of size 0×1) and one L_0^T block, while $A_2 - \lambda B_2$ is a generic 3×3 singular pencil with no regular part.

If $A - \lambda B$ is upper triangular and a zero element appears on the diagonal, then the pencil is singular. We see that both examples have this property (the (2, 2) entries of A_1 and B_1 are zero as well as for A_2 and B_2). This situation will appear if we apply the QZ algorithm to a square singular pencil in infinite precision. Such a pair $(\alpha, \beta) = (0, 0)$ is called an indeterminate eigenvalue 0/0. In the presence of roundoff, the QZ algorithm may fail to detect and isolate the singularity due to the ill-conditioning of the problem as illustrated below.

The eigenvalue problem for a singular pair is much more complicated than for a regular pair. Consider, for example, the singular pair

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

which has one finite eigenvalue 1 and one indeterminate eigenvalue 0/0 (corresponding to a singular part $\text{diag}(L_0, L_0^T)$). To see that neither the eigenvalue 1 nor the singular part is well determined by the data, consider the slightly perturbed problem

$$A' = \begin{bmatrix} 1 & \epsilon_1 \\ \epsilon_2 & 0 \end{bmatrix}, \quad B' = \begin{bmatrix} 1 & \epsilon_3 \\ \epsilon_4 & 0 \end{bmatrix},$$

where the ϵ_i are tiny nonzero numbers. It follows that $\{A', B'\}$ is regular with eigenvalues ϵ_1/ϵ_3 and ϵ_2/ϵ_4 . Given *any* two complex numbers λ_1 and λ_2 , we can find arbitrary tiny ϵ_i such that $\lambda_1 = \epsilon_1/\epsilon_3$ and $\lambda_2 = \epsilon_2/\epsilon_4$ are the eigenvalues of $\{A', B'\}$. Since, in principle, roundoff could change $\{A, B\}$ to $\{A', B'\}$, we cannot hope to compute accurate or even meaningful eigenvalues of singular problems, without further information. Typically, this information includes restrictions of allowable perturbations so that unperturbed and perturbed problems have similar structural characteristics. For this example the regularization requires that the perturbed pencil also have a 1×1 regular part and a 1×1 singular part.

A well-known class of singular pencils is the class of matrix pairs with intersecting null spaces. Let $x \neq 0$ belong to the intersection of the null spaces of A and B , i.e., $Ax = Bx = 0$. Then for any (α, β) , we have $(\beta A - \alpha B)x = 0$, implying that the pair $\{A, B\}$ is singular. By inspection we see that A_1 and B_1 above have a common one-dimensional column (and row) null space spanned by $x = e_2$. The dimensions of the intersecting column and row null spaces, respectively, are the number of L_0 and L_0^T blocks, respectively, in the pencils' KCF. Notice that the intersection of null spaces of A and B is a sufficient but not a necessary condition for a pencil to be singular, as illustrated with $A_2 - \lambda B_2$ in the first set of examples.

It is possible for a pair $\{A, B\}$ in Schur form to be very close to singular, and so to have very sensitive eigenvalues, even if no diagonal entries of A or B are small. It suffices for A and B to nearly have a common null space. For example, consider the 16×16 matrices

$$A'' = \begin{bmatrix} 0.1 & 1 & & & & \\ & 0.1 & 1 & & & \\ & & \ddots & \ddots & & \\ & & & 0.1 & 1 & \\ & & & & 0.1 & \end{bmatrix} \quad \text{and} \quad B'' = A''.$$

Then $\{A'', B''\}$ has all eigenvalues at 1. Changing A''_{n1} and B''_{n1} to 10^{-16} makes both A'' and B'' singular to machine precision, with a common null vector; i.e., there exists a unit vector x such that $\|A''x\|_2 = \|B''x\|_2 \approx O(10^{-16})$. Then, using a technique analogous to the one applied to the 2×2 example above, we can show that there is a perturbation of A'' and B'' of norm $10^{-16} + \epsilon$, for any $\epsilon > 0$, that makes the 16 perturbed eigenvalues have any prescribed complex values.

With these examples in mind we are ready to introduce the GUPTRI form and the regularization method used to compute meaningful and reliable information.

8.7.5 Generalized Schur-Staircase Form

In general, we cannot guarantee stable computation of the KCF of a pencil, since the transformation matrices that reduce $A - \lambda B$ to KCF can be arbitrarily ill-conditioned. However, it is possible to compute the Kronecker structure (or parts of it) using only unitary transformations. The price we have to pay is a denser canonical form, called a *generalized Schur-staircase form*. This form is block upper triangular with diagonal blocks in staircase form (also block upper triangular) that reveal the fine structure elements of the KCF.

In most applications it is enough to reduce $A - \lambda B$ to a generalized Schur-staircase form, e.g., to GUPTRI form [121, 122]:

$$P^*(A - \lambda B)Q = \begin{bmatrix} A_r - \lambda B_r & * & * \\ 0 & A_{reg} - \lambda B_{reg} & * \\ 0 & 0 & A_l - \lambda B_l \end{bmatrix}, \quad (8.34)$$

where P ($m \times m$) and Q ($n \times n$) are unitary. Here the square upper triangular block $A_{reg} - \lambda B_{reg}$ is regular and has the same regular structure as $A - \lambda B$ (i.e., contains all finite and infinite eigenvalues of $A - \lambda B$). The rectangular block $A_r - \lambda B_r$ has only right minimal indices in its KCF—indeed the same L_j blocks as $A - \lambda B$. Similarly, $A_l - \lambda B_l$ has only left minimal indices in its KCF, the same L_j^T blocks as $A - \lambda B$. If $A - \lambda B$ is singular, at least one of $A_r - \lambda B_r$ and $A_l - \lambda B_l$ will be present in (8.34). If $A - \lambda B$ is regular, $A_r - \lambda B_r$ and $A_l - \lambda B_l$ are not present in (8.34) and the GUPTRI form reduces to $A_{reg} - \lambda B_{reg}$. Staircase forms that reveal the Jordan structure of the zero and infinite eigenvalues are contained in $A_{reg} - \lambda B_{reg}$:

$$A_{reg} = \begin{bmatrix} A_z & * & * \\ 0 & A_f & * \\ 0 & 0 & A_i \end{bmatrix}, \quad B_{reg} = \begin{bmatrix} B_z & * & * \\ 0 & B_f & * \\ 0 & 0 & B_i \end{bmatrix}. \quad (8.35)$$

In summary, the diagonal blocks of the GUPTRI form of $A - \lambda B$ describe the Kronecker structure as follows:

- $A_r - \lambda B_r$ has all right singular structure (the right minimal indices).
- $A_z - \lambda B_z$ has all Jordan structure for the zero eigenvalue.
- $A_f - \lambda B_f$ has all finite, nonzero eigenvalues.
- $A_i - \lambda B_i$ has all Jordan structure for the infinite eigenvalue.
- $A_l - \lambda B_l$ has all left singular structure (the left minimal indices).

The explicit structure of the diagonal blocks in staircase form is presented in the next section. The nonzero, finite eigenvalues of $A - \lambda B$ (if any) are in the block $A_f - \lambda B_f$ but their multiplicities or Jordan structures are not computed explicitly. However, it is possible to extract the Jordan structure of a finite, nonzero eigenvalue of $A - \lambda B$ by computing the *RZ* (right-zero)-staircase form (see §8.7.6) of the shifted pencil $(A_f - \lambda_i B_f) - \lambda B_f$, which has zero as an eigenvalue of multiplicity ≥ 1 .

Given $A - \lambda B$ in GUPTRI form we also know different pairs of *reducing subspaces* [451, 121]. Suppose the eigenvalues on the diagonal of $A_{reg} - \lambda B_{reg}$ are ordered so that the first k , say, are in Λ_1 (a subset of the spectrum) and the remainder are outside Λ_1 . Then the GUPTRI form can also be expressed as

$$P^*(A - \lambda B)Q = \begin{bmatrix} A_{11} - \lambda B_{11} & A_{12} - \lambda B_{12} \\ 0 & A_{22} - \lambda B_{22} \end{bmatrix}, \quad (8.36)$$

where $A_{11} - \lambda B_{11}$ contains $A_r - \lambda B_r$ and the regular part corresponding to Λ_1 , and $A_{22} - \lambda B_{22}$ contains the remaining regular part and $A_l - \lambda B_l$. If $A_r - \lambda B_r$ is $m_r \times n_r$, then the left and right reducing subspaces corresponding to Λ_1 are spanned by the leading $m_r + k$ columns of P (denoted P_1) and the leading $n_r + k$ columns of Q (denoted Q_1), respectively, such that

$$(A - \lambda B)Q_1 = P_1(A_{11} - \lambda B_{11}).$$

When Λ_1 is empty, the corresponding reducing subspaces are called *minimal*, and when Λ_1 contains the whole spectrum the reducing subspaces are called *maximal*. The computation of minimal and maximal reducing subspaces appears in several applications, e.g., computing controllable and observable subspaces of generalized linear systems, which represent ill-posed problems in control theory [447, 120].

8.7.6 GUPTRI Algorithm

The algorithm for computing the GUPTRI form is based on two reductions (*staircase forms*) [121]. The first is the *RZ*-staircase form that reveals the right singular structure and the Jordan structure of the zero eigenvalue of $A - \lambda B$.

The *RZ*-algorithm uses a finite number of unitary equivalence transformations, where in step k ($= 1, 2, \dots$), n_k = dimension of the column null space of $A^{(k)}$ and $n_k - r_k$ = dimension of the intersecting column null space of $A^{(k)}$ and $B^{(k)}$ are determined. Here, $A^{(1)} = A$ and $B^{(1)} = B$, and $\{A^{(k)}, B^{(k)}\}$ for $k > 1$ correspond to the deflated matrix pair obtained after the equivalence transformation in step $k - 1$. The structure indices (*RZ*-indices) display the Kronecker structure as follows:

$$\begin{aligned} n_k - r_k &= \text{number of } L_{k-1} \text{ blocks,} \\ r_k - n_{k+1} &= \text{number of } J_k(0) \text{ blocks.} \end{aligned}$$

Before we state the general case we consider a 6×8 matrix pencil in *RZ*-staircase form:

$$\begin{aligned} A_{rz} - \lambda B_{rz} &= \left[\begin{array}{ccc} -\lambda B_{11} & A_{12} - \lambda B_{12} & A_{13} - \lambda B_{13} \\ 0 & -\lambda B_{22} & A_{23} - \lambda B_{23} \\ 0 & 0 & -\lambda B_{33} \end{array} \right] \\ &= \left[\begin{array}{cccc|cc|cc} 0 & 0 & 0 & 0 & \mathbf{x} & \mathbf{x} & \mathbf{x} & \mathbf{x} \\ 0 & 0 & 0 & 0 & \mathbf{x} & \mathbf{x} & \mathbf{x} & \mathbf{x} \\ 0 & 0 & 0 & 0 & \mathbf{x} & \mathbf{x} & \mathbf{x} & \mathbf{x} \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{y} & \mathbf{y} \\ 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{y} & \mathbf{y} \\ 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{y} & \mathbf{y} \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array} \right] - \lambda \left[\begin{array}{cccc|cc|cc} 0 & \mathbf{x} \\ 0 & 0 & \mathbf{x} & \mathbf{x} & \mathbf{x} & \mathbf{x} & \mathbf{x} & \mathbf{x} \\ 0 & 0 & 0 & \mathbf{x} & \mathbf{x} & \mathbf{x} & \mathbf{x} & \mathbf{x} \\ \hline 0 & 0 & 0 & 0 & \mathbf{y} & \mathbf{y} & \mathbf{y} & \mathbf{y} \\ 0 & 0 & 0 & 0 & 0 & \mathbf{y} & \mathbf{y} & \mathbf{y} \\ 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{y} & \mathbf{y} \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{z} \end{array} \right]. \end{aligned}$$

Nonzero entries after each deflation are marked with a unique letter (x from first step, y from second step, etc.), and they appear in bold or italic. The superdiagonal blocks of A_{rz} have full column rank and the diagonal blocks of B_{rz} have full row rank. The nonzero entries of the full rank blocks are marked in bold font. We use this notation in later examples as well.

The diagonal blocks (defining the “stairs”) in $A_{rz} - \lambda B_{rz}$ are of size $r_i \times n_i$ and show the following information:

$$\begin{aligned} n_1 &= 4 = \dim \mathcal{N}(A^{(1)}), & r_1 &= 3 = n_1 - \dim \mathcal{N}(A^{(1)} \cap B^{(1)}), \\ n_2 &= 2 = \dim \mathcal{N}(A^{(2)}), & r_2 &= 2 = n_2 - \dim \mathcal{N}(A^{(2)} \cap B^{(2)}), \\ n_3 &= 2 = \dim \mathcal{N}(A^{(3)}), & r_3 &= 1 = n_3 - \dim \mathcal{N}(A^{(3)} \cap B^{(3)}), \\ n_4 &= 0 = \dim \mathcal{N}(A^{(4)}). \end{aligned}$$

Now, we can conclude that the KCF of $A_{rz} - \lambda B_{rz}$ is $\text{diag}(L_0, L_2, J_1(0), J_3(0))$.

In general, let A and B be complex $m \times n$ matrices. Then it can be shown that there exist unitary matrices $U \in \mathcal{C}^{m \times m}$ and $V \in \mathcal{C}^{n \times n}$ such that the matrix pencil $A - \lambda B$ is transformed to the following so-called *RZ-staircase form* [121, 122]:

$$U^*(A - \lambda B)V = \begin{bmatrix} A_{rz} & A_{12} \\ 0 & A_{rest} \end{bmatrix} - \lambda \begin{bmatrix} B_{rz} & B_{12} \\ 0 & B_{rest} \end{bmatrix}, \quad (8.37)$$

where the staircase block structure of A_{rz} and B_{rz} reveals the right singular structure and the Jordan structure of the zero eigenvalue of $A - \lambda B$:

$$\begin{aligned} A_{rz} &= \begin{bmatrix} 0 & A_{12} & * & * & * & * \\ 0 & 0 & A_{23} & * & * & * \\ 0 & 0 & 0 & \ddots & * & * \\ \vdots & \vdots & \vdots & \ddots & A_{j-2,j-1} & * \\ 0 & \cdots & 0 & 0 & 0 & A_{j-1,j} \end{bmatrix}, \\ B_{rz} &= \begin{bmatrix} B_{11} & B_{12} & * & * & * & * \\ 0 & B_{22} & B_{23} & * & * & * \\ 0 & 0 & B_{33} & \ddots & * & * \\ \vdots & \vdots & \vdots & \ddots & B_{j-2,j-1} & * \\ 0 & \cdots & 0 & 0 & B_{j-1,j-1} & B_{j-1,j} \end{bmatrix}. \end{aligned} \quad (8.38)$$

The subblocks in (8.37) and (8.38) have the following properties:

- B_{ii} is $r_i \times n_i$ and $A_{i-1,i}, B_{i-1,i}$ are $r_{i-1} \times n_i$, where $n_i \geq r_i \geq n_{i+1} \geq r_{i+1}$.
- $B_{ii} = [0, R_{ii}]$, where R_{ii} is $r_i \times r_i$ and upper triangular.
- B_{ii} has full row rank r_i for $i = 1, \dots, j-1$.
- $A_{i-1,i}$ has full column rank n_i for $i = 2, \dots, j$.
- A_{rest} and B_{rest} are $r_{rest} \geq 0$ by $n_{rest} \geq 0$, where A_{rest} has full column rank if it appears.

Three cases can appear in the RZ-staircase form depending on r_{rest} and n_{rest} :

1. $r_{rest} > 0$ and $n_{rest} > 0$, in which case A_{rest} has full column rank.
2. $r_{rest} > 0$ and $n_{rest} = 0$.
3. $r_{rest} = n_{rest} = 0$.

In cases 1 and 2, the blocks appear as in (8.37) and the remaining Kronecker structure of $A - \lambda B$ is in $A_{rest} - \lambda B_{rest}$. In case 3, $A_{rest} - \lambda B_{rest}$ does not exist in (8.37). In all three cases, it is possible that the j th block column of $A_{rz} - \lambda B_{rz}$ (8.38) does not appear; if it does, $A_{j-1,j}$ also has full column rank. For convenience, let $r_j = 0$.

We see that the 6×8 example above corresponds to case 3 and that $A_{rz} - \lambda B_{rz}$ does not have a j th block column.

The second reduction is the *LI* (left-infinity)-staircase form that reveals the left singular structure and the Jordan structure of the infinite eigenvalue of $A - \lambda B$. This dual staircase form is obtained by working from the southeast corner of $B - \mu A$ and replacing column compressions by row compressions in the *RZ*-algorithm. The block indices n_i and r_i are dimensions of corresponding row null spaces and define the *LI*-indices. Moreover, $n_k - r_k$ and $r_k - n_{k+1}$ are the number of L_{k-1}^T and N_k blocks, respectively.

Both the *RZ*-staircase and *LI*-staircase reductions give us two types of structure elements which must be separated to obtain the GUPTRI form. For example, the right singular structure and the Jordan structure of the zero eigenvalue are separated by first applying the *RZ*-staircase reduction to $B_{rz} - \mu A_{rz}$ and insisting on the same right minimal indices. Then we obtain $\{A_r, B_r\}$ and are left with a pencil, say, $A_2 - \lambda B_2$ corresponding to the zero eigenvalue. Finally, $\{A_z, B_z\}$ is obtained by transforming $A_2 - \lambda B_2$ to *RZ*-staircase form.

We return to the 6×8 example and show the separated *R*-staircase and *Z*-staircase forms:

$$A_r - \lambda B_r = \left[\begin{array}{cc|cc|cc} 0 & 0 & \mathbf{x} & \mathbf{x} \\ 0 & 0 & 0 & \mathbf{y} \end{array} \right] - \lambda \left[\begin{array}{cc|cc|cc} 0 & \mathbf{x} & \mathbf{x} & \mathbf{x} \\ 0 & 0 & \mathbf{y} & \mathbf{z} \end{array} \right]$$

and

$$A_z - \lambda B_z = \left[\begin{array}{cc|cc|cc} 0 & 0 & \mathbf{x} & \mathbf{x} \\ 0 & 0 & \mathbf{x} & \mathbf{x} \\ 0 & 0 & 0 & \mathbf{y} \\ 0 & 0 & 0 & 0 \end{array} \right] - \lambda \left[\begin{array}{cc|cc|cc} \mathbf{x} & \mathbf{x} & \mathbf{x} & \mathbf{x} \\ 0 & \mathbf{x} & \mathbf{x} & \mathbf{x} \\ 0 & 0 & \mathbf{y} & \mathbf{y} \\ 0 & 0 & 0 & \mathbf{z} \end{array} \right].$$

As for $A_{rz} - \lambda B_{rz}$, the superdiagonal blocks of A_r and A_z have full column rank and the diagonal blocks of B_r and B_z have full row rank. In the following table, the structure indices for the *RZ*-, *R*-, and *Z*-staircase forms of the 6×8 example are summarized. We see that the *RZ*-staircase indices are the sum of the *R*- and *Z*-staircase indices, respectively.

i	1	2	3	4
n_i	4	2	2	0
r_i	3	2	1	0

i	1	2	3	4
n_i	2	1	1	0
r_i	1	1	0	0

i	1	2	3	4
n_i	2	1	1	0
r_i	2	1	1	0

In summary, the GUPTRI algorithm for computing (8.34) and (8.35) comprises seven reduction steps. The first three steps apply the *RZ*-staircase reduction to different blocks of $A - \lambda B$, giving the right singular structure ($A_r - \lambda B_r$) and the zero Jordan structure ($A_z - \lambda B_z$) in the top left corner of A and B . Similarly, the next three steps apply the *LI*-staircase reduction to different blocks of the remaining pencil, giving the infinite Jordan structure ($A_i - \lambda B_i$) and the left singular structure ($A_l - \lambda B_l$) in the bottom right corner of A and B . Then a square regular pencil is left in the middle of the transformed pencil, which corresponds to the finite and nonzero eigenvalues of $A - \lambda B$. By applying the QZ algorithm to this pencil, the upper triangular block $A_f - \lambda B_f$ is obtained.

A 24×21 Singular Pencil in GUPTRI Form. For completeness, we consider a pencil $A - \lambda B$ with all different types of structure blocks in its KCF:

$$\text{diag}(L_0, L_2, J_1(0), J_3(0), J_1(\alpha), J_1(\beta), J_1(\gamma), N_1, N_3, 2L_0^T, L_1^T, L_2^T, L_3^T).$$

Consequently, the GUPTRI form of this 24×21 pencil includes all types of diagonal blocks in (8.34) and (8.35). Since $A - \lambda B$ and the 6×8 example have the same right singular structure and Jordan structure for the zero eigenvalue, they also have the same $A_r - \lambda B_r$ and $A_z - \lambda B_z$ (see above). The GUPTRI form of the Jordan structure of the infinity eigenvalue ($A_i - \lambda B_i$) and the left singular structure ($A_l - \lambda B_l$) are as follows:

$$A_i - \lambda B_i = \left[\begin{array}{c|cc|cc} z & y & x & x \\ \hline 0 & y & x & x \\ 0 & 0 & x & x \\ 0 & 0 & 0 & x \end{array} \right] - \lambda \left[\begin{array}{c|cc|cc} 0 & y & x & x \\ \hline 0 & 0 & x & x \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array} \right],$$

$$A_l - \lambda B_l = \left[\begin{array}{c|ccc|ccc} z & y & y & x & x & x \\ \hline z & y & y & x & x & x \\ 0 & y & y & x & x & x \\ 0 & y & y & x & x & x \\ 0 & 0 & y & x & x & x \\ 0 & 0 & 0 & x & x & x \\ 0 & 0 & 0 & x & x & x \\ 0 & 0 & 0 & x & x & x \\ 0 & 0 & 0 & 0 & x & x \\ 0 & 0 & 0 & 0 & 0 & x \\ 0 & 0 & 0 & 0 & 0 & x \\ 0 & 0 & 0 & 0 & 0 & 0 \end{array} \right] - \lambda \left[\begin{array}{c|ccc|ccc} z & y & y & x & x & x \\ \hline 0 & y & y & x & x & x \\ 0 & y & y & x & x & x \\ 0 & 0 & 0 & x & x & x \\ 0 & 0 & 0 & x & x & x \\ 0 & 0 & 0 & x & x & x \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{array} \right].$$

Since these forms are computed using the *LI*-staircase reduction, the block indices n_i and r_i start to count from the southeast corner. Now, superdiagonal blocks of B_i and B_l have full row rank and diagonal blocks of A_i and A_l have full column rank. In the following table, the structure indices for the *LI*-, *L*-, and *I*-staircase forms are summarized.

i	1	2	3	4
n_i	7	4	3	1
r_i	5	3	2	0

i	1	2	3	4
n_i	5	3	2	1
r_i	3	2	1	0

i	1	2	3	4
n_i	2	1	1	0
r_i	2	1	1	0

Finally, the block in the GUPTRI form corresponding to the finite and nonzero eigenvalues α, β , and γ has the following appearance:

$$A_f - \lambda B_f = \left[\begin{array}{c|cc} x & x & x \\ \hline 0 & x & x \\ 0 & 0 & x \end{array} \right] - \lambda \left[\begin{array}{c|cc} x & x & x \\ \hline 0 & x & x \\ 0 & 0 & x \end{array} \right].$$

The diagonal elements in A_f and B_f are the eigenvalue pairs whose ratios are α, β , and γ (in any order).

So far the description for computing the GUPTRI form has relied on infinite precision arithmetic. In the presence of roundoff the problem is regularized by allowing

a deflation criterion for range/null space separations and thereby makes it possible to compute the GUPTRI form of a nearby matrix pencil.

This GUPTRI form is computed by a sequence of unitary equivalence transformations. The equivalence transformations are built from rank-revealing factorizations used to find orthonormal bases for different null spaces associated with the matrix pair.

Criterion for Determining the Numerical Rank. The GUPTRI algorithm makes use of the SVD for determining the numerical rank. Let $A = U\Sigma V^*$ be the SVD of an $n \times n A$, where U and V are unitary and $\Sigma = \text{diag}(\sigma_1, \dots, \sigma_n)$ with $0 \leq \sigma_1 \leq \dots \leq \sigma_n$.

Given a tolerance ϵ , the numerical rank of the matrix is the largest k such that $\sigma_{n-k+1} > \epsilon \cdot \sigma_n$. This definition does not ensure that the rank is stable in the sense that a small perturbation relative to ϵ could cause the matrix to change rank. The rank is sensitive to perturbations if σ_{n-k+1} is only slightly larger than σ_{n-k} . Given a constant gap , the requirement that k should be chosen as the largest number such that $\sigma_{n-k+1} > \epsilon \cdot \sigma_n$ and $\sigma_{n-k+1} > gap \cdot \sigma_{n-k}$ is added [121, 122]. In practice, $\sigma_{n-k+1} > \epsilon \|A\|_F$ is used; i.e., $\|A\|_2$ is replaced by $\|A\|_F$. If the gap criterion fails, σ_{n-k} is also treated as zero. This process is repeated until the gap criterion is satisfied. If this is not possible, the numerical rank is not defined for the tolerance parameters ϵ and gap .

The GUPTRI Form of a Regularized Problem. The deflations made in the staircase algorithm result in an exact GUPTRI form of a nearby matrix pencil $A + \delta A - \lambda(B + \delta B)$:

$$P^*(A + \delta A - \lambda(B + \delta B))Q = \begin{bmatrix} A_r - \lambda B_r & * & * \\ 0 & A_{reg} - \lambda B_{reg} & * \\ 0 & 0 & A_l - \lambda B_l \end{bmatrix}.$$

If all range/null space separations in the GUPTRI algorithm are well defined with respect to the deflation tolerance parameters ϵ and gap , then $\|(\delta A, \delta B)\|_F = O(\epsilon \|(A, B)\|_F)$. The nearby problem represents a regularized problem that has a stable Kronecker structure with respect to the deflation criteria of the algorithm.

8.7.7 Software Availability

The software package (in FORTRAN 77) for computing a GUPTRI form of an arbitrary $A - \lambda B$ and error bounds for computed generalized eigenvalues and reducing subspaces was published in [121, 122]. The software is developed for dense matrices stored as double precision complex data. An interface to MATLAB in terms of a MEX-file is also available. See the book's homepage, ETHOME, for access to this software.

8.7.8 More on GUPTRI and Numerical Examples

A typical user will call four of the routines, namely GUPTRI, REORDR, BOUND, and EVALBD. In the following we give a brief description of the functionality of these routines. We believe that the best way to get acquainted with singular or near to singular problems and the GUPTRI form is to start using the MATLAB interface of the GUPTRI routine (see p. 273).

GUPTRI: Reduces $A - \lambda B$ to a GUPTRI form (8.34)–(8.35) in terms of a unitary equivalence transformation. The finite nonzero eigenvalues (diagonal elements) of $A_{reg} - \lambda B_{reg}$ may appear in any order. Besides $\{A, B\}$, the user must provide three input parameters (EPSU, GAP, and ZERO) that control the computation of the GUPTRI form. These three parameters are also arguments for the MATLAB function (see p. 273).

REORDR: It is possible directly from the computed GUPTRI decomposition to produce a pair of reducing subspaces associated with any part of the spectrum $\lambda(A, B)$. The user must then call the routine REORDR and provide an integer function FTEST(ALPHA, BETA), describing the spectrum of the reducing subspace to be computed, as input parameter (see section 6 in [122] for an example).

Given that the specified regular part of $A - \lambda B$ is in upper triangular form, the 1×1 diagonal blocks (the generalized eigenvalues) are reordered using pairs of Givens rotations which are accumulated in previous transformation matrices P and Q . After the reordering, the eigenvalues specified by the function FTEST appear in $A_{11} - \lambda B_{11}$ (8.36), i.e., at the top northwest corner of the specified regular part of $A - \lambda B$ and the corresponding pair of reducing subspaces can easily be read off from the leading columns of P and Q .

BOUND and EVALBD: Error bounds for reducing subspaces and generalized eigenvalues are presented in [119, 122]. We have illustrated by examples that eigenvalues and reducing subspaces are ill posed, which means that extra conditions on allowable perturbations δA and δB of A and B must be imposed. BOUND and EVALBD compute error bounds for pairs of reducing subspaces, as well as for selected eigenvalues of the regular part. For the reducing subspace error bounds, it is required that $(A + \delta A) - \lambda(B + \delta B)$ has reducing subspaces of the same dimension as $A - \lambda B$. For the eigenvalue bounds, it is additionally required that $(A + \delta A) - \lambda(B + \delta B)$ has the same number of eigenvalues in its selected regular part. Both of these conditions are automatically verified by the software.

Based on input parameters, BOUND decides which perturbation theorem is applicable and computes the required quantities in error bounds for eigenvalues and reducing subspaces. EVALBD takes the output of BOUND and evaluates the reducing subspace bounds of the appropriate theorem for a given input $\text{DELTA} = \|(\delta A, \delta B)\|_E$.

Robustness of Computed GUPTRI Form and Error Bounds. The GUPTRI algorithm and software is numerically stable in the sense that it computes the exact Kronecker structure (generalized Schur-staircase form) of a nearby pencil $A' - \lambda B'$. $\delta \equiv \|(A - A', B - B')\|_E$ is an upper bound on the distance to the closest $\{A + \delta A, B + \delta B\}$ with the KCF of $\{A', B'\}$. An accurate estimate of δ is the square root of the sum of the squares of all singular values interpreted as zeros during the reduction to GUPTRI form.

We refer to sections 5 and 6 in [122] for a more detailed discussion of the computed GUPTRI form and the associated error bounds for reducing subspaces and generalized eigenvalues. For example, section 6 presents a sample usage of all four routines for an application in control theory (computing the controllable subspace and uncontrollable modes).

Arithmetic and Space Complexity. It is quite a costly procedure to compute a GUPTRI form and associated error bounds. In the worst case (e.g., a generic $(n+1) \times n$ pencil), GUPTRI requires $O(n^4)$ floating point operations. The storage requirements of each routine are discussed in [122] (sections 3 and 7). The real bottleneck in the present software is the workspace required by the routine BOUND and the computations performed therein. There is a routine (BNDWSP) that computes the exact amount of workspace needed in the routine BOUND.

MATLAB Interface to GUPTRI. An interface to MATLAB in terms of a MEX-file is available for the routine GUPTRI. In the following we give a brief description of the arguments and output of the guptri function and demonstrate its usage on a small numerical example.

The most general call,

$$[S, T, P, Q, kstr] = \text{guptri}(A, B, EPSU, GAP, ZERO),$$

stores the transformation matrices in P and Q and P^*AQ and P^*BQ in S and T , respectively. The computed Kronecker structure is revealed by $kstr$, as described in the example discussed later in this section.

The guptri function reduces an $m \times n$ pencil $A - \lambda B$ to generalized upper triangular form $S - \lambda T = P^*(A - \lambda B)Q$ as in (8.34)–(8.35):

$$S = \begin{bmatrix} A_r & * & * & * & * \\ 0 & A_z & * & * & * \\ 0 & 0 & A_f & * & * \\ 0 & 0 & 0 & A_i & * \\ 0 & 0 & 0 & 0 & A_l \end{bmatrix}, \quad T = \begin{bmatrix} B_r & * & * & * & * \\ 0 & B_z & * & * & * \\ 0 & 0 & B_f & * & * \\ 0 & 0 & 0 & B_i & * \\ 0 & 0 & 0 & 0 & B_l \end{bmatrix}, \quad (8.39)$$

where the diagonal blocks of S and T in staircase forms describe the Kronecker structure of $A - \lambda B$ (see §8.7.6).

Besides $\{A, B\}$, the user can optionally provide three input parameters (EPSU, GAP, and ZERO) that control the computation of the GUPTRI form. EPSU (relative uncertainty in data) and GAP (should be at least 1 and nominally 1000) are used to make rank decisions in order to determine the Kronecker structure of an input pencil (see p. 271). The default value is $GAP = 1000$, but other values may be necessary for certain examples. ZERO is used as a logical variable, which when set to true (ZERO = nonzero value) forces guptri to zero out small singular values during the reduction process so that the returned pencil really has the computed GUPTRI form (see p. 271). Otherwise, the returned pencil is a true equivalence transformation of the input pencil and all zero blocks in the GUPTRI form will contain small entries (normally of size EPSU at most).

Example in MATLAB. Let us consider the following 5×5 singular pencil $A - \lambda B$ with integer entries:

$$A = \begin{bmatrix} 22 & 34 & 31 & 31 & 17 \\ 45 & 45 & 42 & 19 & 29 \\ 39 & 47 & 49 & 26 & 34 \\ 27 & 31 & 26 & 21 & 15 \\ 38 & 44 & 44 & 24 & 30 \end{bmatrix}, \quad B = \begin{bmatrix} 13 & 26 & 25 & 17 & 24 \\ 31 & 46 & 40 & 26 & 37 \\ 26 & 40 & 19 & 25 & 25 \\ 16 & 25 & 27 & 14 & 23 \\ 24 & 35 & 18 & 21 & 22 \end{bmatrix}.$$

By construction the KCF of this example is $\text{diag}(L_0, J_2(0), J_1(2), N_1, L_0^T)$. We compute the GUPTRI form of $A - \lambda B$ by the call

```
[S, T, P, Q, kstr] = guptri(A, B);
```

i.e., we use the default values on EPSU, GAP, and ZERO. The output from this call is displayed below:

S =

0	0	-31.2179	69.7186	-142.6727
0	0	0	15.8532	-42.1039
0	0	0	-15.5639	-3.4712
0	0	0	0	13.5979
0	0	0	0	0

T =

0	-21.5942	-22.1660	44.5153	-110.9818
0	0	-25.0581	19.3190	-43.8905
0	0	0	-7.7820	9.2041
0	0	0	0	0
0	0	0	0	0

P =

0.2697	0.4520	0.8072	0.0226	-0.2660
0.4045	0.5101	-0.4985	0.5400	-0.1900
0.6742	-0.3818	0.1705	0.2136	0.5701
0.1348	0.5620	-0.1771	-0.6188	0.5017
0.5394	-0.2718	-0.1986	-0.5285	-0.5625

Q =

-0.4044	0.1510	0.4636	0.7253	-0.2697
0.6985	-0.4097	0.1200	0.1975	-0.5394
0.1470	0.6896	-0.5633	0.1481	-0.4045
-0.4044	-0.0310	0.1859	-0.5886	-0.6742
-0.4044	-0.5769	-0.6471	0.2582	-0.1348

kstr =

2	1	0	-1	2	0	-1	1	-1
1	1	0	-1	1	0	-1	1	-1

The first part of kstr (all the columns to the left of the first -1) shall be interpreted as follows:

- The number of L_{k-1} blocks is $n_k - r_k \equiv \text{kstr}(1, k+1) - \text{kstr}(2, k+1)$.
- The number of $J_k(0)$ blocks is $r_k - n_{k+1} \equiv \text{kstr}(2, k) - \text{kstr}(1, k+1)$.

The second part of kstr contains the corresponding information about L_i^T and N_i blocks and the third contains information about the size of the regular part. For this

example we see that the computed GUPTRI form has the expected Kronecker structure, including one L_0 block ($\text{kstr}(1, 1) - \text{kstr}(2, 1) = 1$), one $J_2(0)$ block ($\text{kstr}(2, 2) - \text{kstr}(1, 3) = 1$), one L_0^T block ($\text{kstr}(1, 5) - \text{kstr}(2, 5) = 1$), one N_1 block ($\text{kstr}(2, 5) - \text{kstr}(1, 6) = 1$), and finally a 1×1 regular part corresponding to the finite nonzero eigenvalue 2 ($\text{kstr}(1, 8) = \text{kstr}(2, 8) = 1$). The L_0 block corresponds to A and B having a common column null space (both S and T have a first column with zero entries). Similarly, the L_0^T block corresponds to A and B having a common row null space (both S and T have a last row with zero entries). Since L_0 and L_0^T are the only singular blocks, $A_r - \lambda B_r$ is of size 0×1 and $A_l - \lambda B_l$ is of size 1×0 . The remaining Kronecker structure in the GUPTRI form is contained in $S(1:4, 2:5)$ and $T(1:4, 2:5)$, i.e., rows 1 to 4 and columns 2 to 5 of S and T : $A_z - \lambda B_z = S(1:2, 2:3) - \lambda T(1:2, 2:3)$, $A_f - \lambda B_f = S(3, 4) - \lambda T(3, 4)$, and finally $A_i - \lambda B_i = S(4, 5) - \lambda T(4, 5)$.

The computed transformation matrices P and Q are orthogonal to machine precision accuracy. Moreover,

$$\delta = \| (A - A', B - B') \|_F = 2.8184e-13$$

is an upper bound on the distance to the closest $\{A + \delta A, B + \delta B\}$ with the KCF of $\{A', B'\}$, where $A' = PSQ^*$ and $B' = PTQ^*$. For this example, δ is of size $O(\epsilon_M \| (A, B) \|_F)$, where ϵ_M is the machine precision.

By setting `ZERO = 0` and making the call (with the default values on `EPSU` and `GAP`)

$$[S, T, P, Q, \text{kstr}] = \text{guptri}(A, B, \text{EPSU}, \text{GAP}, \text{ZERO}),$$

we perform a true equivalence transformation of $A - \lambda B$:

`S =`

-2.9227e-15	0	3.1218e+01	6.9719e+01	-1.4267e+02
2.0749e-15	-3.0175e-15	1.7377e-14	1.5853e+01	-4.2104e+01
-5.4031e-15	-1.7747e-15	8.8818e-16	-1.5564e+01	-3.4712e+00
1.6694e-15	-4.4157e-15	2.2204e-15	0	1.3598e+01
-8.3206e-16	3.1071e-16	1.9984e-15	-2.2204e-16	0

`T =`

-6.9809e-31	2.1594e+01	2.2166e+01	4.4515e+01	-1.1098e+02
3.4328e-31	0	2.5058e+01	1.9319e+01	-4.3890e+01
1.7764e-15	0	0	-7.7820e+00	9.2041e+00
-5.5511e-16	0	0	0	-7.5221e-15
0	0	0	0	3.1465e-15

The tiny nonzero elements in the lower triangular parts of S and T correspond to the singular values that are interpreted as numerically zero in the rank determination process of the GUPTRI algorithm. The results with the default value on `ZERO` correspond to the regularized pencil that has the computed S and T as its exact GUPTRI form with Kronecker structure as reported in `kstr`.

We end the discussion by exposing our seemingly harmless 5×5 example to the MATLAB function `eig`. The call `[V,D] = eig(A,B)` is supposed to compute a diag-

onal eigenvalue matrix D and a full matrix V whose columns are the corresponding eigenvectors so that $AV = BVD$. MATLAB computes a D with the diagonal entries

```
-1.8351e+16
2.0000e+00
7.2695e-01 - 4.1359e-25i
-6.2535e-16 + 2.4399e-08i
-5.9077e-16 - 2.4399e-08i
```

and an eigenvector matrix V with condition number $\|V\| \cdot \|V^{-1}\| = 8.8817e+08$. The large condition number and the residual $\|AV - BVD\| = 15.595$ signal that this is not a harmless example and further investigations are necessary. We conclude that it is only by software tools like guptri that we can get a more complete understanding of such ill-posed problems. And they do exist in real applications! Examples include controllability and observability issues in linear systems theory (see [447, 120]).

8.7.9 Notes and References

Theory and computation of spectral properties of singular matrix pencils have been intensively studied during the last two decades. Much motivation has come from systems and control theory, for example, computing the controllable subspace and uncontrollable modes (e.g., see [348, 447, 120] and references therein). The basic theory dates back to Kronecker in 1890 [276] and is well presented in Gantmacher [187]. The method described there for computing the KCF is numerically unstable. Several authors have more recently proposed algorithms that reduce a singular $A - \lambda B$ to generalized Schur-staircase form which are numerically stable. They compute the exact GUPTRI form (or something similar) of a pencil $A' - \lambda B'$ differing only slightly from the input pencil $A - \lambda B$.

Kublanovskaya introduced the first staircase algorithm for computing the Jordan structure of matrices in 1966 [278]. She used a normalized RQ factorization for rank determinations and null space separations. In 1970 Ruhe [371] introduced the use of the SVD in the staircase algorithm. The use of the SVD in an algorithm based on the chain relations was introduced by Golub and Wilkinson [199] and further analyzed and generalized in [250]. Kågström and Ruhe [253, 252] developed the first library-quality software for the complete reduction to Jordan normal form (JNF), with the capability of returning after different steps in the reduction. Recently, Chaitin-Chatelin and Frayssé [77] developed a nonstaircase qualitative approach. Kågström and Wiberg [254] considered the problem of computing partial canonical information for large-scale eigenvalue problems. They combined the implicitly restarted Arnoldi method (IRAM) [419] for computing a partial Schur form with staircase algorithms for computing the Jordan–Schur and the Weierstrass–Schur forms of matrices and regular matrix pencils, respectively.

Van Dooren [446, 451] gave a generalization of Kublanovskaya's staircase algorithm [278] to singular pencils using unitary equivalence transformations. In 1977 Kublanovskaya introduced the AB algorithm for computing the right singular structure and the Jordan structure of the zero eigenvalue for a singular pencil (e.g., see [279, 280]). Kågström [251] gave an RG-SVD/RG-QZD algorithm that provided a base for later work on software. Beelen and Van Dooren presented a faster but less reliable algorithm which requires $O(m^2n)$ operations for an $m \times n$ pencil. Error bounds for

computed quantities like reducing subspaces and eigenvalues were given by Demmel and Kågström [119] and were also implemented in their software [121, 122].

Recently, several papers were published that use geometry of matrix and matrix pencil spaces to understand Jordan and Kronecker structure problems and algorithms. Demmel and Edelman [116] used the staircase algorithm to calculate the dimension of matrices and pencils with a given canonical form. Elmroth and Kågström [160] made a comprehensive study of the set of 2×3 pencils, the smallest nontrivial rectangular case. Edelman and Ma presented a geometrical explanation of staircase algorithm failures [154]. Edelman, Elmroth, and Kågström [152, 153] study versatility and stratifications. They discuss and complete the mathematical theory for orbits and bundles of matrices (Arnold [18]) and pencils (Pokrzywa [367], De Hoyos [106]) and propose that knowledge of the closure relations may be applied in the staircase algorithm [153].

StratiGraph is a recent Java-based tool for computation and presentation of graphs displaying closure hierarchies of Jordan and Kronecker structures [248, 159]. Given the dimension of the problem and a canonical structure, the user can easily get information about nearby Jordan and Kronecker structures in the closure hierarchy. The stratification of orbits of pencils associated with problems in control theory has recently been studied by several authors, including Hinrichsen and O'Halloran [231], Boley [55], and Garcia-Planas [188]. For control applications, the pencils of interest typically have no row indices or no column indices, which can be seen as special cases of the general $A - \lambda B$ problem.

8.8 Stability and Accuracy Assessments

Z. Bai and R. Li

In this section, we discuss the tools to assess the accuracy of computed eigenvalues and corresponding eigenvectors of the GNHEP of a regular matrix pair $\{A, B\}$. We only assume the availability of residual vectors which are usually available upon the exit of a successful computation or cost marginal to compute afterwards. For the treatment of error estimation for the computed eigenvalues, eigenvectors, and deflating subspaces of dense GNHEPs, see Chapter 4 of the *LAPACK Users' Guide* [12].

The situation for general regular pairs $\{A, B\}$ is more complicated than the standard NHEP discussed in §7.13 (p. 228), especially when B is singular, in which case the *characteristic polynomial* $\det(A - \lambda B)$ no longer has degree n , the dimension of the matrices A and B . Even when B is mathematically nonsingular but nearly singular, problems arise when one tries to convert it to a standard eigenvalue problem for $B^{-1}A$, which then could have huge eigenvalues and consequently cause numerical instability. To account for all possibilities, a homogeneous representation of an eigenvalue λ by a nonzero pair of numbers (α, β) has been proposed:

$$\lambda \equiv \alpha/\beta, \quad |\alpha|^2 + |\beta|^2 > 0.$$

When $\beta = 0$, such pairs represent *infinite* eigenvalues, and this occurs when B is singular. Such representations are clearly not unique since $(\xi\alpha, \xi\beta)$ represents the same ratio for any $\xi \neq 0$, and consequently the same eigenvalue. So really a pair (α, β) is a representative from a class of pairs that give the same ratio.

With this new representation of an eigenvalue, the characteristic polynomial takes the form $\det(\beta A - \alpha B)$, which does have total degree of n in α and β . (In fact the i th term in its expansion is a multiple of $\alpha^i \beta^{n-i}$.)

But how do we measure the difference of two eigenvalues, given the fact of non-uniqueness in their representations? We resort to the chordal metric for $\lambda = \alpha/\beta$ and $\tilde{\lambda} = \tilde{\alpha}/\tilde{\beta}$; their distance in chordal metric is defined as

$$\chi(\lambda, \tilde{\lambda}) = \chi((\alpha, \beta), (\tilde{\alpha}, \tilde{\beta})) = \frac{|\alpha\tilde{\beta} - \beta\tilde{\alpha}|}{\sqrt{|\alpha|^2 + |\beta|^2} \sqrt{|\tilde{\alpha}|^2 + |\tilde{\beta}|^2}} = \frac{|\lambda - \tilde{\lambda}|}{\sqrt{1 + |\lambda|^2} \sqrt{1 + |\tilde{\lambda}|^2}}.$$

We are now ready to address the issue of assessing the accuracy of computed approximations.

Residual Vectors. Let $(\tilde{\alpha}, \tilde{\beta})$ denote a computed eigenvalue, and let \tilde{x} be its corresponding computed eigenvector, i.e.,

$$\tilde{\beta}A\tilde{x} \approx \tilde{\alpha}B\tilde{x}.$$

Sometimes, its corresponding left computed eigenvector \tilde{y} is also available:

$$\tilde{\beta}\tilde{y}^*A \approx \tilde{\alpha}\tilde{y}^*B.$$

The residual vectors corresponding to the computed eigenvalue $(\tilde{\alpha}, \tilde{\beta})$ are defined as

$$r = \tilde{\beta}A\tilde{x} - \tilde{\alpha}B\tilde{x} \quad \text{and} \quad s^* = \tilde{\beta}\tilde{y}^*A - \tilde{\alpha}\tilde{y}^*B,$$

respectively. For the simplicity, we normalize the approximate eigenvectors so that $\|\tilde{x}\|_2 = 1$ and $\|\tilde{y}\|_2 = 1$ and normalize the approximate eigenvalue so that $|\tilde{\alpha}|^2 + |\tilde{\beta}|^2 = 1$.

Transfer Residual Errors to Backward Errors. It can be shown that the computed eigenvalue and eigenvector(s) are the exact ones of a nearby matrix pair, i.e.,

$$\tilde{\beta}(A + E)\tilde{x} = \tilde{\alpha}(B + F)\tilde{x},$$

and

$$\tilde{\beta}\tilde{y}^*(A + E) = \tilde{\alpha}\tilde{y}^*(B + F)$$

if \tilde{y} is available, where error matrices E and F are small relative to the norms of A and B .

1. Only \tilde{x} is available but \tilde{y} is not. Then the optimal error matrix (E, F) (in both the 2-norm and the Frobenius norm) for which $(\tilde{\alpha}, \tilde{\beta})$ and \tilde{x} are an exact eigenvalue and its corresponding eigenvector of the pair $(A + E, B + F)$ satisfies

$$\|(E, F)\|_2 = \|(E, F)\|_F = \|r\|_2. \tag{8.40}$$

2. Both \tilde{x} and \tilde{y} are available. Then the optimal error matrices (E_2, F_2) (in the 2-norm) and (E_F, F_F) (in the Frobenius norm) for which $(\tilde{\alpha}, \tilde{\beta})$, \tilde{x} , and \tilde{y} are an

exact eigenvalue and its corresponding eigenvectors of the pair $\{A + E, B + F\}$ satisfy

$$\|(E_2, F_2)\|_2 = \max\{\|r\|_2, \|s^*\|_2\} \quad (8.41)$$

and

$$\|(E_F, F_F)\|_F = \sqrt{\|r\|_2^2 + \|s^*\|_2^2 - (\tilde{\beta}\tilde{y}^* A \tilde{x} - \tilde{\alpha}\tilde{y}^* B \tilde{x})^2}. \quad (8.42)$$

See [256, 431, 473].

We say the algorithm that delivers the approximate eigenpair $((\tilde{\alpha}, \tilde{\beta}), \tilde{x})$ is τ -backward stable for the pair with respect to the norm $\|\cdot\|$ if it is an exact eigenpair for $\{A + E, B + F\}$ with $\|(E, F)\| \leq \tau$; analogously the algorithm that delivers the eigentriplet $((\tilde{\alpha}, \tilde{\beta}), \tilde{x}, \tilde{y})$ is τ -backward stable for the triplet with respect to the norm $\|\cdot\|$ if it is an exact eigentriplet for $\{A + E, B + F\}$ with $\|(E, F)\| \leq \tau$. With these in mind, statements can be made about the backward stability of the algorithm which computes the eigenpair $((\tilde{\alpha}, \tilde{\beta}), \tilde{x})$ or the eigentriplet $((\tilde{\alpha}, \tilde{\beta}), \tilde{x}, \tilde{y})$. Conventionally, an algorithm is called backward stable if $\tau = O(\epsilon_M \|(A, B)\|)$.

Error Bound for Computed Eigenvalues. It can be proved [425] that up to the first order of $\|(E, F)\|_2$, there is an eigenvalue (α, β) of A and B satisfying

$$\chi((\alpha, \beta), (\tilde{\alpha}, \tilde{\beta})) \lesssim c_{(\alpha, \beta)} \cdot \|(E_2, F_2)\|_2, \quad (8.43)$$

where

$$c_{(\alpha, \beta)} \equiv \frac{\|x\|_2 \|y\|_2}{\sqrt{|y^* Ax|^2 + |y^* Bx|^2}}.$$

It is called the *individual condition number* of (α, β) , which in actual computations will be estimated with the exact eigenvectors x and y replaced by their approximations \tilde{x} and \tilde{y} .

Error Bound for Computed Eigenvectors. LAPACK [12] presents a few error bounds for computed eigenvectors of the (small) dense GNHEP of a regular matrix pair $\{A, B\}$, assuming the availability of the generalized Schur decomposition of the pair. Interested readers are referred to [425, 431]. It is generally not possible to assess the accuracy of computed eigenvectors of the large sparse problems with only partial information of a few eigenvalues and eigenvectors.

This page intentionally left blank

Chapter 9

Nonlinear Eigenvalue Problems

9.1 Introduction

In theory and practical applications one may encounter eigenproblems that are more complicated than the standard and generalized eigenproblems discussed in previous chapters. In §9.2 we pay attention to the important class of quadratic eigenproblems, with a small sidestep to higher order polynomial eigenproblems in §9.3.

In §9.4 the scope of the algebraic eigenvalue problem is widened to geometrical properties of invariant subspaces. A template is presented that can be used to solve variational problems that are defined over sets of subspaces. More specifically, algorithms and software are provided for the optimization of a real valued function $F(Y)$ over Y such that $Y^*Y = I$. It is shown, by example, how many eigenvalue-related problems, such as the Procrustes problem, determination of nearest Jordan structure, and trace minimization, can be attacked with the given techniques.

9.2 Quadratic Eigenvalue Problems

Z. Bai, G. Sleijpen, and H. van der Vorst

9.2.1 Introduction

In this section, we consider the quadratic eigenvalue problem (QEP) of the form

$$(\lambda^2 M + \lambda C + K)x = 0 \quad \text{and} \quad y^*(\lambda^2 M + \lambda C + K) = 0, \quad (9.1)$$

where M , K , and C are given matrices of size $n \times n$. The nontrivial n -vectors x , y , and the corresponding scalars λ are the right, left eigenvectors, and eigenvalues, respectively.

The matrix function $L(\lambda) \equiv \lambda^2 M + \lambda C + K$ is a special case of a matrix polynomial, or a λ -matrix; see, for example, [187, 284, 194]. In this case, it is a λ -matrix of degree 2. The matrix function $L(\lambda)$ is said to be *regular* if $\det(L(\lambda))$ is not identical to zero for all λ . Otherwise, it is called *singular*.

An important special case of the quadratic eigenvalue problem is when

$$M^* = M > 0, \quad C^* = C, \quad \text{and} \quad K^* = K > 0. \quad (9.2)$$

These matrices are sometimes called mass, damping, and stiffness matrices, respectively, referring to their origin in mechanical engineering models; see, for instance, [145]. In some problems, the stiffness matrix K is only semi-positive definite. In this case, we may consider a shifted QEP to be discussed in §9.2.3.

One of the factors that makes the QEP different from standard eigenproblems $Ax = \lambda x$, or generalized eigenproblems $Ax = \lambda Bx$, is that there are $2n$ eigenvalues for QEP, with at most $2n$ right (and left) eigenvectors. Of course, in an n -dimensional space the right (and left) eigenvectors no longer form an independent set. This is illustrated by the following simple example. The triplet

$$M = \begin{bmatrix} 5 & 2 \\ 1 & 4 \end{bmatrix}, \quad C = \begin{bmatrix} 0 & 1 \\ 0 & 7 \end{bmatrix}, \quad K = \begin{bmatrix} 2 & 0 \\ 0 & 12 \end{bmatrix}$$

has 4 different (but pairwise conjugate) eigenvalues (rounded to five decimals):

$$\begin{aligned} \lambda_1 &= -0.9396 + 1.5749i, & \lambda_2 &= -0.9396 - 1.5749i, \\ \lambda_3 &= -0.0049 + 0.6296i, & \lambda_4 &= -0.0049 - 0.6296i. \end{aligned}$$

The associated eigenvectors (normalized so that the first coordinate is equal to 1) are:

$$\begin{aligned} x_1 &= (1, -2.4756 - 0.9779i)^T, & x_2 &= (1, -2.4756 + 0.9779i)^T, \\ x_3 &= (1, 0.0326 - 0.0132i)^T, & x_4 &= (1, 0.0326 + 0.0132i)^T. \end{aligned}$$

The four eigenvectors are obviously dependent, but, in actual problems, each of them may represent a relevant state of the system.

One has to be careful with Rayleigh quotients for quadratic eigenproblems. Indeed, given x as a right eigenvector for the QEP (9.1), i.e.,

$$(\lambda^2 M + \lambda C + K)x = 0,$$

one can form a *quadratic Rayleigh quotient*:

$$\lambda^2(x^* M x) + \lambda(x^* C x) + (x^* K x) = 0. \quad (9.3)$$

However, this equation has two roots; one of the roots is an eigenvalue, the other root may be a spurious one. For instance, if we compute the quadratic Rayleigh quotient for our example, with (λ_1, x_1) , then clearly, the pair (λ_1, x_1) satisfies equation (9.3). If we solve equation (9.3), then we find the two roots $\mu_1 = -0.9396 + 1.5749i$, $\mu_2 = -0.8776 - 1.6057i$. We see that λ_1 is recovered (by μ_1) and the other root has no meaning for the given QEP.

In an effort to decide which of the two is the desired one and which is the spurious one, one could compute the residual vector

$$r_\mu \equiv (\mu^2 M + \mu C + K)x_1,$$

and this leads to $\|r_{\mu_1}\|_2 \approx 8.4 \times 10^{-14}$, $\|r_{\mu_2}\|_2 \approx 12.5$, which, in this case, clearly points out that μ_2 is not an eigenvalue. We cannot exclude the possibility that in contrived

examples, one might make a wrong choice, which may lead to a delay in a specific iterative solution method.

For more general matrices, we can have defectiveness, as for the standard eigenproblems, which means that there is not necessarily a complete set of eigenvectors. In the next section, we will relate the QEP to a generalized standard problem, which helps to shed more light on this matter.

9.2.2 Transformation to Linear Form

It is easy to see that the QEP in (9.1) is equivalent to the following generalized “linear” eigenvalue problem:¹

$$Az = \lambda Bz \quad \text{and} \quad w^* A = \lambda w^* B, \quad (9.4)$$

where

$$A = \begin{bmatrix} 0 & I \\ -K & -C \end{bmatrix}, \quad B = \begin{bmatrix} I & 0 \\ 0 & M \end{bmatrix} \quad (9.5)$$

and

$$z = \begin{bmatrix} x \\ \lambda x \end{bmatrix}, \quad w = \begin{bmatrix} (\lambda M + C)^* y \\ y \end{bmatrix}. \quad (9.6)$$

The generalized eigenvalue problem (9.4) is commonly called a linearization of the QEP (9.1). It can be shown that for any matrices A and B of the above forms, the right and left eigenvectors z and w have the structures described in (9.6).

Note that from the factorization

$$A - \lambda B = \begin{bmatrix} 0 & I \\ -I & -\lambda M - C \end{bmatrix} \begin{bmatrix} \lambda^2 M + \lambda C + K & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} I & 0 \\ -\lambda I & I \end{bmatrix}, \quad (9.7)$$

we can conclude that the pencil $A - \lambda B$ is *equivalent*² to the matrix

$$\begin{bmatrix} \lambda^2 M + \lambda C + K & 0 \\ 0 & I \end{bmatrix} \quad (9.8)$$

and

$$\det(A - \lambda B) = \det(\lambda^2 M + \lambda C + K).$$

This means that the eigenvalues of the original QEP (9.1) coincide with the eigenvalues of the generalized eigenvalue problem (9.4). Furthermore, we have that

- $L(\lambda)$ is regular if and only if $A - \lambda B$ is regular;
- if M (hence B) is nonsingular, then $L(\lambda)$ is regular;
- if K (hence A) is nonsingular, then $L(\lambda)$ is regular.

For the theory on regular pencils (A, B) , see, for instance, [425, Chap. VI]. We will assume that at least M is nonsingular throughout this section.

¹The term “linear” is in quotation marks because λ appears in the eigenvectors as well.

²Two matrix polynomials $M_1(\lambda)$ and $M_2(\lambda)$ of size $n \times n$ are called equivalent if $M_1(\lambda) = E(\lambda)M_2(\lambda)F(\lambda)$ for some $n \times n$ matrix polynomials $E(\lambda)$ and $F(\lambda)$ with constant nonzero determinants (unimodular).

A disadvantage of the above reduction to linear form is that if the matrices M , C , and K are all Hermitian, then this is not reflected in the reduced form (9.5), where A is non-Hermitian. This can be repaired as follows.

In fact, the matrix pair (A, B) in (9.4) can be chosen in a more general form

$$A = \begin{bmatrix} 0 & W \\ -K & -C \end{bmatrix}, \quad B = \begin{bmatrix} W & 0 \\ 0 & M \end{bmatrix},$$

where W can be any arbitrary nonsingular matrix. Note that now the matrix pencil $A - \lambda B$ is equivalent to the matrix polynomial (9.8) if and only if W is nonsingular, and because of (9.7),

$$\det(A - \lambda B) = \det(W) \cdot \det(\lambda^2 M + \lambda C + K).$$

For example, if the matrices M , K , and C are all symmetric, as in the special case (9.2), and K is nonsingular, then we may choose $W = -K$, which leads to the following symmetric generalized “linear” eigenvalue problem

$$Az = \lambda Bz \quad \text{and} \quad w^* A = \lambda w^* B, \quad (9.9)$$

where

$$A = \begin{bmatrix} 0 & -K \\ -K & -C \end{bmatrix}, \quad B = \begin{bmatrix} -K & 0 \\ 0 & M \end{bmatrix}, \quad (9.10)$$

and

$$z = \begin{bmatrix} x \\ \lambda x \end{bmatrix}, \quad w = \begin{bmatrix} y \\ \bar{\lambda} y \end{bmatrix}. \quad (9.11)$$

Both A and B are symmetric, but may be indefinite.

9.2.3 Spectral Transformations for QEP

Invert QEP. For most iterative methods for solving a generalized eigenvalue problem, the formulation (9.4), with either (9.5) or with (9.10), is suitable if one wants to determine a few of the exterior eigenvalues and eigenvectors. If one wants to compute some of the smallest (in modulus) eigenvalues and eigenvectors, then the obvious transformation is $\mu = 1/\lambda$, and, after multiplying the QEP (9.1) with μ^2 , we obtain the *invert QEP*:

$$(M + \mu C + \mu^2 K) x = 0. \quad (9.12)$$

Here it is assumed that 0 is not an eigenvalue of the original QEP (9.1), i.e., that K is nonsingular.

The QEP for the triplet $\{K, C, M\}$ can be linearized as discussed in §9.2.2, for instance, as (9.4) with (9.5), where M interchanged with K . We can reformulate this generalized linearized eigenproblem in terms of λ , instead of μ , which leads to

$$Az = \frac{1}{\lambda} Bz, \quad (9.13)$$

where

$$A = \begin{bmatrix} -C & -M \\ I & 0 \end{bmatrix}, \quad B = \begin{bmatrix} K & 0 \\ 0 & I \end{bmatrix}, \quad z = \begin{bmatrix} x \\ \lambda x \end{bmatrix}. \quad (9.14)$$

Note that from the factorization

$$B - \lambda A = \begin{bmatrix} I & \lambda M \\ 0 & I \end{bmatrix} \begin{bmatrix} \lambda^2 M + \lambda C + K & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} I & 0 \\ -\lambda I & I \end{bmatrix}$$

we know that the pencil $B - \lambda A$ is equivalent to

$$\begin{bmatrix} \lambda^2 M + \lambda C + K & 0 \\ 0 & I \end{bmatrix}.$$

Since $\det(B - \lambda A) = \det(\lambda^2 M + \lambda C + K)$, we conclude that the matrix pencil $B - \lambda A$ is regular if and only if the quadratic matrix polynomial $\lambda^2 M + \lambda C + K$ is regular and the eigenvalues of the original QEP (9.1) coincide with the eigenvalues of the matrix pencil $B - \lambda A$.

For the special case (9.2), we may formulate the generalized eigenvalue problem $Az = \frac{1}{\lambda}Bz$, with

$$A = \begin{bmatrix} C & M \\ M & 0 \end{bmatrix}, \quad B = \begin{bmatrix} -K & 0 \\ 0 & M \end{bmatrix}. \quad (9.15)$$

In this case, both matrices are Hermitian, but indefinite. Linearization with (9.15) results after left multiplication of (9.14) with a block-diagonal matrix $\text{diag}(-I, -M)$. Therefore, if $\det(M) \neq 0$, then the pencil $B - \lambda A$ is regular if and only if the quadratic matrix polynomial $\lambda^2 M + \lambda C + K$ is regular.

Shifted QEP. With a shift $\lambda = \mu + \sigma$, the *shifted QEP* is

$$\left(\mu^2 \widehat{M} + \mu \widehat{C} + \widehat{K} \right) x = 0, \quad (9.16)$$

where $\widehat{M} = M$, $\widehat{C} = C + 2\sigma M$, and $\widehat{K} = K + \sigma C + \sigma^2 M$. The shift transforms eigenvalues λ of (9.1) close to σ into eigenvalues μ close to 0.

The corresponding generalized “linear” eigenvalue problem is (again in terms of λ , rather than μ)

$$\begin{bmatrix} 0 & I \\ -\widehat{K} & -\widehat{C} \end{bmatrix} \begin{bmatrix} x \\ (\lambda - \sigma)x \end{bmatrix} = (\lambda - \sigma) \begin{bmatrix} I & 0 \\ 0 & \widehat{M} \end{bmatrix} \begin{bmatrix} x \\ (\lambda - \sigma)x \end{bmatrix}$$

or

$$\begin{bmatrix} 0 & \widehat{K} \\ \widehat{K} & \widehat{C} \end{bmatrix} \begin{bmatrix} x \\ (\lambda - \sigma)x \end{bmatrix} = (\lambda - \sigma) \begin{bmatrix} \widehat{K} & 0 \\ 0 & -\widehat{M} \end{bmatrix} \begin{bmatrix} x \\ (\lambda - \sigma)x \end{bmatrix},$$

if the Hermitian of the matrix triplet $\{M, K, C\}$ wants to be preserved.

Shift-and-Invert QEP. Combining the above shift-and-invert spectral transformations (SI), the so-called *shift-and-invert QEP* becomes

$$\left(\mu^2 \widehat{M} + \mu \widehat{C} + \widehat{K} \right) x = 0, \quad (9.17)$$

where

$$\mu = \frac{1}{\lambda - \sigma},$$

and $\widehat{M} = \sigma^2 M + \sigma C + K$, $\widehat{C} = C + 2\sigma M$, and $\widehat{K} = M$. The exterior eigenvalues μ of the QEP (9.17) approximate the eigenvalues λ of the original QEP (9.1) closest to the shift σ . These eigenvalues λ are given by

$$\sigma + \frac{1}{\mu}.$$

Again, the corresponding generalized “linear” eigenvalue problem in terms of λ , rather than μ , is

$$\begin{bmatrix} -\widehat{C} & -\widehat{K} \\ I & 0 \end{bmatrix} \begin{bmatrix} x \\ (\lambda - \sigma)x \end{bmatrix} = \frac{1}{\lambda - \sigma} \begin{bmatrix} \widehat{M} & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} x \\ (\lambda - \sigma)x \end{bmatrix}$$

or

$$\begin{bmatrix} \widehat{C} & \widehat{K} \\ \widehat{K} & 0 \end{bmatrix} \begin{bmatrix} x \\ (\lambda - \sigma)x \end{bmatrix} = \frac{1}{\lambda - \sigma} \begin{bmatrix} -\widehat{M} & 0 \\ 0 & \widehat{K} \end{bmatrix} \begin{bmatrix} x \\ (\lambda - \sigma)x \end{bmatrix},$$

if the Hermitian of the matrix triplet $\{M, K, C\}$ wants to be preserved.

QEP with Cayley Transform. With the so-called Cayley transform,

$$\mu = \frac{\alpha\lambda - \beta}{\lambda - \tau},$$

where the parameters α , β , and τ are chosen such that $\alpha\tau - \beta \neq 1$, the original QEP (9.1) becomes

$$(\mu^2 \widehat{M} + \mu \widehat{C} + \widehat{K}) x = 0, \quad (9.18)$$

where $\widehat{M} = \tau^2 M + \tau C + K$, $\widehat{C} = -2\tau\beta M - (\alpha\tau + \beta)C - 2\alpha K$, and $\widehat{K} = \beta^2 M + \alpha\beta C + \alpha^2 K$. Eigenvalues λ of the original QEP (9.1) close to the *antishift* τ are transformed into large (in modulus) eigenvalues μ of the QEP (9.18). Eigenvalues λ close to the *shift* β/α correspond to eigenvalues μ of (9.17) close to 0.

Note that the triple $\{\widehat{M}, \widehat{C}, \widehat{K}\}$ is symmetric if that is the case for the real triple $\{M, C, K\}$ and if α , β , and τ are real.

9.2.4 Numerical Methods for Solving Linearized Problems

As discussed in §9.2.2, one can use either the generalized eigenvalue problem (9.4) with (9.5) or the generalized eigenvalue problem (9.9) with (9.10) for solving the corresponding quadratic eigenvalue problem (9.1).

If all matrices M , C , and K are Hermitian and M is positive definite, as in the special case (9.2), then the decision comes down to choosing either intrinsically non-Hermitian generalized eigenvalue problem (9.4) and (9.5), with a Hermitian positive definite B matrix, or a generalized eigenvalue problem (9.9) and (9.10), where both A and B matrices are Hermitian but neither of them will be positive definite.

Numerical methods discussed in Chapter 8 can be used for solving these generalized “linear” eigenvalue problems. For example, in the MSC/NASTRAN [274], the non-Hermitian formulation (9.4) with (9.5) is used.

The symmetric indefinite Lanczos method discussed in §8.6 is specifically targeted for the generalized symmetric indefinite eigenvalue problem (9.9). The potential trouble

is that in the symmetric indefinite Lanczos method, the basis vectors are orthogonal with respect to an indefinite inner product. Therefore, these basis vectors may not be linearly independent and the algorithm may break down and be numerically unstable. Nevertheless, this is often an attractive way to solve the original QEP because of potential savings in memory requirements and floating point operations. See §8.6 for further details.

9.2.5 Jacobi–Davidson Method

The possible disadvantage of the linearized approach is the doubling of the dimension of the problems; that is, a problem with n -dimensional matrices M , C , and K is transformed to a generalized problem with $2n$ -dimensional matrices A and B . This is avoided in the Jacobi–Davidson method to be discussed in this section. In this method, the QEP is first projected onto a low-dimensional subspace, which leads to a QEP of modest dimension. This low-dimensional projected QEP can be solved with any method of choice. Expansion of the subspace is realized by a Jacobi–Davidson correction equation. For polynomial eigenproblems this technique was first suggested and discussed in [408, sec. 8].

As we will see below, this method can also be applied directly to polynomial eigenproblems

$$(\lambda^\ell C_\ell + \cdots + \lambda C_1 + C_0)x = 0, \quad (9.19)$$

and no transformation to a generalized “linear” eigenvalue problem will be required, where C_i for $i = 0, 1, \dots, \ell$ are given $n \times n$ matrices. For simplicity, we will only present the case for $\ell = 2$.

In the first part of a Jacobi–Davidson iteration step, for solving the polynomial eigenproblem

$$\Psi(\lambda)x = 0 \quad (9.20)$$

where

$$\Psi(\lambda) \equiv \lambda^2 C_2 + \lambda C_1 + C_0, \quad (9.21)$$

the projected polynomial problem

$$(\theta^2 V^* C_2 V + \theta V^* C_1 V + V^* C_0 V)s = 0 \quad (9.22)$$

is solved. The columns v_i of the $n \times m$ matrix V form a basis for the search subspace. For stability reasons, the columns are constructed to be orthonormal. The projected problem has the same order as the original one but is of much smaller dimension, typically $m \ll n$. We will assume that the solution vectors s are normalized, $\|s\|_2 = 1$. First, a Ritz value θ with the desired properties, such as the largest real part or closest to some target τ , is selected and for the associated eigenvector s . Then the Ritz vector $u \equiv Vs$ and the residual $r \equiv \Psi(\theta)u$ is computed. For expansion of the search space the vector p ,

$$p \equiv \Psi'(\theta)u,$$

with

$$\Psi'(\theta) = 2\theta C_2 + C_1$$

is also computed.

In the second part of the Jacobi–Davidson iteration step, the search subspace $\text{span}(V)$ is expanded by a vector $t \perp u$ that solves (approximately) the correction equation

$$\left(I - \frac{p u^*}{u^* p} \right) \Psi(\theta) (I - u u^*) t = -r. \quad (9.23)$$

The next column of V is obtained by orthonormalizing the approximate solution against the previously computed columns v_1, \dots, v_m .

This process is repeated until an eigenpair (λ, x) has been detected, i.e., until the residual vector r is sufficiently small. The basic form of the algorithm is presented in Algorithm 9.1. We refer to [413] for an example of a quadratic eigenvalue problem arising from an acoustic problem, which has been solved with this reduction technique, for n up to 250,000.

ALGORITHM 9.1: Jacobi–Davidson Method for QEP

- (1) choose an $n \times m$ orthonormal matrix V
- (2) **for** $i = 0, 1, 2$
- (3) compute $W_i = C_i V$ and $M_i = V^* W_i$
- (4) **end for**
- (5) iterate until convergence
- (6) compute the eigenpairs (θ, s) of $(\theta^2 M_2 + \theta M_1 + M_0) s = 0$.
- (7) select the desired eigenpair (θ, s) with $\|s\|_2 = 1$.
- (8) compute $u = V s$, $p = \Psi'(\theta) u$, $r = \Psi(\theta) u$.
- (9) **if** $(\|r\|_2 < \epsilon)$, $\lambda = \theta$, $x = u$, **stop**
- (10) solve (approximately) a $t \perp u$ from
- (11)
$$\left(I - \frac{p u^*}{u^* p} \right) \Psi(\theta) (I - u u^*) t = -r$$
- (12) orthogonalize t against V , $v = t / \|t\|_2$
- (13) **for** $i = 0, 1, 2$
- (14) compute $w_i = C_i v$
- (15)
$$M_i = \begin{bmatrix} M_i & V^* w_i \\ v^* W_i & v^* w_i \end{bmatrix}, W_i = [W_i, w_i]$$
- (16) **end for**
- (17) expand $V = [V, v]$

If the dimension of the search subspace becomes too large, if the number of columns of V is equal to, say, $m = m_{\max}$, then the process can be continued with a search subspace of smaller dimension. Take for the new search subspace the space spanned by the best m_{\min} Ritz pairs from the last step; that is, take $V = [u_1, \dots, u_{m_{\min}}]$, where $u_1, \dots, u_{m_{\min}}$ are the Ritz vectors associated with the best available m_{\min} Ritz values $\theta_1, \dots, \theta_{m_{\min}}$. Then apply modified Gram–Schmidt to orthonormalize V and restart with this matrix. Note that the new search matrix $V = V_{m_{\min}}$ can be expressed in terms of the old search matrix $V = V_{m_{\max}}$ as $V_{m_{\min}} = V_{m_{\max}} T$ for some $m_{\max} \times m_{\min}$ matrix T . The transformation matrix T can be computed explicitly and can be used to update the auxiliary matrices W_i ($= W_i T$) and M_i ($= T^* M_i T$).

Eigenvectors that already have been detected can be kept in the search subspace (explicit deflation) if more eigenvectors are wanted. Keeping the eigenvectors in the search subspace prevents the process from reconstructing known eigenvectors.

In §4.7.3 and §8.4.2, we suggested using “implicit” deflation, since that approach is based on Schur forms and this is more stable, with better conditioned correction equations. However, in this general polynomial setting, it is not clear how to incorporate implicit deflation: the Schur form and the generalized Schur form cannot easily be generalized.

9.2.6 Notes and References

Numerical algorithm design and analysis for the solution of QEPs are still an active research topic. Besides those methods discussed in this section, some alternative methods are available in the literature; see, for example, [284, 277, 372, 102, 209]. Most of these methods are variants of Newton’s method. They generally have good local convergence properties and find one eigenpair at a time. In [209], a proper deflation technique was presented for finding more than one eigenpair.

Recently, backward errors and condition of numbers for the quadratic eigenvalue problem, and more generally, the polynomial eigenvalue problem, was presented in [435]. In [432], a perturbation analysis of the quadratic eigenvalue problem (9.2) was presented.

9.3 Higher Order Polynomial Eigenvalue Problems

Some applications lead to a higher order polynomial eigenvalue problem (PEP)

$$\Psi(\lambda)x = 0 \quad \text{and} \quad y^*\Psi(\lambda) = 0, \quad (9.24)$$

where $\Psi(\lambda)$ is a matrix polynomial defined as

$$\Psi(\lambda) \equiv \lambda^\ell C_\ell + \lambda^{\ell-1} C_{\ell-1} + \cdots + \lambda C_1 + C_0,$$

in which the C_j are square n by n matrices. A thorough study of the mathematical properties of matrix polynomials can be found in [194].

In order to make the eigenvalue problem well defined, these matrices have to satisfy certain properties; in particular C_ℓ should be nonsingular. Similar to the quadratic problem, these problems can also be linearized to

$$Az = \lambda Bz,$$

where

$$A \equiv \begin{bmatrix} 0 & I & 0 & \cdots & 0 \\ 0 & 0 & I & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \vdots & \ddots & I \\ -C_0 & -C_1 & -C_2 & \cdots & -C_{\ell-1} \end{bmatrix}, \quad B \equiv \begin{bmatrix} I & & & & \\ & I & & & \\ & & \ddots & & \\ & & & I & \\ & & & & C_\ell \end{bmatrix}.$$

The relation between x and z is given by $z = (x^T, \lambda x^T, \dots, \lambda^{\ell-1} x^T)^T$. $B^{-1}A$ is a block companion matrix of the PEP. The generalized eigenproblem $Az = \lambda Bz$ can be solved with one of the methods discussed in Chapter 8. A disadvantage of this approach

is that one has to work with larger matrices of order $n \times \ell$, and these matrices also have $n \times \ell$ eigenpairs, of course. This implies that one has to check which of the computed eigenpairs satisfies the original polynomial equation. Ruhe [372] (see also Davis [102]) discussed methods that directly handle the problem (9.24), for instance, with Newton's method. For larger values of n one may expect all sorts of problems with the convergence of these techniques. In §9.2.5, we have discussed a method that can be used to attack problems with large n . In that approach, one first projects the given problem (9.24) onto a low-dimensional subspace and obtains a similar problem of low dimension. This low-dimensional polynomial eigenproblem can then be solved with one of the approaches mentioned above. In [221] a fourth-order polynomial problem has been solved successfully, using this reduction technique.

9.4 Nonlinear Eigenvalue Problems with Orthogonality Constraints

R. Lippert and A. Edelman

9.4.1 Introduction

This section increases the scope of the algebraic eigenvalue problem by focusing on the geometrical properties of the eigenspaces. We discuss a template that solves variational problems defined on sets of subspaces. Areas where such problems arise include electronic structures computation, eigenvalue regularization, control theory, signal processing, and graphics camera calibration. The underlying geometry also provides the numerical analyst theoretical insight into the common mathematical structure underlying eigenvalue algorithms (see [151, 155, 416]).

Suppose that one wishes to optimize a real-valued function $F(Y)$ over Y such that $Y^*Y = I$, where Y^* is either transpose or Hermitian transpose as appropriate. Our template is designed as a means to optimize such an F .

One simple case is optimization over square orthogonal (or unitary) matrices such as the least squares simultaneous diagonalization of symmetric matrices (also known as INDSCAL [107]) problem described later. Another simple case is optimization over the unit sphere, as in symmetric Rayleigh quotient minimization. In between, we have rectangular $n \times p$ ($n \geq p$) matrices Y with orthonormal columns such as the orthogonal Procrustes problem.

Furthermore, some functions F may have the symmetry property $F(Y) = F(YQ)$ for all orthogonal (unitary) Q with a specified block-diagonal structure, which causes some search directions to have no effect on the value of F . For example, if A is a symmetric matrix, $n \geq p$, and $F(Y) = \text{tr}(Y^*AY)$, then $F(Y) = F(YQ)$ for all $p \times p$ orthogonal Q . More generally, some functions F have the property that $F(Y) = F(YQ)$, where Q is orthogonal (unitary) with the block-diagonal form

$$Q = \begin{bmatrix} Q_1 & 0 & \cdots & 0 \\ 0 & Q_2 & \cdots & 0 \\ \vdots & \ddots & \cdots & \vdots \\ 0 & 0 & \cdots & Q_p \end{bmatrix},$$

where the Q_i are orthogonal (unitary) matrices. More complicated problems with block-diagonal orthogonal (unitary) symmetry can arise in eigenvalue regularization

problems (an example of such a problem is the sample nearest-Jordan block problem solved in the examples).

This chapter is organized as follows. §9.4.2 discusses the basics of calling the template code. §9.4.3 discusses the objective functions and their derivatives for the example problems explored in this chapter. §9.4.4 contains sample runs of instances of the example problems. §9.4.5 explains the code structure and the places where a user may wish to make modifications. §9.4.6 will cover some of the basic mathematics concerning the geometry of the Stiefel manifold.

9.4.2 MATLAB Templates

The templates are ready for immediate use or as a departure point for generalization,³ e.g. problems over multiple variables with orthogonality constraints, or code optimizations. In the simplest mode, the user needs only to supply a function to minimize $F(Y)$ (and first and second derivatives, optionally) in $F.m$ (in $dF.m$ and $ddF.m$) and an initial guess Y_0 . The calling sequence is then a single call to sg_min (named in honor of Stiefel and Grassmann):

```
[fopt, yopt] = sg_min(Y0).
```

For example, if the function $F.m$ has the form

```
function f=F(Y)
f=trace( Y' * diag(1:10) * Y * diag(1:3) );
```

we can call $sg_min(rand(10,3))$, which specifies a random starting point.

We strongly recommend also providing first derivative information:

```
function df=dF(Y)
df = 2 * diag(1:10) * Y * diag(1:3);
```

The code can do finite differences, but it is very slow and problematic. Second derivative information can also be provided by the user (this is not nearly as crucial for speed as providing first derivative information, but may improve accuracy):

```
function ddf=ddF(Y,H)
ddf = 2 * diag(1:10) * H * diag(1:3);
```

A sample test call where $F(Y)$ is known to have optimal value 10 is

```
>> rand('state',0); % initialize random number generator
>> fmin = sg_min(rand(10,3))
iter      grad          F(Y)          flops      step type
0        1.877773e+01    3.132748e+01      2470      none
invgrad: Hessian not positive definite, CG terminating early
1        1.342879e+01    2.011465e+01      163639      Newton step
invgrad: Hessian not positive definite, CG terminating early
2        1.130915e+01    1.368044e+01      344198      Newton step
invgrad: Hessian not positive definite, CG terminating early
3        5.974819e+00    1.063045e+01      515919      Newton step
invgrad: max iterations reached inverting the Hessian by CG
```

³See the book's homepage, ETHOME, for access to the software described in this section.

```

4      1.135417e+00  1.006835e+01    789520      Newton step
5      5.526359e-02  1.000009e+01    1049594      Newton step
6      5.072118e-05  1.000000e+01    1337540      Newton step
7      1.276025e-06  1.000000e+01    1762455      Newton step

```

```
fmin =
10.0000
```

The full calling sequence to `sg_min` is

```
[fopt, yopt]=sg_min(Y0,rc,mode,metric,motion,verbose, ...
gradtol,ftol,partition),
```

where `Y0` is required and the optional arguments are (see Table 9.1):

`rc` specifies whether the matrices will have complex entries or not. Although most of the code is insensitive to this issue, `rc` is vital for counting the dimension of the problem correctly. When omitted, `sg_min` guesses are based on whether `Y0` has nonzero imaginary part.

`mode` selects the optimization method that will be used. '`newton`' selects Newton's method with a conjugate gradient-based inversion of the Hessian. '`dog`' selects a dog-leg step algorithm which interpolates a steepest descent and a Newton's method step. '`frcg`' selects a Fletcher-Reeves conjugate gradient. Lastly, '`prcg`' selects a Polak-Ribiere conjugate gradient, which has the advantage that it does not require a very accurate Hessian (and thus it is the safest of these methods if one uses a finite difference approximation to implement `ddF.m`). While '`newton`' is the default, this is by no means our endorsement of it over the other methods, which might be more accurate and less expensive for some problems.

`metric` selects the kind of geometry with which to endow the constraint surface. This ultimately affects the definition of the covariant Hessian. '`flat`' projects the result of applying the unconstrained Hessian onto the tangent space, while '`euclidean`' and '`canonical`' add on *connection* terms specific to their geometries. '`euclidean`' is the default.

Table 9.1: A short list of the optional arguments for `sg_min`

Argument	Description
<code>rc</code>	{'real', 'complex' }
<code>mode</code>	{'newton', 'dog', 'prcg', 'frcg' }
<code>metric</code>	{'flat', 'euclidean', 'canonical' }
<code>motion</code>	{'approximate', 'exact' }
<code>verbose</code>	{'verbose', 'quiet'}
<code>ftol</code>	First convergence tolerance
<code>gradtol</code>	Second convergence tolerance
<code>partition</code>	Partition of p indicating symmetries of f

motion selects whether line movement along the manifold will be by the analytic solution to the geodesic equations of motions for the metric selected or by a computationally less expensive approximation to the solution (default). (For a 'flat' metric, there is no geodesic equation, so this argument has no effect in that case.)

verbose determines whether the function will display reports on each iteration while the function executes. When this argument is 'verbose' (the default) data will be displayed and also recorded in the global **SGdata**. When this argument is 'quiet' then no convergence data is displayed or recorded.

gradtol and **ftol** We declare convergence if both of two conditions are true:

grad/**gradinit** < **gradtol** (default **1e-7**) or (**f-fold**)/**f** < **ftol** (default **1e-10**) where **gradinit** is the initial magnitude of the gradient and **fold** is the value of $F(Y)$ at the last iteration.

partition is a cell array whose elements are vectors of indices that represent a disjoint partition of **1:p**. If F has no symmetry, then the partition is **num2cell(1:p)**. If $F(Y) = F(YQ)$ for all orthogonal Q , then the partition is **{1:p}**. The partition is **{1:2,3:5}** if the symmetry in F is $F(Y) = F(YQ)$ for orthogonal Q with sparsity structure

$$\begin{bmatrix} \times & \times \\ \times & \times \\ & & \times & \times & \times \\ & & \times & \times & \times \\ & & \times & \times & \times \end{bmatrix}.$$

The user could equally well specify **{3:5,1:2}** or **{[5 3 4],[2 1]}** for the partition; i.e., a partition is a set of sets. The purpose of the partition is to project away the null space of the Hessian resulting from any block-diagonal orthogonal symmetries of $F(Y)$. If the argument is omitted, our code will pick a partition by determining the symmetries of F (using its **partition.m** function). However, the user should be aware that a wrong partition can destroy convergence.

9.4.3 Sample Problems and Their Differentials

This section serves as a sample guide on the manipulation of objective functions of matrices with orthonormal columns. We have found a few common tricks worth emphasizing.

Once one has a formula for the objective function $F(Y)$, we define the formula for $dF(Y)$ implicitly by $\text{tr}(V^*dF(Y)) = \frac{d}{dt}F(Y(t))|_{t=0}$, where $Y(t) = Y + tV$ (or any curve $Y(t)$ for which $\dot{Y}(0) = V$). The reader may recall that $\text{tr}(A^*B) = \sum_{i,j} A_{ij}B_{ij}$, so it functions just like the real inner product for vectors,⁴ and the implicit definition of $dF(Y)$ is actually the directional derivative interpretation of the gradient of $F(Y)$ as an unconstrained function in a Euclidean space.

For most of the functions we have used in our examples, the easiest way to obtain the formula for dF is to actually use the implicit definition.

For example, if $F(Y) = \text{tr}(AY^*BY)$ one then has

$$\text{tr}(V^*dF(Y)) = \text{tr}(AV^*BY + AY^*BV).$$

⁴Over complex numbers, we always take the real part of $\text{tr}(A^H B)$.

Since the value of the trace is invariant under cyclic permutations of products and transposes, we may rewrite this equation as

$$\text{tr}(V^* dF(Y)) = \text{tr}(V^* BYA + V^* B^* YA^*),$$

and, since V is unrestricted, this implies that $dF(Y) = BYA + B^*Y^*A^*$.

The process we recommend is:

- try to write $F(Y)$ as a trace;
- compute $\frac{d}{dt} F(Y(t))|_{t=0}$, where we let $V = \dot{Y}(t)$;
- use trace identities to rewrite every term to have a V^* in the front;
- strip off the V^* leaving the $dF(Y)$.

As a check, we recommend using the finite difference `dF.m` code supplied in the subdirectory `finitendiff` to check derivations before proceeding.

The software needs a function called `ddF.m`, which returns $\frac{d}{dt} dF(Y(t))|_{t=0}$ for $\dot{Y}(0) = H$. The sort of second derivative information required by the software is easier to derive than the first. If one has an analytic expression for $dF(Y)$, then one need only differentiate.

If, for some reason, the computation for `ddF.m` costs much more than two evaluations of $dF(Y)$ with `dF.m`, the reader may just consider employing the finite difference function for `ddF.m` found in `finitendiff` (or simply use `ddF.m` as a check).

It is, however, strongly suggested that one use an analytic expression for computing $dF(Y)$, as the finite difference code for it requires a large number of function evaluations ($2np$).

9.4.3.1 The Procrustes Problem

The Procrustes problem (see [158]) is the minimization of $\|AY - YB\|_F$ for constant A and B over the manifold $Y^*Y = I$. This minimization determines the nearest matrix \hat{A} to A for which

$$Q^* \hat{A} Q = \begin{bmatrix} B & * \\ 0 & * \end{bmatrix};$$

i.e. the columns of B span an invariant subspace of \hat{A} .

The differential of $F(Y) = \frac{1}{2}\|AY - YB\|_F^2 = \frac{1}{2}\text{tr}(AY - YB)^*(AY - YB)$ is given by

$$dF(Y) = A^*(AY - YB) - (AY - YB)B^*.$$

This can be derived following the process outlined above. Observe that

$$\begin{aligned} \frac{d}{dt} F(Y(t))|_{t=0} &= \frac{1}{2}\text{tr}((AV - VB)^*(AY - YB) + (AY - YB)^*(AV - VB)) \\ &= \text{tr}((AV - VB)^*(AY - YB)) \\ &= \text{tr}(V^*(A^*(AY - YB)) - V^*(AY - YB)B^*). \end{aligned}$$

The second derivative of $F(Y)$ is given by the equation

$$\frac{d}{dt} dF(Y(t))|_{t=0} = A^*(AH - HB) - (AH - HB)B^*,$$

where $\dot{Y}(0) = H$, which can be obtained by varying the expression for dF .

9.4.3.2 Nearest-Jordan Structure

Now suppose that the B block in the Procrustes problem is allowed to vary with Y . Moreover, suppose that $B(Y)$ is in the nearest staircase form to Y^*AY ; that is,

$$B(Y) = \begin{bmatrix} \lambda_1 I & * & * \\ 0 & \lambda_2 I & * \\ 0 & 0 & \dots \end{bmatrix}$$

for fixed block sizes, where the $*$ -elements are the corresponding matrix elements of Y^*AY and the $\lambda_i I$ blocks are either fixed or determined by some heuristic, e.g., taking the average trace of the blocks they replace in Y^*AY . Then a minimization of $\|AY - YB(Y)\|_F$ finds the nearest matrix as a particular Jordan structure, where the structure is determined by the block sizes and the eigenvalues are λ_i . When the λ_i are fixed, we call this the *orbit* problem, and when the λ_i are selected by the heuristic given we call this the *bundle* problem.

Such a problem can be useful in regularizing the computation of Jordan structures of matrices with ill-conditioned eigenvalues.

The form of the differential of $F(Y)$, surprisingly, is the same as that of $F(Y)$ for the Procrustes problem

$$dF(Y) = A^*(AY - YB(Y)) - (AY - YB(Y))B(Y)^*.$$

This is because $\text{tr}(-(AY - YB(Y))^*Y\dot{B}) = \text{tr}((B(Y) - Y^*AY)^*\dot{B}) = 0$ for the B selected as above for either the orbit or the bundle case, where $\dot{B} = \frac{d}{dt}B(Y(t))|_{t=0}$.

In contrast, the form of the second derivatives is a bit more complicated, since B now depends on Y :

$$\frac{d}{dt}dF(Y(t))|_{t=0} = \frac{d}{dt}dF_{\text{Proc}}(Y(t))|_{t=0} - A^*Y\dot{B} - AY\dot{B}^* + YB\dot{B}^* + Y\dot{B}B^*,$$

where $\dot{Y}(0) = H$, dF_{Proc} is just short for the Procrustes (B constant) part of the second derivative, and $\dot{B} = \frac{d}{dt}B(Y(t))|_{t=0}$, which is the staircase part of $Y^*AH + H^*AY$ (with trace averages or zeros on the diagonal depending on whether bundle or orbit).

9.4.3.3 Trace Minimization

In this case we consider a “Rayleigh quotient” style of iteration to find the eigenspaces of the smallest eigenvalues of a symmetric positive definite matrix A . That is, we can minimize $F(Y) = \frac{1}{2}\text{tr}(Y^*AY)$ [14, 183, 392]. The minimizer Y^* will be an $n \times p$ matrix whose columns span the eigenspaces of the lowest p eigenvalues of A .

For this problem,

$$dF(Y) = AY,$$

it is easily seen that

$$\frac{d}{dt}dF(Y(t))|_{t=0} = AH,$$

where $\dot{Y}(0) = H$.

9.4.3.4 Trace Minimization with a Nonlinear Term

We consider a nonlinear eigenvalue problem related to the trace minimization of §9.4.3.3: minimize $F(Y) = \frac{1}{2}(\text{tr}(Y^*AY) + g(Y))$, where $g(Y)$ is some nonlinear term.

This sort of minimization occurs in electronic structures computations such as local density approximations (LDAs), where the columns of Y represent electron wave functions, A is a fixed Hamiltonian, and $g(Y)$ represents the energy of the electron-electron interactions (see, for example, [151, 428]). In this case, the optimal Y represents the state of lowest energy of the system.

The only additions to the differentials of the trace minimization are terms from $g(Y)$ which vary from problem to problem. For our example, we have taken $g(Y) = \frac{K}{4} \sum_i \rho_i^2$ for some coupling constant K , where $\rho_i = \sum_j |Y_{ij}|^2$ (the charge density).

In this case

$$dF(Y) = dF_{\text{tracemin}} + K \text{diag}(\rho)Y,$$

and

$$\frac{d}{dt} dF(Y(t))|_{t=0} = \frac{d}{dt} dF_{\text{tracemin}}(Y(t))|_{t=0} + K \text{diag}(\rho)H + K \text{diag}\left(\frac{d}{dt}\rho\right)Y,$$

where $\dot{Y}(0) = H$ and $\frac{d}{dt}\rho_i|_{t=0} = \sum_j Y_{ij}H_{ij}$.

9.4.3.5 Simultaneous Schur Decomposition Problem

Consider two matrices A and B which in the absence of error have the same Schur vectors; i.e., there is a $Y \in \mathcal{O}(n)$ such that Y^*AY and Y^*BY are both block upper triangular where $\mathcal{O}(n)$ is the set of n by n orthogonal matrices. Now suppose that A and B are somewhat noisy from measurement errors or some other kind of lossy filtering. In that case the Y that upper triangularizes A might not upper triangularize B as well. How does one find the best Y ?

This is a problem that was presented to us by Schilders [396], who phrased it as a least squares minimization of $F(Y) = \frac{1}{2}(\|\text{low}(Y^*AY)\|_F^2 + \|\text{low}(Y^*BY)\|_F^2)$, where $\text{low}(M)$ is a mask returning the block lower triangular part of M , where M is broken up into 2×2 blocks.

For this problem the differential is a bit tricky and its derivation instructive:

$$\begin{aligned} \text{tr}(V^*dF(Y)) &= \text{tr}(\text{low}(Y^*AY)^*\text{low}(Y^*AV + V^*AY) \\ &\quad + \text{low}(Y^*BY)^*\text{low}(Y^*BV + V^*BY)), \\ \text{tr}(V^*dF(Y)) &= \text{tr}(\text{low}(Y^*AY)^*(Y^*AV + V^*AY) \\ &\quad + \text{low}(Y^*BY)^*(Y^*BV + V^*BY)), \\ \text{tr}(V^*dF(Y)) &= \text{tr}(V^*(AY\text{low}(Y^*AY)^* + A^*Y\text{low}(Y^*AY) \\ &\quad + BY\text{low}(Y^*BY)^* + B^*Y\text{low}(Y^*BY))), \\ dF(Y) &= AY\text{low}(Y^*AY)^* + A^*Y\text{low}(Y^*AY) \\ &\quad + BY\text{low}(Y^*BY)^* + B^*Y\text{low}(Y^*BY), \end{aligned}$$

where the second equation results from observing that $\text{tr}(\text{low}(M)^*\text{low}(N)) = \text{tr}(\text{low}(M)^*N)$ and the third from properties of the trace.

With second derivatives given by

$$\begin{aligned} \frac{d}{dt} dF(Y(t))|_{t=0} = & AH\text{low}(Y^*AY)^* + A^*H\text{low}(Y^*AY) + AY\text{low}\left(\frac{d(Y^*AY)}{dt}\right)^* \\ & + A^*Y\text{low}\left(\frac{d(Y^*AY)}{dt}\right) + BH\text{low}(Y^*BY)^* + B^*H\text{low}(Y^*BY) \\ & + BY\text{low}\left(\frac{d(Y^*BY)}{dt}\right)^* + B^*Y\text{low}\left(\frac{d(Y^*BY)}{dt}\right), \end{aligned}$$

where $\dot{Y}(0) = H$, $\frac{d(Y^*AY)}{dt}|_{t=0} = H^*AY + Y^*AH$ and $\frac{d(Y^*BY)}{dt}|_{t=0} = H^*BY + Y^*BH$.

9.4.3.6 Simultaneous Diagonalization

Another problem like the simultaneous Schur problem, involving sets of matrices with similar structures, is the problem of finding the best set of eigenvectors for a set of symmetric matrices given that the matrices are known to be simultaneously diagonalizable, but may have significant errors in their entries from noise or measurement error. Instances of this problem arise in the psychometric literature, where it is called an INDSCAL problem [107].

Phrased in terms of a minimization, one has a set of symmetric matrices A_i and wishes to find $Y \in \mathcal{O}(n)$ that minimizes $F(Y) = \frac{1}{2} \sum_i \| [Y, A_i] \|_F^2 = \frac{1}{2} \sum_i \| YA_i - A_i Y \|_F^2$.

We then have

$$dF(Y) = \sum_i [[Y, A_i], A_i^*]$$

and

$$\frac{d}{dt} dF(Y(t))|_{t=0} = \sum_i [[H, A_i], A_i^*],$$

where $\dot{Y}(0) = H$.

9.4.4 Numerical Examples

We have provided implementations for the sample problems described.

We present the examples and their verbose outputs. We have included these outputs so the reader can check that his/her copy of the package is executing properly. The user should be running MATLAB (version 5 or compatible) in the following directory:

```
>> !ls
Fline.m      dgrad.m      grad.m          move.m      sg_min.m
README       dimension.m   gradline.m     nosym.m    sg_newton.m
clamp.m      dtangent.m   invgrad_CG.m  partition.m sg_prcg.m
connection.m examples     invgrad_MINRES.m sg_dog.m   tangent.m
dFline.m     finitediff   ip.m           sg_frcg.m
```

(Note that there may also be the additional subdirectories docs/ or @cell/.)

Each example problem has a subdirectory in the examples subdirectory.

```
>> !ls examples
jordan      ldatoy      procrustes  simdiag      simschrur  tracemin
```

Each of these subdirectories contains an implementation for $F(Y)$ ($dF(Y)$ and $ddF(Y, H)$), which sets the (global) parameters of $F(Y)$ (this function must be called before any other), a `guess` function to generate an initial guess, and possibly some auxiliary functions for computing $F(Y)$, related quantities, or instances of a specific problem.

```
>> !ls examples/lldatoy
F.m           dF.m       guess.m
Kinetic.m     ddF.m     parameters.m
>> !ls examples/jordan
Block.m       dBlock.m   ddF.m       guess.m      post.m
F.m           dF.m       frank.m    parameters.m
```

We executed the examples in the supported optimization modes ('newton', 'dog', 'frcg', and 'prcg') and the 'euclidean' metric. For different instances of the same problem, different modes might perform best, so the reader should not feel that a particular mode will exhibit superior performance in all instances.

9.4.4.1 A Sample Procrustes Problem

In this example, we attempt to find a Y which minimizes $\|AY - YB\|_F$ for a pair of randomly selected A (12×12) and B (4×4).

```
>> !cp examples/procrustes/*.m .
>> randn('state',0);
>> [A,B] = randprob;
>> parameters(A,B);
>> Y0 = guess;
>> [fn,Yn] = sg_min(Y0,'newton','euclidean');
iter      grad          F(Y)          flops          step type
0        2.334988e+01  3.071299e+01    4751          none
  invdgrad: Hessian not positive definite, CG terminating early
1        1.171339e+01  1.376463e+01    365678        Newton step
  invdgrad: Hessian not positive definite, CG terminating early
2        7.843279e+00  7.616381e+00    677599        Newton step
  invdgrad: Hessian not positive definite, CG terminating early
3        5.131680e+00  4.945824e+00    992823        Newton step
  invdgrad: Hessian not positive definite, CG terminating early
4        5.642834e+00  3.512826e+00   1293761        Newton step
  invdgrad: Hessian not positive definite, CG terminating early
5        5.500553e+00  1.721329e+00   1607969        Newton step
  invdgrad: Hessian not positive definite, CG terminating early
6        4.666307e+00  1.192561e+00   1964675        Newton step
  invdgrad: Hessian not positive definite, CG terminating early
7        3.576069e+00  6.850532e-01   2272355        Newton step
  invdgrad: max iterations reached inverting the Hessian by CG
8        1.228119e+00  3.046816e-01   2820625        Newton step
9        4.673779e-02  2.506848e-01   3345266        Newton step
10       6.411668e-04  2.505253e-01   3873582        Newton step
11       1.965463e-06  2.505253e-01   4430245        Newton step
12       1.620267e-06  2.505253e-01   5022629        Newton step
```

Figure 9.1 shows the convergence curve for this run.

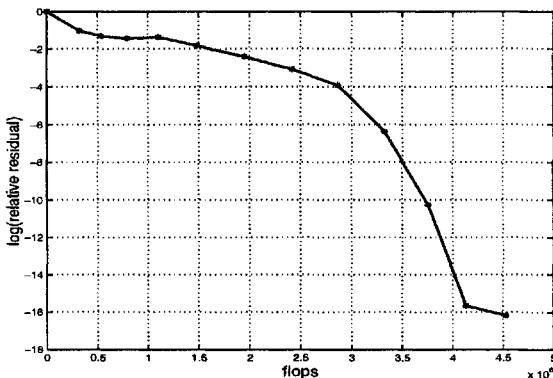


Figure 9.1: Procrustes problem

9.4.4.2 A Sample Jordan Block Problem

We now look for a nearby matrix with a Jordan structure $J_4(\alpha) \oplus J_2(\beta) \oplus (\text{anything})$ to the 12×12 Frank matrix, a matrix whose Jordan structure is known to be very difficult.

```
>> !cp examples/jordan/*.m .
>> A = frank(12);
>> evs = sort(eig(A)); eigvs = [mean(evs(1:4)) mean(evs(5:6))];
>> parameters(A,eigvs,[4; 2],'bundle')
    1 Jordan block of order 4 of eigenvalue 0.076352
    1 Jordan block of order 2 of eigenvalue 0.464128
>> Y0 = guess;
>> [fn,Yn] = sg_min(Y0,'dog','euclidean');
iter      grad          F(Y)            flops      step type
0       1.775900e-02   9.122159e-06     16257      none
invdgrad: max iterations reached inverting the Hessian by MINRES
1       1.011261e-03   4.121994e-07     4638811      good dog
invdgrad: max iterations reached inverting the Hessian by MINRES
2       4.250874e-04   3.523843e-07     9274101      good dog
invdgrad: max iterations reached inverting the Hessian by MINRES
3       2.146629e-03   1.696542e-07    13936305      good dog
invdgrad: max iterations reached inverting the Hessian by MINRES
4       5.260491e-04   5.326314e-08    18553494      good dog
invdgrad: max iterations reached inverting the Hessian by MINRES
5       4.861126e-04   7.545415e-09    23231554      good dog
invdgrad: max iterations reached inverting the Hessian by MINRES
6       4.228777e-05   3.014858e-09    27846197      good dog
invdgrad: max iterations reached inverting the Hessian by MINRES
7       2.954536e-06   2.924391e-09    32508118      good dog
8       2.868836e-08   2.924238e-09    37207124      good dog
9       2.093780e-09   2.924238e-09    38384200      good dog
10      2.479121e-09   2.924238e-09    39163025      good dog
11      1.713240e-09   2.924238e-09    39921924      bad dog
```

Figure 9.2 shows the convergence curve for this run.

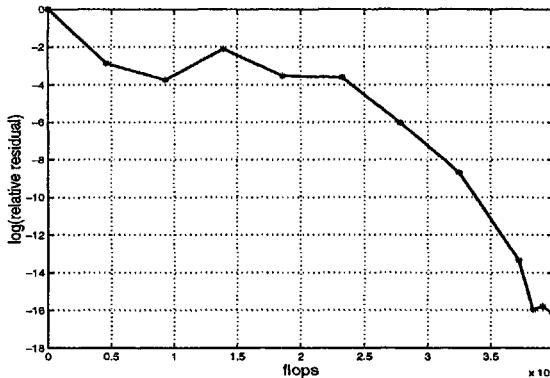


Figure 9.2: Jordan problem

9.4.4.3 A Sample Trace Minimization Problem

In this problem we minimize $\text{trace}(Y^*AY)$, where A is a 12×12 second difference operator in one dimension and Y is 12×4 .

Note: we do not recommend that this problem be solved this way, since, for constant A , it can be done faster with conventional eigenvalue solvers.

```
>> !cp examples/tracemin/*.m .
>> randn('state',0);
>> A = Kinetic(12);
>> parameters(A);
>> Y0 = guess(4);
>> [fn,Yn] = sg_min(Y0,'frcg','euclidean');
iter      grad          F(Y)           flops
0       2.249340e+00   4.339273e+00    3807
1       1.635681e+00   1.774835e+00   106770
2       6.506188e-01   1.160615e+00   196319
3       4.400435e-01   1.043566e+00   295232
4       3.460854e-01   9.601858e-01   386244
5       3.418816e-01   9.122465e-01   471214
6       2.591414e-01   8.543284e-01   560763
7       1.225067e-01   8.319959e-01   661414
8       5.660065e-02   8.280403e-01   785192
9       2.346852e-02   8.271969e-01   935587
10      9.258692e-03   8.270659e-01   1105819
11      4.723177e-03   8.270398e-01   1261468
12      3.282461e-03   8.270316e-01   1417625
13      1.984453e-03   8.270281e-01   1576710
14      9.808233e-04   8.270270e-01   1726651
15      4.339561e-04   8.270267e-01   1876592
16      1.990167e-04   8.270267e-01   2022569
```

17	6.563434e-05	8.270267e-01	2163632
18	3.165945e-05	8.270267e-01	2322727
19	2.015557e-05	8.270267e-01	2468720
20	1.261539e-05	8.270267e-01	2612825
21	7.123104e-06	8.270267e-01	2748702
22	2.552485e-06	8.270267e-01	2935969
23	9.679096e-07	8.270267e-01	3119232
24	3.168742e-07	8.270267e-01	3287067
25	2.062080e-07	8.270267e-01	3472366

Figure 9.3 shows the convergence curve for this run.

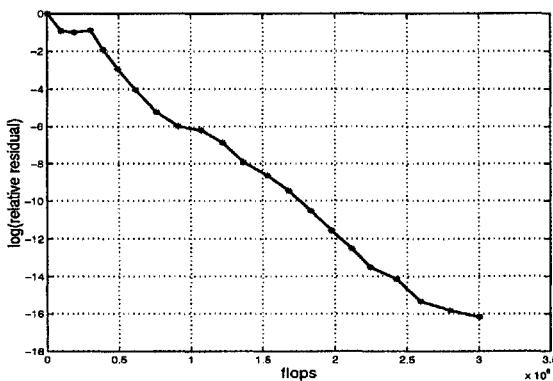


Figure 9.3: Trace minimization problem

9.4.4.4 A Sample LDA Toy Problem (Trace Minimization with Nonlinear Term)

We attempt to solve a toy LDA problem described in §9.4.3.4 for four electrons on a one-dimensional grid with 12 points and a coupling constant of 20. In this case, we use the first four eigenvectors of the linear problem as a starting point.

```
>> !cp examples/ldatoy/*.m .
>> K = Kinetic(12);
>> parameters(K,20);
>> Y0 = guess(4);
>> [fn,Yn] = sg_min(Y0,'newton','euclidean');
iter      grad          F(Y)          flops      step type
0        2.432521e+00    7.750104e+00    4554      none
invdgrad: Hessian not positive definite, CG terminating early
1        4.770038e-01    7.546426e+00    367756    steepest step
2        4.664971e-02    7.531450e+00    789739    Newton step
3        3.363736e-04    7.531232e+00   1284452    Newton step
4        1.809273e-07    7.531232e+00   1976096    Newton step
5        1.843536e-07    7.531232e+00   2778377    Newton step
```

Figure 9.4 shows the convergence curve for this run.

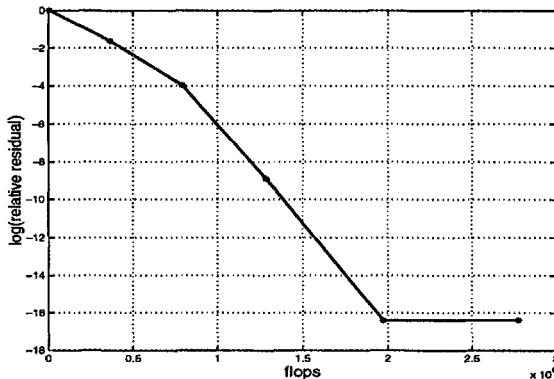


Figure 9.4: LDA toy problem

9.4.4.5 A Sample Simultaneous Schur Decomposition Problem

In this case we use the sample problem given to us by Schilders.

```
>> !cp examples/simschur/*.m .
>> [A,B] = noisy;
>> parameters(A,B);
>> Y0 = guess;
>> [fn,Yn] = sg_min(Y0,'prcg','euclidean');
iter      grad          F(Y)          flops
0        2.205361e-01    1.687421e-02      56493
1        1.087502e-01    8.329681e-03     1537489
2        5.834580e-02    5.947527e-03     3045842
3        3.954742e-02    5.322286e-03     4243901
4        1.920065e-02    5.016544e-03     5700349
5        9.449972e-03    4.938962e-03     7062826
6        4.304992e-03    4.920311e-03     8509245
7        2.649005e-03    4.916373e-03     9982763
8        1.839592e-03    4.914510e-03    11492687
9        1.162756e-03    4.913601e-03    12952876
10       5.450777e-04    4.913326e-03    14401149
11       2.461091e-04    4.913273e-03    15848402
12       1.463300e-04    4.913261e-03    17354270
13       7.851036e-05    4.913257e-03    18762063
14       5.400429e-05    4.913256e-03    20205661
15       2.786051e-05    4.913255e-03    21689844
16       1.570592e-05    4.913255e-03    23166944
17       8.390343e-06    4.913255e-03    24528843
18       4.060847e-06    4.913255e-03    25901423
19       2.047984e-06    4.913255e-03    27271928
20       1.161407e-06    4.913255e-03    28481228
21       7.367792e-07    4.913255e-03    29776935
```

22	4.640484e-07	4.913255e-03	31533888
23	2.206626e-07	4.913255e-03	33245809
24	1.467168e-07	4.913255e-03	35084169
25	7.946751e-08	4.913255e-03	36975859
26	4.631213e-08	4.913255e-03	38934071
27	2.119895e-08	4.913255e-03	40761932

Figure 9.5 shows the convergence curve for this run.

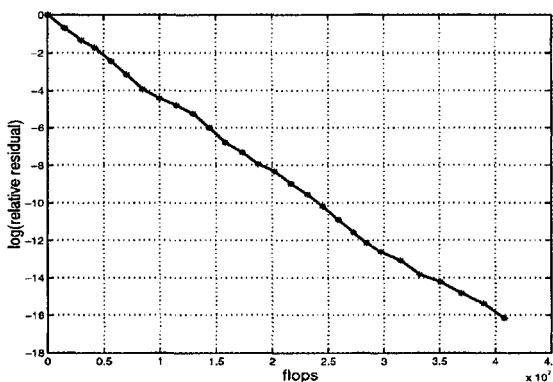


Figure 9.5: Simultaneous Schur problem

9.4.4.6 A Sample Diagonalization Problem

We attempt to simultaneously diagonalize two noisy (random noise) versions of `diag(1:10)` and `diag(1:10)^2`. Our initial point is a small random perturbation of the identity.

```
>> !cp examples/simdiag/*.* .
>> randn('state',0);
>> [A,B] = noisy;
>> parameters(A,B);
>> Y0 = guess;
>> [fn,Yn] = sg_min(Y0,'frcg','euclidean');
iter      grad          F(Y)          flops
0        2.887592e+03    9.671138e+02      30940
1        1.419011e+03    2.411349e+02      343392
2        6.199732e+02    7.601495e+01      718792
3        3.051636e+02    3.300591e+01     1106535
4        2.538995e+02    2.078422e+01     1518380
5        1.440247e+02    1.258174e+01     1923273
6        1.189063e+02    9.500450e+00     2332153
7        9.279907e+01    7.309071e+00     2743897
8        8.483878e+01    6.093323e+00     3161294
9        7.763100e+01    4.117233e+00     3609412
10       7.448803e+01    3.220241e+00     4026633
11       6.404668e+01    2.328540e+00     4439092
.....(many iterations later).....
169      2.196881e-06    4.010660e-07     67843218
170      2.309773e-06    4.010660e-07     68212219
```

171	2.435095e-06	4.010660e-07	68558476
172	1.763263e-06	4.010660e-07	68946596
173	2.081518e-06	4.010660e-07	69314166
174	1.251419e-06	4.010660e-07	69639992
175	1.474565e-06	4.010660e-07	70028565
176	8.949991e-07	4.010660e-07	70353929
177	8.506853e-07	4.010660e-07	70742988
178	7.154996e-07	4.010660e-07	71068090
179	8.003997e-07	4.010660e-07	71395085
180	3.164977e-07	4.010660e-07	71721113
181	3.701357e-07	4.010660e-07	72030590

Figure 9.6 shows the convergence curve for this run.

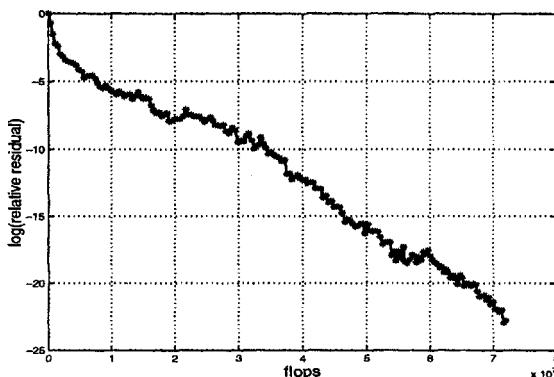


Figure 9.6: Simultaneous diagonalization problem

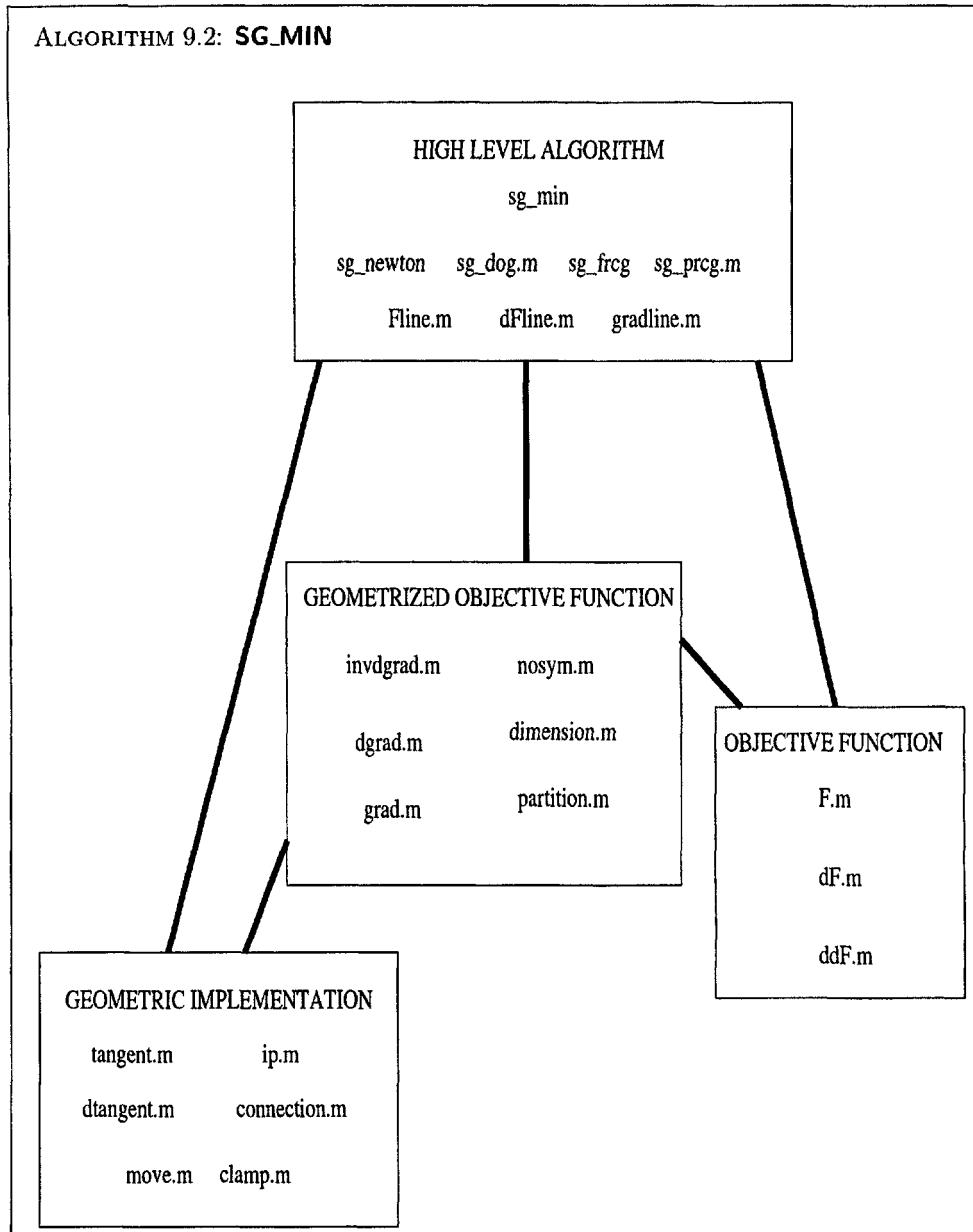
9.4.5 Modifying the Templates

This template is written to function as is, but can be tailored to specific problems, resulting in substantial improvements in performance. There are also functions which could be in-lined and computations which can be reused, of which we have not taken advantage for the sake of having a more readable code. Many components are the usual routines found in the literature (our line minimization routines are based on MATLAB's `fmin` and `fzero`, for example). Certainly it is possible for improvements to be made in the optimization routines themselves, and we welcome suggestions from optimization experts.

To make modifications as painless as possible, we present a section sketching the basic structure of the `sg_min` code. Readers who do not wish to modify any of the functions beyond `F.m`, `dF.m`, and `ddF.m` can skip this section.

9.4.5.1 Subroutine Dependencies

Algorithm 9.2 shows the dependencies of the various subroutines on each other. We have grouped the routines together based on four loosely defined roles:



- *Objective function.* These are the routines which implement the objective function and its derivatives. This group is the only group that a user interested only in the basic functionality need ever modify to adapt the template. Functions in this group are used by functions in the geometrized objective function and high-level algorithm groups.
- *Geometric implementation.* These routines implement the geometric features of the Stiefel manifold. This includes the projection of unconstrained vectors onto

the constraint (i.e., tangent) space (`tangent`), line movement (`move`), and the connection term used for covariant differentiation (`connection`). These routines are independent of the objective function group, but are essential to the geometrized objective function group.

- *Geometrized objective function.* This group uses the routines of the geometric implementation to project the differential and second differential of $F(Y)$ onto the constraint (i.e., tangent) surface producing the geometrically covariant gradient (`grad`) and Hessian (`dgrad`). Additionally, two routines for inverting the covariant Hessian (`invdgrad_CG` and `invdgrad_MINRES`) are located here. This group provides the raw tools out of which one builds implementations in the high-level algorithm group. Lastly, functions which detect (`partition`) and remove (`nosym`) block-diagonal orthogonal symmetries are found in the group.
- *High-level algorithm.* This group implements the constrained versions of various optimization algorithms. It is here that search directions are selected, line minimizations are performed (using `Fline`, `dFline`, and `gradline`), and convergence criteria are defined.

Lastly, every function reads from a global `SGParameters` structure whose fields contain information about the manifold and the computation. In our examples we have used a separate global `FParameters` to store information related to the computation of $F(Y)$. The fields of `SGParameters` are set in `sg_min`.

9.4.5.2 Under the Hood

`sg_min` is built from a couple of fairly elementary optimization codes which have been put through a geometrization that allows them to be situated on the Stiefel manifold. The basic elements of a geometrization are the rules for how to take inner products, how to turn unconstrained differentials into constrained gradients, how to differentiate gradient fields covariantly, and how to move about on the manifold.

The `sg_min` routine parses the arguments and sets the defaults. Finally, it calls `sg_newton`, `sg_dog`, `sg_frcg`, or `sg_prcg`.

The pseudocode for `sg_newton` and `sg_frcg` (as examples) are given in the boxes below.

These are fairly generic routines for optimization. At this level of description, one would not necessarily be able to tell them from unconstrained routines (see [170, 192, 368, 338]). What places them on the Stiefel manifold are the definitions of `grad`, `dgrad`, `ip` (the dot product), and `move`, which have been made in such a way that the constraints of the Stiefel manifold are respected. Likewise, `sg_dog` and `sg_prcg` have been similarly transported.

9.4.5.3 What to Modify

Here is a sketch of specific modifications a user may wish to make on the code.

- *Line minimization.* Currently, line minimizations are being performed by the MATLAB functions `fmin` and `fzero` through the wrappers `Fline`, `dFline`, and `gradline`. However, one may have better ways to line-minimize within an optimization routine or one may wish to supplement `fmin` and `fzero` for reasons of economy.

```

function [fn,Yn]= sg_newton(Y)
ginitial = grad(Y); g = ginitial;
f = F(Y); fold = 2*f;
while (||g|| > eps) & (||g||/||ginitial|| > gradtol) & (|fold/f-1 | > ftol)
    sdir = -g;
% steepest descent direction and line minimization on different scales
    fsa = minimize F(move(Y,sdir,sa)) for sa ∈ [-1, 1] · | $\frac{f}{\langle g, sdir \rangle}$ |;
    fsb = minimize F(move(Y,sdir,fb)) for sb ∈ [-1, 1] · | $\frac{\langle g, sdir \rangle}{\langle sdir, dgrad·sdir \rangle}$ |;
    ndir = -dgrad-1· g;
% Newton direction and line minimization
    fna = minimize F(move(Y,ndir,na)) for na ∈ [-1, 1] · | $\frac{f}{\langle g, ndir \rangle}$ |;
    fnb = minimize F(move(Y,ndir,nb)) for nb ∈ [-1, 1] · | $\frac{\langle g, ndir \rangle}{\langle ndir, dgrad·ndir \rangle}$ |;
% compare the best Newton function value with the best steepest
    if (fsa < fsb) st=sa; fst=fsa; else st=sb; fst=fsb; end
    if (fna < fnb) nt=na; fnt=fna; else nt=nb; fnt=fnb; end
    if (fst < fnt)
        dir = sdir; t = st; fnew = fst;
    else
        dir = ndir; t = nt; fnew = fnt;
    end
% move to the new point
    Y = move(Y,dir,t); fold= f; f = fnew;
    g=grad(Y);
end
fn = f;
Yn = Y;

```

```

function [fn,Yn]= sg_frcg(Y)
ginitial = grad(Y); g = ginitial;
f = F(Y); fold = 2*f;
while (||g|| > eps) & (||g||/||ginitial|| > gradtol) & (|fold/f-1 | > ftol)
    dir = -g;
% get a Hessian conjugate direction
    if (not first iteration)
        dir = dir -  $\frac{dir \cdot (dgrad \cdot dir_{old})}{dir_{old} \cdot (dgrad \cdot dir_{old})} * dir_{old}$ ;
    end
% line minimization
    fcga = minimize F(move(Y,dir,cga)) for cga ∈ [-1, 1] · | $\frac{f}{\langle g, dir \rangle}$ |;
    fcgb = minimize F(move(Y,dir,cgb)) for cgb ∈ [-1, 1] · | $\frac{\langle g, dir \rangle}{\langle dir, dgrad \cdot dir \rangle}$ |;
    if (fcga < fcgb) t=cga; fnew=fcga; else t=cgb; fnew=fcgb; end
% move to the new point
    [Y,dirold] = move(Y,dir,t); fold= f; f = fnew;
    g=grad(Y);
end
fn = f;
Yn = Y;

```

- *High-level algorithms.* The user may wish to use a minimization routine differing from or more sophisticated than the methods currently available, or one may have a different stopping criterion [17] that we did not anticipate. It is possible to adapt other flat optimization algorithms to new `sg_` algorithms without having to know too many details. Most unconstrained minimizations contain line searches ($y \rightarrow y + t*v$), Hessian applications ($H*v$), and inner products ($v'w$). To properly geometrize the algorithm, one has to do no more than replace these components with `move(Y,V,t)`, `dgrad(Y,V)`, and `ip(Y,V,W)`, respectively.
- *Hessian inversion.* We currently provide codes to invert the covariant Hessian in the functions `invdgrad_CG` (which uses a conjugate gradient method) and `invdgrad_MINRES` (which uses a MINRES algorithm). Any matrix inversion routine which can stably invert black box linear operators could be used in place of these, and no other modifications to the code would be required.
- *Workspaces.* To make the code more readable we have not implemented any workspace variables even though there are many computations which could be reused. For example, in the objective function group, for many of the sample problems, products of the form $A*Y$ or $Y*B$ or complicated matrix functions of Y such as $B(Y)$ in the Jordan example could be saved in a workspace. Some of the terms in the computation of `grad` and `dgrad` could likewise be saved (in fact, our current implementation essentially recomputes the gradient every time the covariant Hessian is applied).
- *In-lining.* Although we have separated the `tangent` and `dtangent` functions from the `grad` and `dgrad` functions, one often finds that when these functions are in-lined, some terms cancel out and therefore do not need to be computed. One might also consider in-lining the `move` function when performing line minimizations so that one could reuse data to make function evaluations and point updates faster.
- *Sphere or $\mathcal{O}(n)$ geometry.* Some problems are always set on the unit sphere or on $\mathcal{O}(n)$, both of which have simpler geometries than the Stiefel manifold. Geodesics on both are easier to compute, as are tangent projections. To take advantage of this, one should modify the geometric implementation group.
- *Globals.* In order to have adaptable and readable code, we chose to use global structures, `SGParameters` and `FParameters`, to hold data which could be put in the argument lists of all the functions. The user may wish to modify his/her code so that this data is passed explicitly to each function as individual arguments.
- *Preconditioning.* The Hessian solve by conjugate gradient routine, `invdgrad`, does not have any preconditioning step. Conjugate gradient can perform very poorly without preconditioning, and the preconditioning of black box linear operators like `dgrad` is a very difficult problem. The user might try to find a good preconditioner for `dgrad`.

9.4.6 Geometric Technicalities

This section is a brief explanation of the geometric concepts used to produce the methods employed by `sg_min` and its subroutines. This information is mostly a recapitulation of material readily available in the differential geometry literature.

9.4.6.1 Manifolds

A manifold, M , is a collection of points which have a differential structure. In plain terms, this means that one is able to take derivatives of some reasonable class of functions, the C^∞ functions, defined on M . What this class of differentiable functions is can be somewhat arbitrary, though some technical consistency conditions must be satisfied. We will not discuss those conditions in this section.

We consider the manifold $\text{Stief}(n, k)$ (for purposes of clearer explanation, we restrict our discussion to the real version of the Stiefel manifold) of points which are written as $n \times k$ matrices satisfying the constraint $Y^*Y = I$. We will select for our set of C^∞ functions those real-valued functions which are restrictions to $\text{Stief}(n, k)$ of functions of nk variables which are $C^\infty(\mathcal{R}^{n \times k})$ about $\text{Stief}(n, k)$ in the $\mathcal{R}^{n \times k}$ sense. It should not be difficult to convince oneself that the set of such functions must satisfy any technical consistency condition one could hope for.

Additionally, $M = \text{Stief}(n, k)$ is a smooth manifold. This means that about every point of $\text{Stief}(n, k)$ we can construct a local coordinate system which is C^∞ . For example, consider the point

$$Y = \begin{bmatrix} I \\ 0 \end{bmatrix}$$

and a point in the neighborhood of Y ,

$$\tilde{Y} = \begin{bmatrix} A \\ B \end{bmatrix}.$$

For small B , we can solve for A in terms of the components of B and the components of an arbitrary small antisymmetric matrix Δ by solving $S^*S = I - B^*B$ for symmetric S and letting $A = e^\Delta S$. This can always be done smoothly and in a locally 1-to-1 fashion for small enough B and Δ . Since the components of B are all smooth functions (being restrictions of the coordinate functions of the ambient $\mathcal{R}^{n \times k}$ space), and since the solution for A is $C^\infty(\mathcal{R}^{(n-k) \times k} \oplus \mathcal{R}^{k \times k})$ for small enough B and Δ , we have shown that any point of $\text{Stief}(n, k)$ can be expressed smoothly as a function of $(n-k)k + k(k-1)/2 = nk - k(k+1)/2$ variables. The only difference between $\begin{bmatrix} I \\ 0 \end{bmatrix}$ is a Euclidean rigid motion; therefore, this statement holds for all points in $\text{Stief}(n, k)$.

Summarizing, a manifold is a set of points with a set of differentiable functions defined on it as well as a sense of local coordinatization in which the dependent coordinates can be represented as smooth functions of the independent coordinates. Specifically, we see that there are always $nk - k(k+1)/2$ independent coordinates required to coordinatize neighborhoods of points of $\text{Stief}(n, k)$. This number is called the *dimension* of the manifold, and it should be the same for every point of a manifold (if not, then the manifold is either disconnected or not actually smooth).

9.4.6.2 The Difference Between a Tangent and a Differential

Given a smooth point x on any manifold and a smooth path $p(s)$ in the manifold such that $p(0) = x$, one can construct a differential operator on the C^∞ functions defined near x by the rule

$$D_p f = \frac{d}{ds} f(p(s))|_{s=0}.$$

These differential operators at x form a finite-dimensional vector space, and the dimension of this vector space is equal to the dimension of the manifold. This vector space is called the *tangent space* of M at x , and is often denoted $T_x(M)$.

For the Stiefel manifold, and, generally, all smooth constraint manifolds, the tangent space at any point is easy to characterize. The constraint equation $Y^*Y = I$ can be thought of as $k(k+1)/2$ independent functions on $\text{Stief}(n, k)$ which must all have constant value. If H is a tangent vector of $\text{Stief}(n, k)$ at Y , it must then be that

$$H^*Y + Y^*H = 0,$$

which is obtained by taking $\frac{d}{dt}Y(t)^*Y(t) = 0|_{t=0}$ (where $\dot{Y}(0) = H$). In a constraint manifold, the tangents at Y can equivalently be identified with those infinitesimal displacements of Y which preserve the constraint equations to first order.

For H to be a tangent, its components must satisfy $k(k+1)/2$ independent equations. These equations are all linear in the components of H , and thus the set of all tangents is an $(nk - k(k+1)/2)$ -dimensional subspace of the vector space $\mathcal{R}^{n \times k}$.

A differential operator D_H may be associated with an $n \times k$ matrix, H , given by

$$D_H f = \frac{d}{dt}f(Y + tH)|_{t=0}.$$

D_H is the directional derivative operator in the H direction. One may observe that for H to be a tangent vector, the tangency condition is equivalent to $D_H(Y^*Y) = 0$.

While tangents are $n \times k$ matrices and, therefore, have associated differential operators, *differentials* are something else. Given a C^∞ function f and a point Y , one can consider the equation $D_H f$ for $H \in \mathcal{R}^{n \times k}$ (H is not necessarily tangent). This expression is linear in H and takes on some real value. It is thus possible to represent it as a linear function on the vector space $\mathcal{R}^{n \times k}$,

$$D_H f = \text{tr}(H^*Z),$$

for some appropriate $n \times k$ matrix Z whose values depend on the first-order behavior of f near Y . We identify this Z matrix with df , the differential of f at Y . This is the same differential which is computed by the `dF` functions in the sample problems.

For any constraint manifold the differential of a smooth function f can be computed without having to know anything about the manifold itself. One can simply use the differentials as computed in the ambient space (the unconstrained $\mathcal{R}^{n \times k}$ derivatives in our case). If one then restricts one's differential operators to be only those in tangent directions, then one can still use the unconstrained df in $\text{tr}(H^*df)$ to compute $D_H f$ for $H \in T_Y(\text{Stief}(n, k))$. This is why it requires no geometric knowledge to produce the `dF` functions.

9.4.6.3 Inner Products, Gradients, and Differentials

In flat spaces, we often identify the differential of a function with its gradient. However, when dealing with a more general setting, one can run into problems making sense out of such a definition.

For example, the gradient is a vector, and it should be possible to think of vectors as infinitesimal displacements of points. In $\text{Stief}(n, k)$, any infinitesimal displacement δY must satisfy $\delta Y^*Y + Y^*\delta Y = 0$. Thus, df may not always be a vector, since it does not

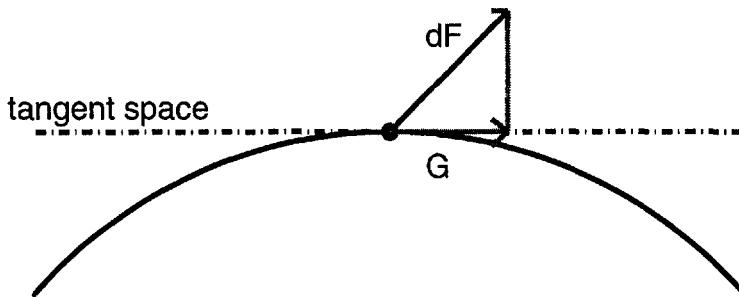


Figure 9.7: The unconstrained differential of $F(Y)$ can be projected to the tangent space to obtain the covariant gradient, G , of F .

necessarily satisfy this equation. A gradient should be an infinitesimal displacement that points in the direction of the displacement which will give the greatest increase in f .

If the tangent space has an inner product, though, one can find a useful way to identify the df uniquely with a tangent vector. Let $ip(H_1, H_2)$ be a symmetric nondegenerate bilinear form on the tangent space of $\text{Stief}(n, k)$ at Y . Then one can define the gradient, G , implicitly by

$$ip(G, H) = \text{tr}(H^* df) = D_H f.$$

Since ip is a nondegenerate form, this is sufficient to define G . The function `tangent` carries out this projection from differentials to tangents (shown in Figure 9.7). This operation is performed by `grad` to produce the gradient of the objective function.

9.4.6.4 Getting Around $\text{Stief}(n, k)$

We've now laid the groundwork for a meaningful definition of the gradient in the setting of a constraint manifold. At this point, one could run off and try to do a steepest descent search to maximize one's objective functions. Trouble will arise, however, when one discovers that there is no sensible way to combine a point and a displacement to produce a new point because, for finite s , $Y + sH$ violates the constraint equations and thus does not give a point on the manifold.

For any manifold, one updates Y by solving a set of differential *equations of motion* of the form

$$\begin{aligned} \frac{d}{dt} Y &= H, \\ \frac{d}{dt} H &= -\Gamma(H, H). \end{aligned}$$

The Γ term is crafted to ensure that H remains a tangent vector for all times t , thus keeping the path on the manifold. It is called the *connection*. (The connection term also depends on Y .)

To see how these equations could be satisfied, we take the infinitesimal constraint equation for H ,

$$H^* Y + Y^* H = 0,$$

and differentiate it with respect to t , to get

$$\Gamma(H, H)^*Y + Y^*\Gamma(H, H) = 2H^*H.$$

This can be satisfied generally by $\Gamma(H, H) = Y(H^*H) + T(H, H)$, where $T(H, H)$ is some arbitrary tangent vector.

In the next subsection, we will have reason to consider $\Gamma(H_1, H_2)$ for $H_1 \neq H_2$. For technical reasons, one usually requests that

$$\Gamma(H_1, H_2) = \Gamma(H_2, H_1)$$

(this is called the *torsion-free* property) and that

$$ip(\Gamma(H_1, H_3), H_2) + ip(H_1, \Gamma(H_2, H_3)) = 0$$

(this is called the *metric compatibility* property). These two properties uniquely determine the $T(H_1, H_2)$ term, and thereby uniquely specify the connection. On any manifold, the unique connection with these properties is called the *Levi-Civita* connection.

The connection for the Stiefel manifold using two different inner products (the Euclidean and the canonical) can be found in the work by Edelman, Arias, and Smith (see [151, 155, 416]). The function `connection` computes $\Gamma(H_1, H_2)$ in the template software.

Usually the solution of the equations of motions on a manifold are very difficult to carry out. For the Stiefel manifold, analytic solutions exist and can be found in the aforementioned literature, though we have found very little performance degradation between moving along paths via the equations of motion and simply performing some orthogonalizing factorization on $Y + sH$, as long as the displacements are small. The `move` function supports multiple methods of geodesic motion, depending on the degree of approximation desired.

9.4.6.5 Covariant Differentiation

With the notion of a gradient and a notion of movement that respects the constraints of the manifold, one might wish to begin with an optimization routine of some sort. A steepest descent could be implemented from these alone. However, in order to carry out sophisticated optimizations, one usually wants some sort of second derivative information about the function.

In particular, one might wish to know by how much one can expect the gradient to change if one moves from Y to $Y + \epsilon H$. This can actually be a difficult question to answer on a manifold. Technically, the gradient at Y is a member of $T_Y(\text{Stief}(n, k))$, while the gradient at $Y + \epsilon H$ is a member of $T_{Y+\epsilon H}(\text{Stief}(n, k))$. While taking their difference would work fine in a flat space (see Figure 9.8), if this were done on a curved space, it could give a vector which is not a member of the tangent space of either point (see Figure 9.9).

A more sophisticated means of taking this difference is to first move the gradient at $Y + \epsilon H$ to Y in some manner which translates it in a parallel fashion from $Y + \epsilon H$ to Y , and then compare the two gradients within the same tangent space. One can check that for $V \in T_{Y+\epsilon H}(\text{Stief}(n, k))$ the rule

$$V \rightarrow V + \epsilon\Gamma(V, H),$$

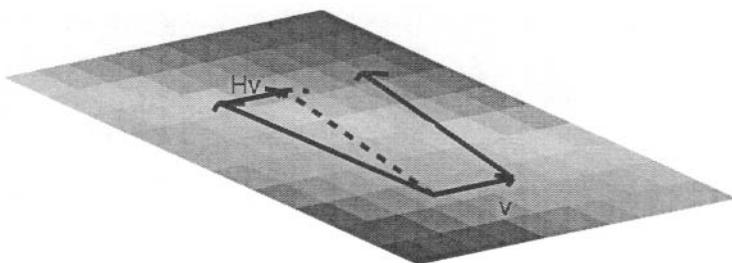


Figure 9.8: In a flat space, comparing vectors at nearby points is not problematic since all vectors lie in the same tangent space.

where $\Gamma(V, H)$ is the Levi–Civita connection, takes V to an element of $T_Y(\text{Stief}(n, k))$ and preserves inner product information (to first order in ϵ). This is the standard rule for parallel transport which can be found in the usual literature ([80, 459, 273, 222], and others).

Using this rule to compare nearby vectors to each other, one then has the following rule for taking derivatives of vector fields:

$$D_H G = \frac{d}{ds} G(Y + sH)|_{s=0} + \Gamma(G, H),$$

where G is any vector field (but we are only interested in derivatives of the gradient field). This is the function implemented by `dgrad` in the software.

In an unconstrained minimization, the second derivative of the gradient $g = \nabla f$ along a vector \vec{h} is the Hessian $[\frac{\partial^2 f}{\partial x_i \partial x_j}]$ times \vec{h} . Covariantly, we then have the analogy,

$$\left[\frac{\partial^2 f}{\partial x_i \partial x_j} \right] \vec{h} = (\vec{h} \cdot \nabla) g \sim D_H G.$$

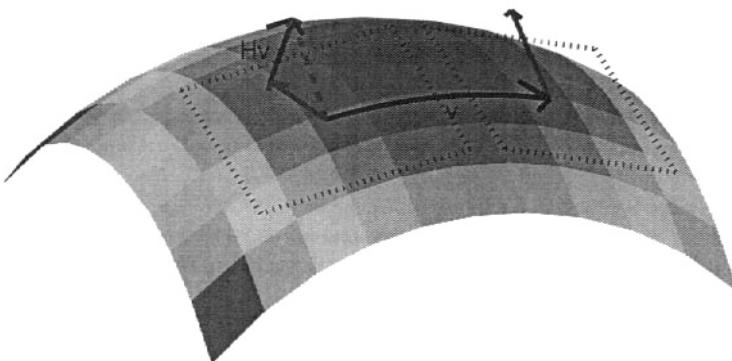


Figure 9.9: In a curved manifold, comparing vectors at nearby points can result in vectors which do not lie in the tangent space.

9.4.6.6 Inverting the Covariant Hessian (Technical Considerations)

Since the tangent space of $\text{Stief}(n, k)$ is a subspace of $\mathcal{R}^{n \times k}$, the covariant Hessian must be inverted stably on this subspace. This requires any algorithms designed to solve $D_H G = V$ for H to really be pseudoinverters in a least squares or some other sense.

A second consideration is that many useful functions $f(Y)$ have the property that $f(YQ) = f(Y)$ for all block-diagonal orthogonal Q (i.e.,

$$Q = \begin{bmatrix} Q_1 & 0 & \cdots & 0 \\ 0 & Q_2 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots \\ 0 & 0 & \cdots & Q_p \end{bmatrix},$$

where the Q_i are orthogonal matrices). In this case, all tangent vectors of the form

$$H = Y \begin{bmatrix} H_1 & 0 & \cdots & 0 \\ 0 & H_2 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots \\ 0 & 0 & \cdots & H_p \end{bmatrix},$$

where the H_i are antisymmetric, have the property $D_H f = 0$. These extra symmetry vectors are then null vectors of the linear system $D_H G = V$. Because of these null directions, the effective dimension of the tangent space is reduced by the dimension of this null space (this is the dimension returned by the `dimension` function).

Thus, we see that in order to invert the covariant Hessian, we must take care to use a stable inversion scheme which will project out components of H which do not satisfy the infinitesimal constraint equation and those which are in the direction of the additional symmetries of f . The `invdgrad` function carries out a stable inversion of the covariant Hessian by a conjugate gradient routine, with the `dgrad` function calling the function `nosym` to project out any extra symmetry components.

Chapter 10

Common Issues

10.1 Sparse Matrix Storage Formats

J. Dongarra

The efficiency of most of the iterative methods considered in this book is determined primarily by the performance of the matrix-vector product and therefore on the storage scheme used for the matrix. Often, the storage scheme used arises naturally from the specific application problem.

In this section we will review some of the more popular sparse matrix formats that have been used in numerical software packages such as ITPACK [263], NSPCG [345], and SPARSPAK [191], some of which have more recently been adopted as part of a new software standard by the BLAS Technical Forum (see ETHOME for further information). In §10.2.2, we demonstrate how the matrix-vector product is formulated using two of the sparse matrix formats.

If the coefficient matrix A is sparse, large scale eigenvalue problems can be most efficiently solved if the zero elements of A are neither manipulated nor stored. Sparse storage schemes allocate contiguous storage in memory for the nonzero elements of the matrix, and perhaps a limited number of zeros. This, of course, requires a scheme for knowing where the elements fit into the full matrix.

There are many methods for storing the data (see, for instance, Saad [386] and Eijkhout [156]). Here we will discuss compressed row and column storage, block compressed row storage, diagonal storage, jagged diagonal storage, and skyline storage.

10.1.1 Compressed Row Storage

The compressed row and column (in the next section) storage formats are the most general: they make absolutely no assumptions about the sparsity structure of the matrix, and they don't store any unnecessary elements. On the other hand, they are not very efficient, needing an indirect addressing step for every single scalar operation in a matrix-vector product or preconditioner solve.

The compressed row storage (CRS) format puts the subsequent nonzeros of the matrix rows in contiguous memory locations. Assuming we have a nonsymmetric sparse matrix A , we create three vectors: one for floating point numbers (`val`) and the other

two for integers (`col_ind`, `row_ptr`). The `val` vector stores the values of the nonzero elements of the matrix A as they are traversed in a row-wise fashion. The `col_ind` vector stores the column indexes of the elements in the `val` vector. That is, if $\text{val}(k) = a_{i,j}$, then $\text{col_ind}(k) = j$. The `row_ptr` vector stores the locations in the `val` vector that start a row; that is, if $\text{val}(k) = a_{i,j}$, then $\text{row_ptr}(i) \leq k < \text{row_ptr}(i+1)$. By convention, we define $\text{row_ptr}(n+1) = nnz + 1$, where nnz is the number of nonzeros in the matrix A . The storage savings for this approach is significant. Instead of storing n^2 elements, we need only $2nnz + n + 1$ storage locations.

As an example, consider the nonsymmetric matrix A defined by

$$A = \begin{bmatrix} 10 & 0 & 0 & 0 & -2 & 0 \\ 3 & 9 & 0 & 0 & 0 & 3 \\ 0 & 7 & 8 & 7 & 0 & 0 \\ 3 & 0 & 8 & 7 & 5 & 0 \\ 0 & 8 & 0 & 9 & 9 & 13 \\ 0 & 4 & 0 & 0 & 2 & -1 \end{bmatrix}. \quad (10.1)$$

The CRS format for this matrix is then specified by the arrays `{val, col_ind, row_ptr}` given below:

<code>val</code>	10	-2	3	9	3	7	8	7	3	...	9	13	4	2	-1
<code>col_ind</code>	1	5	1	2	6	2	3	4	1	...	5	6	2	5	6
<code>row_ptr</code>															
	1	3	6	9	13	17	20								

If the matrix A is symmetric, we need only store the upper (or lower) triangular portion of the matrix. The tradeoff is a more complicated algorithm with a somewhat different pattern of data access.

10.1.2 Compressed Column Storage

Analogous to CRS, there is compressed column storage (CCS), which is also called the *Harwell-Boeing* sparse matrix format [139]. The CCS format is identical to the CRS format except that the columns of A are stored (traversed) instead of the rows. In other words, the CCS format is the CRS format for A^T .

The CCS format is specified by the 3 arrays `{val, row_ind, col_ptr}`, where `row_ind` stores the row indices of each nonzero, and `col_ptr` stores the index of the elements in `val` which start a column of A . The CCS format for the matrix A in (10.1) is given by

<code>val</code>	10	3	3	9	7	8	4	8	8	...	9	2	3	13	-1
<code>row_ind</code>	1	2	4	2	3	5	6	3	4	...	5	6	2	5	6
<code>col_ptr</code>															
	1	4	8	10	13	17	20								

10.1.3 Block Compressed Row Storage

If the sparse matrix A is composed of square dense blocks of nonzeros in some regular pattern, we can modify the CRS (or CCS) format to exploit such block patterns. Block matrices typically arise from the discretization of partial differential equations in which there are several *degrees of freedom* associated with a grid point. We then partition

the matrix in small blocks with a size equal to the number of degrees of freedom and treat each block as a dense matrix, even though it may have some zeros.

If n_b is the dimension of each block and $nnzb$ is the number of nonzero blocks in the $n \times n$ matrix A , then the total storage needed is $nnz = nnzb \times n_b^2$. The block dimension n_d of A is then defined by $n_d = n/n_b$.

Similar to the CRS format, we require three arrays for the BCRS format: a rectangular array for floating point numbers (`val(1 : nnzb, 1 : nb, 1 : nb)`) which stores the nonzero blocks in (block) row-wise fashion, an integer array (`col_ind(1 : nnzb)`) which stores the actual column indices in the original matrix A of the (1,1) elements of the nonzero blocks, and a pointer array (`row_blk(1 : nd + 1)`) whose entries point to the beginning of each block row in `val(:, :, :)` and `col_ind(:)`. The savings in storage locations and reduction in time spent doing indirect addressing for block compressed row storage (BCRS) over CRS can be significant for matrices with a large n_b .

10.1.4 Compressed Diagonal Storage

If the matrix A is banded with bandwidth that is fairly constant from row to row, then it is worthwhile to take advantage of this structure in the storage scheme by storing subdiagonals of the matrix in consecutive locations. Not only can we eliminate the vector identifying the column and row, but we can pack the nonzero elements in such a way as to make the matrix-vector product more efficient. This storage scheme is particularly useful if the matrix arises from a finite element or finite difference discretization on a tensor product grid.

We say that the matrix $A = (a_{i,j})$ is *banded* if there are nonnegative constants p, q , called the left and right *halfbandwidth*, such that $a_{i,j} \neq 0$ only if $i-p \leq j \leq i+q$. In this case, we can allocate for the matrix A an array `val(1:n, -p:q)`. The declaration with reversed dimensions (`-p:q, n`) corresponds to the LINPACK band format [132], which, unlike compressed diagonal storage (CDS), does not allow for an efficiently vectorizable matrix-vector multiplication if $p+q$ is small.

Usually, band formats involve storing some zeros. The CDS format may even contain some array elements that do not correspond to matrix elements at all. Consider the nonsymmetric matrix A defined by

$$A = \begin{bmatrix} 10 & -3 & 0 & 0 & 0 & 0 \\ 3 & 9 & 6 & 0 & 0 & 0 \\ 0 & 7 & 8 & 7 & 0 & 0 \\ 0 & 0 & 8 & 7 & 5 & 0 \\ 0 & 0 & 0 & 9 & 9 & 13 \\ 0 & 0 & 0 & 0 & 2 & -1 \end{bmatrix}. \quad (10.2)$$

Using the CDS format, we store this matrix A in an array of dimension $(6, -1:1)$ using the mapping

$$\text{val}(i, j) = a_{i, i+j}. \quad (10.3)$$

Hence, the rows of the `val(:, :)` array are

<code>val(:, -1)</code>	0	3	7	8	9	2
<code>val(:, 0)</code>	10	9	8	7	9	-1
<code>val(:, +1)</code>	-3	6	7	5	13	0

Notice the two zeros corresponding to nonexistent matrix elements.

10.1.5 Jagged Diagonal Storage

The jagged diagonal storage (JDS) format can be useful for the implementation of iterative methods on parallel and vector processors (see Saad [385]). Like the CDS format, it gives a vector length of essentially the same size as the matrix. It is more space-efficient than CDS at the cost of a gather/scatter operation.

A simplified form of JDS, called ITPACK storage or Purdue storage, can be described as follows. For the following nonsymmetric matrix, all elements are shifted left:

$$\left[\begin{array}{cccccc} 10 & -3 & 0 & 1 & 0 & 0 \\ 0 & 9 & 6 & 0 & -2 & 0 \\ 3 & 0 & 8 & 7 & 0 & 0 \\ 0 & 6 & 0 & 7 & 5 & 4 \\ 0 & 0 & 0 & 0 & 9 & 13 \\ 0 & 0 & 0 & 0 & 5 & -1 \end{array} \right] \rightarrow \left[\begin{array}{cccccc} 10 & -3 & 1 & & & \\ 9 & 6 & -2 & & & \\ 3 & 8 & 7 & & & \\ 6 & 7 & 5 & 4 & & \\ 9 & 13 & & & & \\ 5 & -1 & & & & \end{array} \right]$$

after which the columns are stored consecutively. All rows are padded with zeros on the right to give them equal length. Corresponding to the array of matrix elements $\text{val}(:, :)$, an array of column indices, $\text{col_ind}(:, :)$, is also stored:

$\text{val}(:, 1)$	10	9	3	6	9	5
$\text{val}(:, 2)$	-3	6	8	7	13	-1
$\text{val}(:, 3)$	1	-2	7	5	0	0
$\text{val}(:, 4)$	0	0	0	4	0	0

$\text{col_ind}(:, 1)$	1	2	1	2	5	5
$\text{col_ind}(:, 2)$	2	3	3	4	6	6
$\text{col_ind}(:, 3)$	4	5	4	5	0	0
$\text{col_ind}(:, 4)$	0	0	0	6	0	0

It is clear that the padding zeros in this structure may be a disadvantage, especially if the bandwidth of the matrix varies strongly. Therefore, in the CRS format, we reorder the rows of the matrix decreasingly according to the number of nonzeros per row. The compressed and permuted diagonals are then stored in a linear array. The new data structure is called *jagged diagonals*.

Specifically, we store the (dense) vector of all the first elements in val , col_ind from each row, together with an integer vector containing the column indices of the corresponding elements. This is followed by the second jagged diagonal consisting of the elements in the second positions from the left. We continue to construct more and more of these jagged diagonals (whose length decreases).

The number of jagged diagonals is equal to the number of nonzeros in the first row, i.e., the largest number of nonzeros in any row of A . The data structure to represent the $n \times n$ matrix A therefore consists of a permutation array ($\text{perm}(1:n)$), which reorders the rows, a floating point array ($\text{jdiag}(:)$) containing the jagged diagonals in succession, an integer array ($\text{col_ind}(:)$) containing the corresponding column indices, and finally a pointer array ($\text{jd_ptr}(:)$) whose elements point to the beginning of each jagged diagonal. The advantages of JDS for matrix multiplications were discussed by Saad in [385].

The JDS format for the above matrix A in using the linear arrays $\{\text{perm}, \text{jdiag}, \text{col_ind}, \text{jd_ptr}\}$ is given below (jagged diagonals are separated by semicolons):

jdiag	1	3	7	8	10	2;	9	9	8	... -1;	9	6	7	5;	13
col_ind	1	1	2	3	1	5;	4	2	3	... 6;	5	3	4	5;	6
perm	5	2	3	4	1	6	jd_ptr	1	7	13	17	.			

10.1.6 Skyline Storage

The final storage scheme we consider is for skyline matrices, which are also called variable band or profile matrices (see Duff, Erisman, and Reid [138]). It is mostly of importance in direct solution methods, but it can be used for handling the diagonal blocks in block matrix factorization methods. A major advantage of solving linear systems having skyline coefficient matrices is that when pivoting is not necessary, the skyline structure is preserved during Gaussian elimination. If the matrix is symmetric, we only store its lower triangular part. A straightforward approach in storing the elements of a skyline matrix is to place all the rows (in order) into a floating point array (`val(:)`), and then keep an integer array (`row_ptr(:)`) whose elements point to the beginning of each row. The column indices of the nonzeros stored in `val(:)` are easily derived and are not stored.

For a nonsymmetric skyline matrix such as the one illustrated in Figure 10.1, we store the lower triangular elements in skyline storage (SKS) format and store the upper triangular elements in a column-oriented SKS format (transpose stored in row-wise SKS format). These two separated *substructures* can be linked in a variety of ways. One approach, discussed by Saad in [386], is to store each row of the lower triangular part and each column of the upper triangular part contiguously into the floating point array (`val(:)`). An additional pointer is then needed to determine where the diagonal elements, which separate the lower triangular elements from the upper triangular elements, are located.

+	x	x													
x	+	x	x												
x	x	+	x				x								
x	x	+	x	x	x	x	x								
			x	+	x	x	x	x							
x	x	x	+	x	x										
x	x	x	+	x	x		x								
			x	+	x	x	x	x							
x	x	x	x	+	x	x	x	x							
x	x	x	x	+	x	x	x	x							
			x	+	x	x	x	x							
x	x	x	x	x	+	x	x	x							
x	x	x	x	x	x	+	x	x	x						
			x	x	x	+	x	x	x	x					
x	x	x	x	x	x	x	+	x	x	x					
			x	x	x	x	x	+	x	x	x				
x	x	x	x	x	x	x	x	x	+	x	x				
			x	x	x	x	x	x	x	+	x	x			
x	x	x	x	x	x	x	x	x	x	x	+	x	x		
			x	x	x	x	x	x	x	x	x	+	x	x	
x	x	x	x	x	x	x	x	x	x	x	x	x	+	x	
			x	x	x	x	x	x	x	x	x	x	x	+	

Figure 10.1: Profile of a nonsymmetric skyline or variable-band matrix.

10.2 Matrix-Vector and Matrix-Matrix Multiplications

J. Dongarra, P. Koev, and X. Li

10.2.1 BLAS

For the past 15 years or so, there has been a great deal of activity in the area of algorithms and software for solving linear algebra problems. The linear algebra community has long recognized the need for help in developing algorithms into software libraries, and several years ago, as a community effort, put together a de facto standard for identifying basic operations required in linear algebra algorithms and software. The hope was that the routines making up this standard, known collectively as the Basic Linear Algebra Subprograms (BLAS), would be efficiently implemented on advanced-architecture computers by many manufacturers, making it possible to reap the portability benefits of having them efficiently implemented on a wide range of machines. This goal has been largely realized.

The key insight of our approach to designing linear algebra algorithms for advanced architecture computers is that the frequency with which data are moved between different levels of the memory hierarchy must be minimized in order to attain high performance. Thus, our main algorithmic approach for exploiting both vectorization and parallelism in our implementations is the use of block-partitioned algorithms, particularly in conjunction with highly tuned kernels for performing matrix-vector and matrix-matrix operations (the Level 2 and 3 BLAS). In general, the use of block-partitioned algorithms requires data to be moved as blocks, rather than as vectors or scalars, so that although the total amount of data moved is unchanged, the latency (or startup cost) associated with the movement is greatly reduced because fewer messages are needed to move the data.

A second key idea is that the performance of an algorithm can be tuned by a user by varying the parameters that specify the data layout. On shared memory machines, this is controlled by the block size, while on distributed memory machines it is controlled by the block size and the configuration of the logical process mesh.

The question arises, "How can we achieve sufficient control over these various factors that effect performance to obtain the levels of performance that machines can offer?" The answer is through use of the BLAS.

There are now three levels of BLAS:

Level 1 BLAS [288]: for vector operations, such as $y \leftarrow \alpha x + y$;

Level 2 BLAS [133]: for matrix-vector operations, such as $y \leftarrow \alpha Ax + \beta y$;

Level 3 BLAS [134]: for matrix-matrix operations, such as $C \leftarrow \alpha AB + \beta C$.

Here, A , B , and C are matrices; x and y are vectors; and α and β are scalars.

The Level 1 BLAS are used in packages like LAPACK, for convenience rather than for performance: they perform an insignificant fraction of the computation, and they cannot achieve high efficiency on most modern supercomputers.

The Level 2 BLAS can achieve near-peak performance on many vector processors, such as a single processor of a CRAY X-MP or Y-MP, or a Convex C-2 machine. However, on other vector processors such as a CRAY-2 or an IBM 3090 VF, the performance

of the Level 2 BLAS is limited by the rate of data movement between different levels of memory.

The Level 3 BLAS overcome this limitation. This third level of BLAS performs $O(n^3)$ floating point operations on $O(n^2)$ data, whereas the Level 2 BLAS perform only $O(n^2)$ operations on $O(n^2)$ data. The Level 3 BLAS also allow us to exploit parallelism in a way that is transparent to the software that calls them. While the Level 2 BLAS offer some scope for exploiting parallelism, greater scope is provided by the Level 3 BLAS.

To a great extent, the user community embraced the BLAS, not only for performance reasons, but also because developing software around a core of common routines like the BLAS is good software engineering practice. Highly efficient machine-specific implementations of the BLAS are available for most modern high-performance computers. The BLAS have enabled software to achieve high performance with portable code.

In the spirit of the earlier BLAS meetings and the standardization efforts of the Message-Passing Interface (MPI) and high-performance Fortran (HPF) forums, a technical forum was established to consider expanding the BLAS in the light of modern software, language, and hardware developments. The BLAS Technical Forum meetings began with a workshop in November 1995 at the University of Tennessee. Meetings were hosted by universities and software and hardware vendors.

Various working groups were established to consider issues such as overall functionality, language interfaces, sparse BLAS, distributed memory dense BLAS, extended and mixed precision BLAS, interval BLAS, and extensions to the existing BLAS. The rules of the forum were adopted from those used for the MPI and HPF forums.

A major aim of the standards defined in this document are to enable linear algebra libraries (both public domain and commercial) to interoperate efficiently, reliably, and easily. We believe that hardware and software vendors, higher level library writers, and application programmers all benefit from the efforts of this forum and are the intended end users of these standards.

Further information about the BLAS and BLAS Technical Forum can be obtained on the book's homepage, ETHOME.

10.2.2 Sparse BLAS

In the spirit of the dense BLAS, work is underway in the BLAS Technical Forum to establish a standard for the sparse BLAS. The sparse BLAS interface addresses computational routines for unstructured sparse matrices. Sparse BLAS also contains the three levels of operations as in the dense case. However, only a small subset of the dense BLAS is specified:

Level 1: sparse dot product, vector update, and gather/scatter;

Level 2: sparse matrix-vector multiply and triangular solve;

Level 3: sparse matrix-dense matrix multiply and triangular solve with multiple right-hand sides.

These are the basic operations used in most of the iterative algorithms for solving sparse linear equations and eigensystems. The Level 2 and 3 sparse BLAS interfaces support nine sparse matrix storage formats. The nine formats are divided into two

categories: the *point entry* formats such as compressed row and the *block entry* formats such as block compressed row. In the block entry formats, the sparsity structure is represented as a series of small dense blocks. Note that the nine formats include some surveyed in §10.1, except JDS and SKS.

The interface design of the sparse BLAS is fundamentally different from the dense BLAS. Since there is no single “best” storage format for any sparse matrix operation, the sparse matrix arguments to the Level 2 and 3 sparse BLAS do not use one particular storage format. Instead, a *generic interface* is defined for these routines, in which a sparse matrix argument is a *handle* (integer) representing the matrix. Before calling a sparse BLAS routine, one needs to call a creation routine to create the sparse matrix handle from one of the nine formats. After calling the sparse BLAS routine, one needs to call a cleanup routine to release the resources associated with the matrix handle. This handle-based approach allows one to use the sparse BLAS independently of the sparse matrix storage. The internal representation is implementation dependent and can be chosen for best performance.

Since matrix-vector products often account for most of the time spent in iterative methods, several research efforts have tried to optimize their performance [438, 439, 240, 239]. In matrix-vector multiplication, each matrix entry participates in only one operation, but each vector entry may participate in more than one operation. A primary goal of optimization is to reduce the amount of movement of the source vector between different levels of memory. The optimization techniques include matrix reordering, cache blocking, and register blocking. A recently developed toolbox, called SPARSITY, embraces all these techniques [240, 239]. Depending on the matrix structure, the authors have observed up to threefold speedup even on uniprocessors. SPARSITY also contains mechanisms to automatically choose the best block sizes based on matrix and machine characteristics.

In many of the iterative methods discussed earlier, both the product of a matrix and that of its transpose times a vector are needed; that is, given an input vector x we want to compute products

$$y = Ax \quad \text{and} \quad y = A^T x.$$

In the following two subsections, we will present algorithms for the storage format from §10.1, CRS.

10.2.2.1 CRS Matrix-Vector Product

The matrix-vector product $y = Ax$ using CRS format can be expressed in the usual way:

$$y_i = \sum_j a_{i,j}x_j,$$

since this traverses the rows of the matrix A . For an $n \times n$ matrix A , the matrix-vector multiplication is given by

```

for i = 1, n
    y(i) = 0
    for j = row_ptr(i), row_ptr(i+1) - 1
        y(i) = y(i) + val(j) * x(col_ind(j))
    end;
end;

```

Since this method only multiplies nonzero matrix entries, the operation count is 2 times the number of nonzero elements in A , which is a significant savings over the dense operation requirement of $2n^2$.

For the transpose product $y = A^T x$ we cannot use the equation

$$y_i = \sum_j (A^T)_{i,j} x_j = \sum_j a_{j,i} x_j,$$

since this implies traversing columns of the matrix, an extremely inefficient operation for matrices stored in CRS format. Hence, we switch indices to

$$\text{for all } j, \text{ do for all } i: \quad y_i \leftarrow y_i + a_{j,i} x_j.$$

The matrix-vector multiplication involving A^T is then given by

```
for i = 1, n
    y(i) = 0
end;
for j = 1, n
    for i = row_ptr(j), row_ptr(j+1)-1
        y(col_ind(i)) = y(col_ind(i)) + val(i) * x(j)
    end;
end;
```

Both matrix-vector products above have largely the same structure, and both use indirect addressing. Hence, their vectorizability properties are the same on any given computer. However, the first product ($y = Ax$) has a more favorable memory access pattern in that (per iteration of the outer loop) it reads two vectors of data (a row of matrix A and the input vector x) and writes one scalar. The transpose product ($y = A^T x$), on the other hand, reads one element of the input vector and one row of matrix A and both reads and writes the result vector y . Unless the machine on which these methods are implemented has three separate memory paths (e.g., Cray's vector computers), the memory traffic will then limit the performance. This is an important consideration for RISC-based architectures.

10.2.2.2 CDS Matrix-Vector Product

If the $n \times n$ matrix A is stored in CDS format, it is still possible to perform a matrix-vector product $y = Ax$ by either rows or columns, but this does not take advantage of the CDS format. The idea is to make a change in coordinates in the doubly nested loop. Replacing $j \rightarrow i + j$ we get

$$y_i \leftarrow y_i + a_{i,j} x_j \quad \Rightarrow \quad y_i \leftarrow y_i + a_{i,i+j} x_{i+j}.$$

With the index i in the inner loop we see that the expression $a_{i,i+j}$ accesses the j th diagonal of the matrix (where the main diagonal has number 0).

The algorithm will now have a doubly nested loop with the outer loop enumerating the diagonals $\text{diag}=-p,q$ with p and q the (nonnegative) numbers of diagonals to the left and right of the main diagonal. The bounds for the inner loop follow from the requirement that

$$1 \leq i, i + j \leq n.$$

The algorithm becomes

```

for i = 1, n
    y(i) = 0
end;
for diag = -diag_left, diag_right
    for loc = max(1,1-diag), min(n,n-diag)
        y(loc) = y(loc) + val(loc,diag) * x(loc+diag)
    end;
end;

```

The transpose matrix-vector product $y = A^T x$ is a minor variation of the algorithm above. Using the update formula

$$\begin{aligned} y_i &\leftarrow y_i + a_{i+j,i}x_j \\ &= y_i + a_{i+j,i+j-j}x_{i+j} \end{aligned}$$

we obtain

```

for i = 1, n
    y(i) = 0
end;
for diag = -diag_right, diag_left
    for loc = max(1,1-diag), min(n,n-diag)
        y(loc) = y(loc) + val(loc+diag, -diag) * x(loc+diag)
    end;
end;

```

The memory access for the CDS-based matrix-vector product $y = Ax$ (or $y = A^T x$) is three vectors per inner iteration. On the other hand, there is no indirect addressing, and the algorithm is vectorizable with vector lengths of essentially the matrix order n . Because of the regular data access, most machines can perform this algorithm efficiently by keeping three base registers and using simple offset addressing.

10.2.3 Fast Matrix-Vector Multiplication for Structured Matrices

Dense matrices that depend on $O(n)$ parameters arise often in practice. Examples of these matrices include

Vandermonde:

$$V = \begin{bmatrix} 1 & 1 & \dots & 1 \\ x_1 & x_2 & \dots & x_n \\ x_1^2 & x_2^2 & \dots & x_n^2 \\ \vdots & & & \\ x_1^{n-1} & x_2^{n-1} & \dots & x_n^{n-1} \end{bmatrix},$$

Toeplitz:

$$T = \begin{bmatrix} t_0 & t_{-1} & \dots & t_{2-n} & t_{1-n} \\ t_1 & t_0 & t_{-1} & & t_{2-n} \\ \vdots & t_1 & t_0 & \ddots & \vdots \\ & & \ddots & \ddots & t_{-1} \\ t_{n-2} & & & & t_{-1} \\ t_{n-1} & t_{n-2} & \dots & t_1 & t_0 \end{bmatrix},$$

Hankel: $H = (H_{ij}) = (c_{i+j})$,

Cauchy: $C = (C_{ij}) = (\frac{1}{x_i - y_j})$, and others [257].

These matrices and their inverses can be multiplied by vectors in $O(n \log^k n)$ time, where $0 \leq k \leq 2$, instead of $O(n^2)$ or $O(n^3)$ time, depending on the structure. This is particularly useful for using iterative solvers with these matrices. The iterative solvers are also useful when solving systems of the form $(A + B)x = b$, where A and B are both structured but belong to different structured classes, or if A is structured and B is banded or sparse, etc. Iterative methods are also useful for solving block systems when the blocks are structured matrices but belong to different structure classes with some blocks also possibly being sparse. For example, the product

$$\begin{bmatrix} A & B \\ C & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} Ax + By \\ Cx \end{bmatrix},$$

where A is diagonal and B and C are Toeplitz can be formed in $O(n \log n)$ time.

Vandermonde and inverses of Vandermonde matrices can be multiplied by a vector in $O(n \log^2 n)$ time [196, 195]. The same is valid for the Vandermonde-like and inverses of Vandermonde-like matrices, where “-like” matrices are defined as in § 10.3.4. Matrix-vector multiplication for Toeplitz and Toeplitz-like matrices takes $O(n \log n)$ time using the fast Fourier transform (FFT) [198]. The Cauchy matrix-vector product Cz is equivalent to the evaluation of the function

$$\Phi(w) = \sum_{i=1}^n \frac{-z_i}{y_i - w}$$

at n different points x_1, x_2, \dots, x_n and can be done in $O(n)$ time by using the fast multipole method [76, 205].

The rest of this section shows how the $O(n \log n)$ matrix-vector multiplication works for Toeplitz matrices [198]. For the other classes of structured matrices we refer the reader to [196] and [195].

A circulant matrix is a special Toeplitz matrix of the form

$$C_n = \begin{bmatrix} a_0 & a_{-1} & \dots & a_{2-n} & a_{1-n} \\ a_1 & a_0 & a_{-1} & & a_{2-n} \\ \vdots & a_1 & a_0 & \ddots & \vdots \\ & & \ddots & \ddots & a_{-1} \\ a_{n-2} & & & & a_{-1} \\ a_{n-1} & a_{n-2} & \dots & a_1 & a_0 \end{bmatrix},$$

which has the property that $a_{-k} = a_{n-k}$ for $1 \leq k \leq n-1$. Circulant matrices are diagonalized by the FFT, i.e.,

$$C_n = F_n^* \cdot \text{diag}(F_n a) \cdot F_n,$$

where $a = [a_0, a_1, \dots, a_n]$ and F_n is the Fourier matrix $F_n(j, k) = \frac{1}{\sqrt{n}} e^{-(j-1)(k-1)2\pi\sqrt{-1}/n}$. It is a well-known fact that F_n is unitary and a product $F_n x$ can be formed in $O(n \log n)$ time [453]. A product $y = C_n x$ can also be formed in $O(n \log n)$ time by the following four steps:

- (1) $f = F_n x$,
- (2) $g = F_n a$,
- (3) $z^T = [f_1 g_1, f_2 g_2, \dots, f_n g_n]$,
- (4) $y = F_n^* z$.

Now, if T_n is a Toeplitz matrix

$$T_n = \begin{bmatrix} t_0 & t_{-1} & \dots & t_{2-n} & t_{1-n} \\ t_1 & t_0 & t_{-1} & & t_{2-n} \\ \vdots & t_1 & t_0 & \ddots & \vdots \\ t_{n-2} & & \ddots & \ddots & t_{-1} \\ t_{n-1} & t_{n-2} & \dots & t_1 & t_0 \end{bmatrix},$$

then the product $T_n x$ can be computed in $O(n \log n)$ time by first embedding T_n into a $2n \times 2n$ circulant matrix C_{2n} , since

$$C_{2n} \cdot \begin{bmatrix} y \\ 0 \end{bmatrix} \equiv \begin{bmatrix} T_n & B_n \\ B_n & T_n \end{bmatrix} \cdot \begin{bmatrix} y \\ 0 \end{bmatrix} = \begin{bmatrix} T_n y \\ B_n y \end{bmatrix}$$

and

$$B_n = \begin{bmatrix} 0 & t_{n-1} & \dots & t_2 & t_1 \\ t_{1-n} & 0 & t_{n-1} & & t_2 \\ \vdots & t_{1-n} & 0 & \ddots & \vdots \\ t_{-2} & & \ddots & \ddots & t_{n-1} \\ t_{-1} & t_{-2} & \dots & t_{1-n} & 0 \end{bmatrix}.$$

10.3 A Brief Survey of Direct Linear Solvers

J. Demmel, P. Koev, and X. Li

Many of the most effective methods in this book, in particular those using shift-and-invert, require the solution of linear systems of the form $(A - \sigma I)x = b$ or $(A - \sigma I)^*x = b$, where σ is a *shift*, usually chosen to be an approximate eigenvalue of A . Much of the time in these methods can be spent solving these systems, so it is important to use the best methods available.

There are two basic approaches to solving $(A - \sigma I)x = b$: direct and iterative. Direct methods typically use variations on Gaussian elimination. They are effective even when σ is close to an eigenvalue and $A - \sigma I$ is nearly singular, which is when iterative methods

have most trouble converging. Section 10.4 discusses iterative solvers and when they can be effectively used in some eigenvalue algorithms, such as the Jacobi–Davidson method. This section considers only direct methods.

A direct method for solving $(A - \sigma I)x = b$ will consist of two steps:

1. Compute a *factorization* of $A - \sigma I$. This is the most expensive part.
2. Use the factorization to solve $(A - \sigma I)x = b$ or $(A - \sigma I)^*x = b$.

Step 1 usually costs much more than step 2 (e.g., $O(n^3)$ vs. $O(n^2)$ in the dense case). The factorization can be reused to solve $(A - \sigma I)x = b$ for many different right-hand sides b . The factorization only needs to be recomputed when the shift σ is changed. Fortunately, most iterative methods solve many linear systems with different right-hand sides and the same shift, so step 1 is performed many fewer times than step 2.

The choice of algorithm for steps 1 and 2 depends on both the *mathematical structure* of $A - \sigma I$ and the *storage structure* of A . By mathematical structure we most often mean whether $A - \sigma I$ is Hermitian or non-Hermitian, and definite or indefinite.

Hermitian and definite: In this case we compute the *Cholesky factorization* $A - \sigma I = \pm LL^T$, where L is a lower triangular matrix. The choice of sign depends on whether $A - \sigma I$ is positive definite (+) or negative definite (-). In the sparse case, we may instead compute $A - \sigma I = \pm PLL^*P^*$, where P is a permutation matrix.

Hermitian and indefinite: In this case we compute the *Hermitian indefinite factorization* $A - \sigma I = PLDL^*P^*$, where L is lower triangular, P is a permutation matrix, and D is block diagonal with 1 by 1 or 2 by 2 blocks, or perhaps tridiagonal.

Non-Hermitian: In this case we compute the *triangular factorization* $A - \sigma I = P_1 L U P_2$, where L is lower triangular, U is upper triangular, and P_1 and P_2 are permutations (sometimes one P_i is omitted).

There are many other structures and factorizations as well, such as when A is Toeplitz ($a_{i,j}$ depends only on $i - j$).

By storage structure we mean being dense, banded, sparse (in one of the formats described in §10.1), or structured (such as storing the first row and column of a Toeplitz matrix, which determines the other entries). A sparse non-Hermitian matrix may also be stored in a sparse Hermitian data structure, as in § 10.1.

There are specialized algorithms and software for many combinations of these mathematical and storage structures, and choosing the right algorithm can make orders of magnitude difference in solution time for large matrices. In this section we summarize the best algorithms and software available for these problems. It is often the case that the best algorithm in a research paper is not available as well designed and easily accessible software, and we shall concentrate on available software. We recommend software that is well maintained and in the public domain, or available at low cost, unless none is available.

We consider four cases: methods for dense matrices, methods for band matrices, methods for sparse matrices, and methods for structured matrices, such as Toeplitz matrices.

10.3.1 Direct Solvers for Dense Matrices

There are two reasons to consider an iterative method for dense matrices instead of using a transformation method:

1. If one only wants a few eigenvalues and/or eigenvectors near one or a few shifts σ , it may be cheaper to use shift-and-invert with an iterative scheme instead of a transformation method. This is more likely true with non-Hermitian matrices than Hermitian ones, for which the transformation methods are faster.
2. When a matrix is not very sparse, or not very large, a dense solver may be faster than a sparse solver.

The choice of dense solver depends on the mathematical structure of $A - \sigma I$ as follows:

A - σI is Hermitian definite. In this case Cholesky is the algorithm of choice. It is implemented in LAPACK computational routines `xPOTRF` to compute the factorization and `xPOTRS` to solve using the factorization (both are combined in LAPACK driver routine `xPOSVX`). Versions for packed data storage are also available (substitute PP for PO). Cholesky is implemented in analogous ScaLAPACK routines `PxPOTRF`, `PxPOTRS`, and `PxPOSVX`. The factorization is in MATLAB as `chol`.

A - σI is Hermitian indefinite. In this case Bunch-Kaufman is the algorithm of choice. It is implemented in real (complex) LAPACK computational routines `xSYTRF`(`xHETRF`) to compute the factorization and `xSYTRS`(`xHETRS`) to solve using the factorization (both are combined in LAPACK driver routine `xSYSVX`(`xHESVX`)). Versions for packed data storage are also available (substitute SP(HP) for SY(HE)). It is not available in ScaLAPACK or MATLAB.

A - σI is non-Hermitian. In this case Gaussian elimination is the algorithm of choice. It is implemented in LAPACK computational routines `xGETRF` to compute the factorization and `xGETRS` to solve using the factorization (both are combined in LAPACK driver routine `xGESVX`). It is implemented in analogous ScaLAPACK routines `PxGETRF`, `PxGETRS`, and `PxGESVX`. The factorization is in MATLAB as `lu`.

10.3.2 Direct Solvers for Band Matrices

This section is analogous to §10.3.1 for dense matrices:

A - σI is Hermitian definite. In this case Cholesky is the algorithm of choice. It is implemented in LAPACK computational routines `xPBTRF` to compute the factorization and `xPBTRS` to solve using the factorization (both are combined in LAPACK driver routine `xPBSVX`). Cholesky is implemented in analogous ScaLAPACK routines `PxPBTRF`, `PxPBTRS`, and `PxPBSV`.

A - σI is non-Hermitian. In this case Gaussian elimination is the algorithm of choice. It is implemented in LAPACK computational routines `xGBTRF` to compute the factorization and `xGBTRS` to solve using the factorization (both are combined in

LAPACK driver routine `xGBSVX`). It is implemented in analogous ScaLAPACK routines `PxGBTRF`, `PxGBTRS`, and `PxGBSV` with partial pivoting, and `PxDBTRF`, `PxDBTRS`, and `PxDBSV` without pivoting (which is riskier but faster than using pivoting).

No routines exploiting Hermitian indefinite structure are available (because there is no simple bound on how much the bandwidth can grow because of pivoting). No band routines are in MATLAB. If $A - \sigma I$ is Hermitian and the bandwidth is narrow enough (such as tridiagonal), direct eigensolver methods from §4.2 should be used.

10.3.3 Direct Solvers for Sparse Matrices

Direct solvers for sparse matrices involve much more complicated algorithms than for dense matrices. The main complication is due to the need for efficiently handling the *fill-in* in the factors L and U . A typical sparse solver consists of four distinct steps as opposed to two in the dense case:

1. An ordering step that reorders the rows and columns such that the factors suffer little fill, or that the matrix has special structure, such as block-triangular form.
2. An analysis step or symbolic factorization that determines the nonzero structures of the factors and creates suitable data structures for the factors.
3. Numerical factorization that computes the L and U factors.
4. A solve step that performs forward and back substitution using the factors.

There is a vast variety of algorithms associated with each step. The review papers by Duff [137] (see also [135, Chapter 6]) and Heath, Ng, and Peyton [219] can serve as excellent references for various algorithms. Usually steps 1 and 2 involve only the graphs of the matrices, and hence only integer operations. Steps 3 and 4 involve floating-point operations. Step 3 is usually the most time-consuming part, whereas step 4 is about an order of magnitude faster. The algorithm used in step 1 is quite independent of that used in step 3. But the algorithm in step 2 is often closely related to that of step 3. In a solver for the simplest systems, i.e., symmetric and positive definite systems, the four steps can be well separated. For the most general unsymmetric systems, the solver may combine steps 2 and 3 (e.g. SuperLU) or even combine steps 1, 2 and 3 (e.g. UMFPACK) so that the numerical values also play a role in determining the elimination order.

In the past 10 years, many new algorithms and much new software have emerged which exploit new architectural features, such as memory hierarchy and parallelism. In Table 10.1, we compose a rather comprehensive list of sparse direct solvers. It is most convenient to organize the software in three categories: the software for serial machines, the software for SMPs, and the software for distributed memory parallel machines.

There is no single algorithm or software that is best for all types of linear systems. Some software is targeted for special matrices such as symmetric and positive definite, some is targeted for the most general cases. This is reflected in column 3 of the table, “Scope.” Even for the same scope, the software may decide to use a particular algorithm or implementation technique, which is better for certain applications but not

Table 10.1: Software to solve sparse linear systems using direct methods.

Code	Technique	Scope	Contact	
Serial platforms				
MA27	Multifrontal	Sym	HSL	[140]
MA41	Multifrontal	Sym-pat	HSL	[6]
MA42	Frontal	Unsym	HSL	[144]
MA47	Multifrontal	Sym	HSL	[141]
MA48	Right-looking	Unsym	HSL	[142]
SPARSE	Right-looking	Unsym	Kundert	[281]
SPARSPAK	Left-looking	SPD	George	[191]
SPOOLES	Left-looking	Sym and Sym-pat	Ashcraft	[21]
SuperLLT	Left-looking	SPD	Ng	[339]
SuperLU	Left-looking	Unsym	Li	[126]
UMFPACK	Multifrontal	Unsym	Davis	[103]
Shared memory parallel machines				
Cholesky	Left-looking	SPD	Rothberg	[405]
DMF	Multifrontal	Sym	Lucas	[308]
MA41	Multifrontal	Sym-pat	HSL	[7]
PanelLLT	Left-looking	SPD	Ng	[211]
PARASPAR	Right-looking	Unsym	Zlatev	[471]
PARDISO	Left-right looking	Sym-pat	Schenk	[395]
SPOOLES	Left-looking	Sym and Sym-pat	Ashcraft	[21]
SuperLU_MT	Left-looking	Unsym	Li	[127]
Distributed memory parallel machines				
CAPSS	Multifrontal	SPD	Raghavan	[220]
DMF	Multifrontal	Sym	Lucas	[308]
MUMPS	Multifrontal	Sym and Sym-pat	Amestoy	[8]
PaStiX	Left-right looking*	SPD	CEA	[224]
PSPASES	Multifrontal	SPD	Gupta	[210]
SPOOLES	Left-looking	Sym and Sym-pat	Ashcraft	[21]
SuperLU_DIST	Right-looking	Unsym	Li	[306]
S+	Right-looking†	Unsym	Yang	[182]

* In spite of the title of the paper

† Uses QR storage to statically accommodate any LU fill-in

Abbreviations used in the table: SPD = symmetric and positive definite; Sym = symmetric and may be indefinite; Sym-pat = symmetric nonzero pattern but unsymmetric values; Unsym = unsymmetric; HSL = Harwell Subroutine Library: <http://www.cse.clrc.ac.uk/Activity/HSL>

for others. In column 2, “Technique,” we give a high-level algorithmic description. For a review of the distinctions between left-looking, right-looking, and multifrontal and their implications on performance, we refer the reader to the papers by Heath, Ng, and Peyton [219] and Rothberg [370]. Sometimes the best (or only) software is not in the public domain, but available commercially or in research prototypes. This is reflected in column 4, “Contact,” which could be the name of a company or the name of the author of the research code.

In the context of shift-and-invert spectral transformation (SI) for eigensystem analysis, we need to factorize $A - \sigma I$, where A is fixed. Therefore, the nonzero structure of $A - \sigma I$ is fixed. Furthermore, for the same shift σ , it is common to solve many systems with the same matrix and different right-hand sides (in which case the solve cost can be comparable to factorization cost). It is reasonable to spend a little more time in steps 1 and 2 to speed up steps 3 and 4. That is, one can try different ordering schemes and estimate the costs of numerical factorization and solutions based on symbolic factorization, and use the best ordering. For instance, in computing the SVD, one has the choice between shift-and-invert on AA^* , A^*A , and $\begin{bmatrix} 0 & A \\ A^* & 0 \end{bmatrix}$, all of which can have rather different factorization costs, as discussed in Chapter 6.

Some solvers have the ordering schemes built in, but others do not. It is also possible that the built-in ordering schemes are not the best for the target applications. It is sometimes better to substitute an external ordering scheme for the built-in one. Many solvers provide well-defined interfaces so that the user can make this substitution easily. One should read the solver documentation to see how to do this, as well as to find out the recommended ordering methods.

10.3.4 Direct Solvers for Structured Matrices

The linear system of equations involving the structured matrices described in §10.2.3 have the attractive property of being solvable in $O(n^2)$ time instead of $O(n^3)$.

The common property that makes it possible to solve such systems fast (in $O(n^2)$ time) is the fact that they have a *low displacement rank* [257].

Displacement rank may be defined as follows: For given matrices A and F and a matrix T we define the operator

$$\Delta_{A,F}(T) = A \cdot T - T \cdot F$$

and denote $\alpha = \text{rank}(\Delta_{A,F}(T))$. We say that α is the “displacement rank” of T with respect to the operator $\Delta_{A,F}$. If α is small (usually $O(1)$) we say that the matrix T has a “low displacement rank.” Since $\text{rank}(\Delta_{A,F}(T)) = \alpha$ there exist $n \times \alpha$ matrices G and B such that $\Delta_{A,F}(T) = G \cdot B^T$. The matrices G and B are called *generators* of T . We will refer to matrices with low displacement rank as having *displacement structure*.

For an example for a Cauchy matrix $C = (C_{ij}) = (1/(x_i - y_j))$ we have

$$\Delta_{\text{diag}(x), \text{diag}(y)}(C) = \text{diag}(x) \cdot C - C \cdot \text{diag}(y) = (1, 1, \dots, 1)^T \cdot (1, 1, \dots, 1).$$

Therefore, a Cauchy matrix has displacement rank 1 and generators $G = B = (1, 1, \dots, 1)^T$

A matrix with low displacement rank α (not necessarily equal to 1) with respect to the operator $\Delta_{\text{diag}(x), \text{diag}(y)}$ is called Cauchy-like. In a similar way one defines Vandermonde-like matrices, Toeplitz-like matrices, etc. These higher displacement rank systems are solvable in $O(\alpha n^2)$ time.

Fast algorithms exploit the displacement equation in order to produce an LU factorization of the matrix F . The algorithms work on the generators of a matrix instead of the matrix itself, which permits them to go much faster.

The block-Toeplitz matrices and matrices of type $T^T T$, where T is Toeplitz, also have a low displacement rank [257].

If A and B have displacement structure, then systems of type $(A + \sigma B)x = b$ can be solved using the fast low displacement rank methods only if $A + \sigma B$ has displacement

structure as well (with respect to a possibly different displacement operator). For example, if A is Hankel and $B = I$ then $A + \sigma B$ is Toeplitz-plus-Hankel, which also has displacement structure. If $A + \sigma B$ does not have displacement structure then one would be restricted to using zero-shifts $\sigma = 0$ or fast iterative methods for which only a fast matrix-vector product is needed [257].

Some of the methods impose additional restrictions on the matrices to be symmetric (Toeplitz) or positive definite. This may result in additional restrictions on the choice of shifts available.

Special attention should be exercised when using fast algorithms for structured matrices since the speed often comes at the price of accuracy. Some displacement-rank algorithms simulate Gaussian elimination (with partial, complete, or no pivoting) by working on the generators instead of on the matrices themselves [193]. Even so, these algorithms need not have the same numerical properties as Gaussian Elimination because of the possibility of “generator growth” [257], which is uncommon, but can appear even for very well conditioned matrices.

There are many methods to stabilize the fast algorithms including the problem of generator growth [257, p. 111], and despite the occasional instabilities these methods are very attractive.

The fast algorithms are very well described in the literature, but reliable software libraries are not available. One should also be aware of the constants hidden in the $O(n^2)$ notation and that those constants make the fast methods faster than the traditional $O(n^3)$ methods only for n sufficiently large (usually at least a few hundred, depending on the structure). The traditional $O(n^3)$ algorithms are often optimized to use BLAS 3 and use the computer’s memory hierarchy efficiently in order for the code to run near the full speed of the processor (see §10.2.1). The fast algorithms can usually only use BLAS 2 and it may be difficult or impossible to optimize the fast algorithms to make efficient use of the computer’s cache and fast memory. This means that even if a fast $O(n^2)$ algorithm and a slow $O(n^3)$ algorithm perform the same number of floating point operations, the “slow” algorithm is very likely to be faster because of these optimization issues.

We should also mention the Bjorck-Pereyra-type algorithms for solving Vandermonde and three-term Vandermonde systems [51, 230]. These $O(n^2)$ methods have remarkable numerical properties. Under some additional restrictions on the order of the nodes in the Vandermonde matrix and the sign pattern of the right-hand side, it is possible to solve those systems to the full machine precision.

10.4 A Brief Survey of Iterative Linear Solvers

H. van der Vorst

In the context of eigenproblems it may be necessary to solve a linear system, for instance, in the following situations:

- The Jacobi–Davidson methods require the (approximate) solution of a linear correction equation.
- Inexact methods, like those discussed in Chapter 11, are based on an approximate shift-and-invert step, for which a linear system needs to be approximately solved.

- Given a good approximation for an eigenvalue, the corresponding left or right eigenvector can be computed from a linear system with the shifted matrix.

In all these cases, one has to solve a linear system with a shifted matrix $A - \theta I$, sometimes embedded in projections as in the Jacobi–Davidson methods. If such systems have to be solved accurately, then direct (sparse) solvers may be considered first. Often one has to solve more than one system with the same shifted matrix, which helps to amortize large costs involved in making a sparse LU decomposition. If the systems need not be solved accurately, or if direct solution methods are too expensive, then iterative methods may be considered. A set of popular iteration methods has been described in *Templates for the Solution of Linear Systems* [41]. Before we present a short overview, we warn the reader that the linear systems related to eigenproblems usually have an indefinite matrix, because of the involved shift. If the shift is near an exterior eigenvalue, then this is not necessarily an obstacle in all cases, but there will certainly be convergence problems for interior shifts. Also, when the shift is very close to an eigenvalue, for instance, if one wants to determine a left or right eigenvalue, then one should realize that iterative methods have great difficulty solving almost singular systems. This holds in particular for the situations of interest, where one is interested in the (almost) singular directions, as is the case in inverse iteration for left and right eigenvectors. It is commonly accepted that iterative methods need efficient preconditioners in order to be attractive. This is in particular the case for shifted matrices. Unfortunately, the construction of effective preconditioners for indefinite matrices is slippery ground to treat upon. See Chapter 11 for more on this.

The currently most popular iterative methods belong to the class of Krylov subspace methods. These methods construct approximations for the solution from the so-called Krylov subspace. The Krylov subspace $\mathcal{K}^i(A; r_0)$ of dimension i , associated with a linear system $Ax = b$ (where A and b may be the preconditioned values, if preconditioning is included) for a starting vector x_0 with residual vector $r_0 = b - Ax_0$ is defined as the subspace spanned by the vectors $\{r_0, Ar_0, A^2r_0, \dots, A^{i-1}r_0\}$.

The different methods can be classified as follows:

- If A is symmetric positive definite, then the *conjugate gradient* method [226] generates, using two-term recurrences, the x_i for which $(x - x_i, A(x - x_i))$ (the so-called A -norm or energy norm) is minimized over all vectors in the current Krylov subspace $\mathcal{K}^i(A; r_0)$.
- If A is only symmetric but not positive definite, then the *Lanczos* [286] and the *MINRES* methods [350] may be considered. In MINRES, the $x_i \in \mathcal{K}^i(A; r_0)$ is determined for which the 2-norm of the residual ($\|b - Ax_i\|_2$) is minimized, while the Lanczos method leads to an x_i for which $b - Ax_i$ is perpendicular to the Krylov subspace.
- If A is unsymmetric, it is in general not possible to determine an optimal $x_i \in \mathcal{K}^i(A, r_0)$ with short recurrences. This was proved in [163]. However, with short recurrences as in conjugate gradients (and MINRES), we can compute the $x_i \in \mathcal{K}^i(A; r_0)$, for which $b - Ax_i \perp \mathcal{K}^i(A^T; s_0)$ (usually, one selects $s_0 = r_0$). This leads to the *bi-conjugate gradient method* [169]. A clever variant is quasi-minimal residual (QMR) [179], which has smoother convergence behavior and is more robust than bi-conjugate gradients.

- (d) If A is unsymmetric, then we can compute the $x_i \in \mathcal{K}^i(A, r_0)$, for which the residual is minimized in the Euclidean norm. This is done by the GMRES method [389]. This requires i inner products at the i th iteration step, as well as i vector updates, which means that the iteration costs, that come in addition to operations with A , grow linearly with i .
- (e) The operations with A^T in the bi-conjugate gradient method can be replaced by operations with A itself, by using the observation that $\langle x, A^T y \rangle$ equals $\langle Ax, y \rangle$, where $\langle \dots \rangle$ represents the inner-product computation. Since the function of the multiplications by A^T in bi-conjugate gradient serves only to maintain the dual space to which residuals are orthogonalized, the replacement of this operator by A allows us to expand the Krylov subspace and to find better approximations for virtually the same costs per iteration as for bi-conjugate gradients. This leads to so-called hybrid methods such as conjugate gradients squared [418], Bi-CGSTAB [445], Bi-CGSTAB(ℓ) [409], TFQMR [174], and hybrids of QMR [78].
- (f) For indefinite systems it may also be effective to apply the conjugate gradient method for the normal equations $A^T A x = A^T b$. Carrying this out in a straight-forward manner may lead to numerical instabilities, because $A^T A$ has a squared condition number. A clever and robust implementation is provided in least squares QR [351].

Many of these methods have been described in [41], and software for them is available. See this book's homepage, ETHOME, for guidelines on how to obtain the software. For some methods, descriptions in templates style have been given in [135]. That book also contains an overview on various preconditioning approaches.

10.5 Parallelism

J. Dongarra and X. Li

In this section we discuss aspects of parallelism in the iterative methods discussed in this book. Since the iterative methods share most of their computational kernels we will discuss these independent of the method. The basic time-consuming kernels of iterative schemes are:

- inner products;
- vector updates;
- matrix-vector products, e.g., $Ap^{(i)}$ (for some methods also $A^T p^{(i)}$);
- solvers for $(A - \sigma B)x = b$.

Inner Products. The computation of an inner product of two vectors can be easily parallelized; each processor computes the inner product of corresponding segments of each vector (local inner products (LIPs)). On distributed memory machines the LIPs then have to be sent to other processors to be combined for the global inner product. This can be done either with an all-to-all send where every processor performs the summation of the LIPs, or by a global accumulation in one processor, followed by a broadcast of the final result. Clearly, this step requires communication.

For shared memory machines, the accumulation of LIPs can be implemented as a critical section where all processors add their local result in turn to the global result, or as a piece of serial code, where one processor performs the summations.

Vector Updates. Vector updates are trivially parallelizable: each processor updates its own segment.

Matrix-Vector Products. The matrix-vector products are often easily parallelized on shared memory machines by splitting the matrix in strips of rows corresponding to the vector segments. Each processor then computes the matrix-vector product of one strip. For distributed memory machines, there may be a problem if each processor has only a segment of the vector in its memory. Depending on the bandwidth of the matrix, we may need communication for other elements of the vector, which may lead to communication bottlenecks. However, many sparse matrix problems arise from a network in which only nearby nodes are connected. For example, matrices stemming from finite difference or finite element problems typically involve only local connections: matrix element $a_{i,j}$ is nonzero only if variables i and j are physically close. In such a case, it seems natural to subdivide the network, or grid, into suitable blocks and to distribute them over the processors. When computing $Ap^{(i)}$, each processor requires the values of $p^{(i)}$ at some nodes in neighboring blocks. If the number of connections to these neighboring blocks is small compared to the number of internal nodes, then the communication time can be overlapped with computational work.

More recently, *graph partitioning* has been used as a powerful tool to deal with the general problems with nonlocal connections. Consider $y \leftarrow Ax$ and the undirected graph G of the (symmetric) matrix A . Assume vertex i stores x_i, y_i , and the nonzero $a_{i,j}$ for all j . Let vertex i represent the job of computing $y_i = \sum_j a_{i,j}x_j$. We can assign weight of vertex i to be the number of operations for computing y_i , and the weight of edge (i,j) to be 1, which represents the cost of sending x_j to vertex i if vertices i and j were mapped on different processors. A good graph partitioning heuristic would partition G into P subsets of vertices corresponding to P processors, such that:

- the sums of vertex weights are approximately equal among the subsets,
- the sum of edge cuts crossing the subsets is minimized.

This ensures a good load balance and minimizes communication. Good graph partitioning software includes Chaco [223] and METIS [259] (also ParMETIS, the parallel version).

After partitioning, further optimizations can be performed to reduce communication. For example, if a subset of vertices contains several edges to another subset, the corresponding elements of the vector can be packed into one message, so that each processor sends no more than one message to another processor. For more detailed discussions on implementation aspects for distributed memory systems, see De Sturler and van der Vorst [111, 112] and Pommerell [369].

High-quality parallel algorithms for distributed memory machines are implemented in software packages such as Aztec [236] and PETSc [38]. The software can be obtained via the book's homepage, ETHOME.

Solvers. In addition to the three kernels above, the iterative methods using shift-and-invert often require the direct solutions of the linear systems. Both matrix factorization and triangular solve involve much more complicated parallel algorithms, especially on massively parallel machines. There has been a large amount of research activity in this area. Many state-of-the-art parallel algorithms for dense and band matrices are implemented in ScaLAPACK (see § 10.3), and those for sparse matrices are implemented in the software packages surveyed in Table 10.1.

Chapter 11

Preconditioning Techniques

11.1 Introduction

The term *preconditioning* is often used in a numerical method as an extra step designed to accelerate the convergence. Preconditioning is an important technique in iterative methods for solving large systems of linear equations. Preconditioning for eigenvalue problems is not as straightforward and has been slower to catch on. There is a large set of literature in Russian and some of it is discussed in §11.3. Ruhe and Wiberg [380, 373, 374] used SOR and the conjugate gradient method to find eigenvalues in the 1970s. Around the same time, quantum chemists, including Davidson [99], developed correction methods for their large symmetric matrices. Later Scott recognized that Davidson's approach is a form of preconditioning, and generalizations were given in [335]. A recent development, the Jacobi–Davidson method, has been presented in detail in earlier chapters.

In this introduction, we attempt to point out some connections, first with some comparisons between preconditioning eigenvalues and linear equations. Then there is a brief discussion of preconditioned versus unpreconditioned methods. And finally, connections are given between various preconditioned eigensolvers.

In this paragraph, we look at how the task of preconditioning compares for eigenvalues versus linear equations. We give several reasons why preconditioning eigenvalue problems is more difficult. The matrix being preconditioned is generally nearly singular. Interior eigenvalues are difficult, similar to preconditioning indefinite systems of linear equations. Unlike preconditioned linear equations, preconditioned eigenvalue methods may need some sort of restarting, even in the symmetric case. However, restarting is often necessary even for unpreconditioned Krylov eigenvalue methods that compute the eigenvector. Finally, we mention that eigenvalue preconditioning methods tend to focus on computing one eigenvalue at a time. This is different from Krylov eigenvalue methods. There is no similar added difficulty in going from unpreconditioned to preconditioned linear equation methods.

In spite of these difficulties, which no doubt contributed to the slow development in preconditioning eigenvalue problems, there is much potential in situations where a good preconditioner can be found. A number of factors should go into the decision as to whether a preconditioning method is best. These include the effectiveness of the preconditioner, the number of eigenvalues desired, whether eigenvectors are needed, and

the competing methods in the particular situation. Generally, preconditioning is more competitive if not too many eigenvalues are desired and if computing the eigenvectors is required. We also want to mention that finding eigenvalues far in the interior of the spectrum is so tough for Krylov methods that either preconditioning or complete factorization of a shifted matrix must be considered. Mesh eigenproblems (see §11.3.1) is one important practical example of a class of eigenproblems, where preconditioning seems particularly promising.

The rest of this introduction looks at similarities between the various preconditioning methods that will be described in this chapter. It was already mentioned that preconditioning methods generally find just one eigenvalue at a time. This is quite different from an unpreconditioned Krylov subspace method, which can find several eigenvalues simultaneously. Sometimes a block approach is used to alleviate this problem for a preconditioning method; e.g., see §11.3.9. Another similarity among preconditioning methods is that global convergence may be trickier. Some methods, e.g., the locally optimal preconditioned conjugate gradient method of §11.3.8, are practically very robust and typically converge with a random initial guess in numerical experiments. However, some other methods, e.g., based on inner-outer iterations, work better if reasonable approximations to eigenvectors are available as starting vectors. This leads us to the topic of hybrid methods, with a Krylov approach to start with and a preconditioning approach to follow. In fact, a number of operators could be used for the Krylov method including A , M^{-1} (where M is an approximation to a shifted matrix $A - \nu I$), and even the preconditioned operator $M^{-1}(A - \nu I)$, for ν and M fixed. Hybrid methods could deliver more robustness, and they deserve further examination.

We look at one more similarity among eigenvalue preconditioning methods. For ease of presenting, we will assume there is a standard symmetric eigenvalue problem. It appears that common to all methods is the operator

$$M^{-1}(A - \nu I), \quad (11.1)$$

where ν is an approximate eigenvalue and M is a (possibly changing) approximation to $A - \nu I$. We will look at some examples of how different methods use this operator here; for a general discussion see §11.3.2. The four methods discussed are the preconditioned power method, the Rayleigh quotient iteration (RQI), the Davidson method, and the Jacobi–Davidson method. The preconditioned power method is Algorithm 11.5 in §11.3. With $B = I$, steps 4 and 5 of this algorithm perform

$$w = T(x^{(i)} - \mu^{(i)} A x^{(i)}) = -\mu^{(i)} T(A - 1/\mu^{(i)} I) x^{(i)}.$$

Letting $T = M^{-1}$ and $1/\mu^{(i)} = \nu$, we see the operator (11.1). The other algorithms in §11.3 have something similar as operator.

Next, the RQI with the preconditioned conjugate gradient method (PCG) of §11.3.7 has the operator $M^{-1}(A - \rho I)$ in the inner iteration of PCG, where ρ is the Rayleigh quotient and M is the preconditioner. And Davidson's original method, Algorithm 11.2 of §11.2.4 (see also §11.3.6) for standard eigenvalue problems, uses $(D - \nu I)^{-1}(A - \nu I)$, where D is the diagonal of A and ν is the current approximate eigenvalue that is found by the Rayleigh–Ritz procedure. Again this fits the form of (11.1). So if the same preconditioning is used, RQI (with PCG) and the Davidson method have the same operator at the core of the method. The difference is that the Davidson method changes ν at every step, while RQI has ρ fixed during a run of PCG, and also the Davidson

method uses the vectors it generates to solve an eigenvalue problem, while RQI uses them for solving linear equations. The linear equations in RQI are an intermediate step toward the eventual goal of solving the eigenvalue problem.

Finally we look at Jacobi–Davidson method; see §11.2.5 and §11.3.7. What is interesting is that the operator (11.1) shows up twice. First, in the outer iteration, from the Jacobi–Davidson correction equation (4.49), letting $M = (I - zz^*)(A - \theta I)(I - zz^*)$, we have $t \approx -M^{-1}r = -M^{-1}(A - \theta I)z$. Second and more importantly, in the inner iteration of generating the approximate solution to $t = -M^{-1}r$ using a preconditioned Krylov subspace iterative method, the operator is $P^{-1}(I - zz^*)(A - \theta I)(I - zz^*)$, where P is the preconditioner for $(I - zz^*)(A - \theta I)(I - zz^*)$. This operator is similar to (11.1), with an added deflation of z . The subspace generated by this preconditioned operator is used for the solution of linear equations.

With many eigenvalue preconditioning methods using at their core closely related operators, results with the same preconditioner might be expected to be similar. This is not the case. The implementation particular to each method can make a big difference, both in convergence rates and in expenses. For example, it will be discussed in the next section that inexact Cayley transforms (and the Jacobi–Davidson method) of §11.2.5 are much better than inexact shift-and-invert Krylov methods, even though they are similar. This is due partly to the fact that the $M^{-1}(A - \nu I)$ operator in the shift-and-invert method has ν fixed, instead of approaching an eigenvalue (the ν is changed in the inexact rational Krylov method, also mentioned in the next section). We can say that all the preconditioning methods share a limitation. They can only be as effective as $M^{-1}(A - \nu I)$ allows them to be. This depends on how good a preconditioner M is.

The Davidson method is perhaps the purest preconditioning method in the sense that with every application of the operator, it uses the best information known (new approximations to the desired eigenpair), and it applies the vectors it generates directly to an eigenvalue problem. On the other hand, there is significant expense for every iteration. Meanwhile, for some other methods, including RQI and Jacobi–Davidson, the inner iteration of a preconditioned linear equations iterative method can be more efficient in both expense and storage. Jacobi–Davidson has this efficiency and also greater robustness than RQI because it uses a subspace for the eigenvalue problem instead of one vector, and because it uses the residual of an approximate eigenvector as the right-hand side of the linear equations, instead of the vector itself.

Finally, the simultaneous PCG method, Algorithm 11.11 of §11.3.9, the most promising method of §11.3, has advantages of fast convergence of the Davidson method without its significant cost and applies the PCG directly to an eigenvalue problem, thus removing any need for inner iterations. Numerical experiments show great practical robustness of the method with respect to initial approximations, a particular choice of the preconditioner, and condition number of the matrix A .

The two sections that follow both look at some of the details of preconditioning eigenvalue problems, but with some different insights.

11.2 Inexact Methods

K. Meerbergen and R. Morgan

The spectral transformation Lanczos method for the Hermitian eigenvalue problem (HEP) and the shift-and-invert Arnoldi method for the non-Hermitian eigenvalue prob-

lem (NHEP) require the solution of linear systems. Usually, direct methods are employed because one can take advantage of one factorization for several back transformations. However, when the use of a direct solver becomes prohibitive, iterative solution methods must be used. Direct methods usually deliver a small residual norm, while the residual norm depends on a tolerance in an iterative method. An iterative method becomes more expensive when a lower residual tolerance is used. Therefore, it is advantageous not to solve the linear system very accurately when an iterative linear system solver is used. For this reason, we call the spectral transformation eigenvalue solvers exact when direct methods are used and inexact when iterative methods are used. The aim of this section is the study of the use of *inexact* linear system solvers in the spectral transformation.

We start from the shift-and-invert Arnoldi method and gradually move towards the (Jacobi) Davidson method and inexact rational Krylov method. First, in §11.2.1, we recall the (exact) matrix transformations used and give their main properties. In §11.2.2, we introduce the notion of inexact transformation [323] and explain its importance. In §11.2.3 and §11.2.4, we give some results for the Rayleigh–Ritz procedure with inexact transformations. This includes results for nonsymmetric problems (Arnoldi [323], Davidson [332]) and symmetric problems (Lanczos [336], Davidson [335, 88]) and the Jacobi–Davidson method. We can also use the matrix recurrence relation of the rational Krylov method [291] for computing eigenvalues, even when linear systems are solved inaccurately. This is studied in §11.2.7.

11.2.1 Matrix Transformations

Consider the eigenvalue problem $Ax = \lambda Bx$. The spectral transformation or shift-and-invert transformation (SI) is defined by

$$T_{\text{SI}} = (A - \mu B)^{-1}B,$$

where μ is the shift or pole. If $Ax = \lambda Bx$ then $T_{\text{SI}}x = \gamma x$ with $\gamma = (\lambda - \mu)^{-1}$. An alternative is the Cayley transform

$$T_C = (A - \mu B)^{-1}(A - \nu B),$$

where μ is the pole and ν the zero. If $Ax = \lambda Bx$ then $T_Cx = \zeta x$ with $\zeta = (\lambda - \mu)^{-1}(\lambda - \nu)$. Since $T_C = I + (\mu - \nu)T_{\text{SI}}$ and Krylov spaces are shift-invariant with respect to the matrix, we have that

$$\mathcal{K}^k(T_C, v_1) = \mathcal{K}^k(T_{\text{SI}}, v_1),$$

so, the Arnoldi method applied to T_{SI} or T_C delivers the same Ritz vectors and after back transformation of γ 's and ζ 's, respectively, leads to the same λ 's.

11.2.2 Inexact Matrix Transformations

The analysis is given for the Cayley transform only, though shift-and-invert could be used as well. The reasons are threefold. First, the shift-and-invert Arnoldi method and the Cayley transform Arnoldi method produce the same Ritz vectors. Second, it makes a link with the (Jacobi) Davidson methods easier to establish. Third, the Cayley transform leads to a more natural approach to the problem.

In the Arnoldi method applied to the Cayley transform, T_C , we must solve a sequence of linear systems

$$(A - \mu B)w_j = (A - \nu B)v_j - s_j$$

for $j = 1, \dots, k$, where s_j is the residual and v_j the last Krylov basis vector. The result w_j is orthonormalized against v_1, \dots, v_j into v_{j+1} . One could also apply the Cayley transform to another vector in the Krylov space rather than to v_j . In this case, we solve the linear system

$$(A - \mu B)w_j = (A - \nu B)y_j - s_j, \quad (11.2)$$

where y_j is chosen as follows: it may be a Ritz vector (Jacobi–Davidson) or the last basis vector (Arnoldi) or the continuation vector (rational Krylov). The residual norm $\|s_j\|$ is a measure of the backward error since we can rewrite (11.2) as

$$(A - \mu B + s_j w_j^* / \|w_j\|^2)w_j = (A - \nu B)y_j.$$

When a direct method is used, it typically happens that $s_j / \|A - \mu B\| \|w_j\|$ is a modest multiple of machine precision. We say that the direct method computes a backward stable solution of the linear system. Since $\|s_j\|$ is very small, we talk about an exact eigenvalue solver. Obtaining a small residual norm by an iterative method is also possible but is usually very expensive. In this section, we study the situation where $\|s_j\|$ is large. In order to give an indication of what we mean by large, a few words about iterative linear system solvers are needed. A linear system $Cx = b$ is said to be solved with a *relative residual tolerance* τ when the solution, x , satisfies $\|b - Cx\| \leq \tau \|b\|$. Krylov methods [388] are typically used. GMRES [389], BiCGSTAB(ℓ) [410], and QMR [179] are among those most widely used. See [41] for templates for all these solvers. Their performance substantially improves when a suitable preconditioner is employed. Hence what we mean by a large error is that $10^{-8} \leq \tau \leq 10^{-2}$.

By putting all the s_j for $j = 1, \dots, k-1$ together in $S_{k-1} = [s_1, \dots, s_{k-1}]$, w_j in W_{k-1} and y_j in Y_{k-1} , we obtain

$$(A - \mu B)W_{k-1} = (A - \nu B)Y_{k-1} - S_{k-1}. \quad (11.3)$$

Let W_{k-1}^\dagger be the generalized Moore–Penrose inverse of W_{k-1} , i.e., $W_{k-1}^\dagger W_{k-1} = I$, and define $E_{k-1} = S_{k-1} W_{k-1}^\dagger$. We can rewrite the relation as

$$(A - \mu B + E_{k-1})W_{k-1} = (A - \nu B)Y_{k-1}. \quad (11.4)$$

This is the relation that we should obtain after $k-1$ steps with the “inexact” Cayley transform

$$T_{IC} = (A - \mu B + E_{k-1})^{-1}(A - \nu B).$$

Note that for a fixed ν and μ , T_{IC} depends on k and the way each of the linear systems (11.2) are solved. When (11.2) is solved by a preconditioner M or a stationary linear system solver, i.e., $w_j = M^{-1}(A - \nu B)y_j$ for $j = 1, \dots, k-1$, then $E_{k-1} = E \equiv M - (A - \mu B)$ is independent of k and y_j and so is T_{IC} .

Eigenpairs are computed from $\text{span}\{v_1, v_2, \dots, v_k\}$. In Arnoldi’s method, this happens by the Hessenberg matrix that arises from the orthogonalization of W_{k-1} ; in the Davidson method, one uses the projection with A and B , e.g., $V_k^* A V_k z = \theta V_k^* B V_k z$.

$\text{Span}(v_1, \dots, v_k)$ contains the eigenvectors associated with the well-separated extreme eigenvalues of T_{IC} . This is a perturbed T_{C} , so the relation with $Ax = \lambda Bx$ is partially lost, and accurately computing eigenpairs of $Ax = \lambda Bx$ may be difficult. To see which parameters play a role in this perturbation, from Theorems 4.3 and 4.4 in [323], it can be shown that if T_{IC} has distinct eigenvalues, then for each eigenpair (ζ, x) of T_{C} there is an eigenpair (η, u) of T_{IC} such that

$$\begin{aligned} |\zeta - \eta| &\leq \kappa_1 |\zeta| \|E_{k-1}\|, \\ \|(I - uu^*/(u^* u))x\| &\leq \kappa_2 |\zeta| \|E_{k-1}\|, \end{aligned}$$

where $\kappa_1, \kappa_2 > 0$. These inequalities show that if $\|E_{k-1}\|$ and/or ζ are small, (ζ, x) and (η, u) correspond very well. In this case, (ζ, x) can be computed via eigenpairs of T_{IC} . Since $\zeta = (\lambda - \mu)^{-1}(\lambda - \nu)$, $\zeta \approx 0$, is reduced to $\lambda \approx \nu$.

This section can be concluded as follows.

- The inexact Cayley transform behaves like the exact one. This implies that eigenvalues near the pole μ are still targeted.
- However, the eigenvalues are perturbed. The level of perturbation depends on E_{k-1} . For any E_{k-1} , the transformation is able to compute eigenvalues near ν accurately. In practice, this means that only one (simple) eigenvalue can be computed very accurately. It also implies that ν needs to be updated from time to time, since λ is unknown. The other eigenvalues are only computed approximately depending on the error level $\|E_{k-1}\|$.
- The sequence (11.2) for $j = 1, \dots, k - 1$ defines a Krylov subspace $\mathcal{K}^k(T_{\text{IC}}, y_1)$.

The remaining question is now how to compute the eigenpairs of T_{IC} or how to exploit them for computing eigenpairs of $Ax = \lambda Bx$. In §11.2.3 and §11.2.4, we discuss the Rayleigh–Ritz technique; i.e., eigenpairs are computed from the orthogonal projection of $Ax = \lambda Bx$ on the $\text{span}(v_1, \dots, v_k)$. In §11.2.7, the eigenpairs are computed from the eigenpairs of T_{IC} directly by use of the rational Krylov recurrence relation. §11.2.6 presents a Lanczos algorithm that uses the recurrence relation for the eigenvectors and the Rayleigh–Ritz projection for the eigenvalues.

Note that μ and ν cannot be chosen too far away from each other. Suppose the eigenvalue λ is wanted. From the conclusion given above, the convergence is faster when μ is chosen close to λ , and the computed eigenvalue can only be accurate when ν is close to λ . In theory, one could select $\mu = \nu$, which is usually used in the Davidson method. Since $T_{\text{C}} = I$ in that case, there is a high risk of stagnation in the latter method. A robust selection is used in the Jacobi–Davidson method, which we discuss in §11.2.5.

11.2.3 Arnoldi Method with Inexact Cayley Transform

In this paragraph, we discuss the case of the inexact Cayley transform within a Galerkin projection framework. On each iteration, a Ritz value θ is chosen as zero. We restrict ourselves to the case where the linear system (11.2) is solved by a preconditioner M or a stationary linear system solver such that $w_j = M^{-1}(A - \theta B)y_j$ for $j = 1, \dots, k - 1$. Thus we can use the Arnoldi method to build the Krylov space of $T_{\text{IC}} \equiv M^{-1}(A - \theta B)$.

ALGORITHM 11.1: Arnoldi Method with Inexact Cayley Transform for GNHEP

- (1) given v_1 , $\|v_1\| = 1$ and choose $\nu \neq \mu$.
- (2) for $j = 1, \dots, k - 1$ do:
 - (3) solve $Mw_j = (A - \nu B)v_j$.
 - (4) orthonormalize w_j against v_1, \dots, v_j into v_{j+1} .
 - (5) compute the projected eigenpairs from

$$V_k^* AV_k z = \theta V_k^* BV_k z.$$
 - (6) select a Ritz pair $(\theta, x = V_k z)$.
 - (7) convergence check: if $\|Ax - \theta Bx\| \leq TOL$, stop
 - (8) select $v_1 = x$, $\nu = \theta$ and restart at step (2)

The disadvantage of this method is that an update of θ requires a complete restart of the Arnoldi process.

The asymptotic convergence towards, e.g., λ_1 , is governed by the separation of the eigenvalues of $T_{IC} = M^{-1}(A - \lambda_1 B) \approx (A - \mu B)^{-1}(A - \lambda_1 B)$. As for the SI, the separation improves when the pole μ is closer to λ_1 .

The separation of the eigenvalues of the exact transformation improves when μ is very close to the Ritz value θ , but the corresponding linear systems are more difficult to solve than for a μ a bit away from the spectrum. In [323], the linear systems are solved with preconditioners such as incomplete factorizations and multigrid. The numerical examples show faster convergence of Algorithm 11.1 when μ lies a bit away from the spectrum.

11.2.4 Davidson Method

As mentioned, Algorithm 11.1 does not allow updates of the zero of the Cayley transform without a complete restart. We now describe the Davidson method, which by contrast updates the zero in each iteration.

The Davidson method [99] was developed to solve quantum chemistry problems in which the matrices have diagonal elements that are both large and varying in magnitude. It was originally derived as a perturbation method. Given an approximate eigenvector, a correction is developed. In [335], the Davidson method was viewed as a diagonal preconditioning method and was generalized for other preconditioners. We now give an algorithm. The vector w_j is a correction to the approximate eigenvector y . The preconditioner is left unspecified, but for Davidson's original method, the choice is $M = (D - \theta I)^{-1}$.

We next discuss the asymptotic convergence of the Davidson method. Observe that if θ and M were fixed, the Davidson method would develop a Krylov subspace generated with the operator $M^{-1}(A - \theta B)$. So once a Ritz pair begins to converge to an eigenpair, say (λ, x) , then the subspace generated from that point is approximately the same as a Krylov subspace generated by $T_{IC} \equiv M^{-1}(A - \lambda B)$. T_{IC} has the correct eigenvector, namely x , and the corresponding eigenvalue of T_{IC} is 0. If the other eigenvalues of T_{IC} are compressed around 1 by the preconditioning, then 0 will be well separated and convergence rapid.

An important question is whether in practice the spectrum of T_{IC} is really an improvement over that of the original eigenvalue problem. There is potential, because

ALGORITHM 11.2: Generalized Davidson Method for GNHEP

- (1) choose a preconditioner M which may either be fixed or changing.
begin with k orthonormal vectors v_1, v_2, \dots, v_k , and let $j = k$.
- (2) compute (y, θ) , the best approximation from $\text{span}\{v_1, v_2, \dots, v_j\}$
to the eigenpair of interest, using the Rayleigh–Ritz procedure.
- (3) find the residual vector for y , $r = (A - \theta B)y$. If $\|r\| \leq TOL$,
accept that eigenpair, otherwise continue.
- (4) compute $w_j = M^{-1}r$. Orthonormalize w_j against v_1, \dots, v_j
to form v_{j+1} . Let $j = j + 1$ and go to (2).

in the best case, where $M = (A - \mu B)$, we then have the exact Cayley transformation. Also, we know that in the iterative solution of linear equations, spectra are improved by preconditioning. On the other hand, preconditioning of eigenvalue problems is probably tougher than for linear equations, because asymptotically the singular matrix $(A - \lambda B)$ is being approximated. This matrix is also indefinite if λ is not an exterior eigenvalue. We will look at two examples of how preconditioning can work for eigenvalue problems.

Example 11.2.1. Here A is symmetric and roughly models matrices from quantum chemistry. B is the identity matrix. The main diagonal of A has elements $1, 2, 3, \dots, 100$ and the off-diagonal elements of the upper triangular portion are selected randomly from the interval $(-1, 1)$. We consider computing the smallest eigenvalue of A using the diagonal preconditioning of the original Davidson method. The eigenvalues of A are $-0.323, 0.721, 1.73, 2.77, \dots, 101.58$. So we let $\lambda_1 = -0.323$. The eigenvalues of $T_{IC} = (D - \lambda_1 I)^{-1}(A - \lambda_1 I)$ are $0.0, 0.263, 0.387, 0.500, \dots, 2.01$. The eigenvalue 0 of T_{IC} is much better separated relative to the entire spectrum than is λ_1 of A . In fact, the gap ratio of λ_1 for A is $\frac{\lambda_2 - \lambda_1}{\lambda_n - \lambda_2} = 0.0094$, while the gap ratio for the eigenvalue 0.0 of T_{IC} is 0.151 . With gap ratio 16 times larger, asymptotic convergence is very roughly four times faster. The next example looks at a matrix that is tougher to precondition. Some results can be given similar to those for preconditioning linear equations.

Example 11.2.2. Consider the model problem from finite difference discretization of Poisson's equation on the unit square. The main diagonal is constant (all 4's), so diagonal preconditioning is ineffective. It merely shifts and scales the spectrum, without changing the relative separation of the eigenvalues. However, incomplete factorization techniques [326] have been shown effective for preconditioning the corresponding linear equations, and some theoretical results have been proved. With no preconditioning, the conjugate gradient method converges in $O(n^{1/2})$ iterations. With incomplete factorization IC(0), convergence is faster, but still approximately $O(n^{1/2})$. However with MIC(0), modified incomplete factorization [212], convergence is $O(n^{1/4})$. It is shown in [330] that the same holds true for computing the smallest eigenvalue. The modified incomplete factorization is of A , not $A - \theta I$. The theoretical result can be backed up by computations. Table 11.1 gives numbers of iterations as the problem size increases. The Lanczos algorithm (corresponding to both no preconditioning and diagonal preconditioning) and IC(0) and MIC(0) preconditionings are used. The starting vector has random values from the interval $(-1, 1)$, and the convergence test is residual norm less than 10^{-8} . No restarting is used. For large problems, MIC(0) preconditioning is

Table 11.1: Preconditioning the model problem

n	Lanczos	IC(0)	MIC(0)
25	13	11	12
49	25	14	15
100	36	17	17
196	51	24	20
400	69	27	22
784	99	40	26
1600	137	49	30
3136	182	64	34
6400	287	101	40

much better. In a log-log plot, the slopes are roughly 0.56 for Lanczos and 0.43 for IC(0). So both are near to the expected 0.5, although IC(0) is significantly better. With MIC(0) preconditioning, the slope is about 0.23, which is near the value of 0.25 predicted by the $O(n^{1/4})$ convergence result.

For computing several eigenvalues, MIC is not as good. We look at computing the smallest five eigenvalues with $n = 1600$. Again the preconditioners are from incomplete factorizations of A , with no attempt to adjust for the particular eigenvalue being computed. This time we restart the Davidson methods, with maximum subspaces of dimension 20, but do not restart Lanczos (if eigenvectors are desired, Lanczos might also need to be restarted). Relatively speaking, Lanczos does better at computing several eigenvalues, because the Davidson method targets one eigenvalue at a time. IC(0) uses 159 iterations, MIC(0) requires 158, and Lanczos 172. However, it should be noted that Lanczos misses the fact that one of the small eigenvalues is double, while the Davidson approaches do compute two eigenvalues at that point.

For a sparse matrix such as in Example 11.2.2, more than just the number of iterations must be considered. The Lanczos algorithm can be much cheaper per iteration than the Davidson approach. This motivates the development of the preconditioned Lanczos method which takes advantage of preconditioning while retaining much of the efficiency of the Lanczos approach. This will be discussed in §11.2.6. Efficiency issues also lead to connections with the Jacobi–Davidson method. One way to reduce the number of iterations required by the Davidson method and thus reduce the overhead for the Rayleigh–Ritz procedure is to develop an extremely good preconditioner. An iterative method can be used to approximately solve $(A - \theta B)w_j = r$ for the new vector w_j in step (3) of Algorithm 11.2. And this solution can be made as accurate as is desired. In the symmetric case, the conjugate gradient method can be used for the iterative method, and it can be preconditioned itself. As with the preconditioned Lanczos method, we then have an inner loop that is efficient in terms of computations. But it is also efficient in terms of storage requirements. There is, however, danger in developing a preconditioner that is too accurate. Exact solution of $(A - \theta B)w_j = r$ gives $w_j = y$, the approximate eigenvector which is already in the subspace. This could be dealt with by instead solving $(A - \mu B)w = r$, with $\mu \neq \theta$, as in the inexact Cayley transform. However, in the Jacobi–Davidson method a different approach is taken; the approximate eigenvector is deflated from the solution of $(A - \theta B)w_j = r$. Discussion of connections with Jacobi–Davidson is continued in the next subsection.

Davidson's method can have difficulty when the starting vector or starting subspace is not very accurate. In [335] and [172], it is suggested that initially M be set to $P - \mu I$, where P is an approximation to A and μ is near the desired eigenvalues. As mentioned in §11.1, another possibility is to use a Krylov subspace method in an initial phase to generate starting vectors for the Davidson method.

11.2.5 Jacobi–Davidson Method with Cayley Transform

The Jacobi–Davidson method is discussed in §7.12. The Jacobi–Davidson method differs from the Davidson method in that the linear system to be solved is projected onto the space orthogonal to the current Ritz vector. This leads to the solution of the correction equation

$$\left(I - \frac{Bx_j x_j^*}{x_j^* B x_j} \right) (A - \theta_j B) \left(I - \frac{x_j x_j^*}{x_j^* x_j} \right) w_j = (A - \theta_j B) x_j, \quad (11.5)$$

where (θ_j, x_j) is a Ritz pair on iteration j . (Note that usually, a minus sign is put in front of the residual in the right-hand side.) Assume that w_j is orthogonal to x_j , i.e., $(I - x_j x_j^* / x_j^* x_j) w_j = w_j$. When an inexact solver is used, we have a residual s_j that satisfies

$$s_j = (A - \theta_j B) x_j - \left(I - \frac{Bx_j x_j^*}{x_j^* B x_j} \right) (A - \theta_j B) w_j.$$

Note that the projection in front of w_j is dropped since w_j is assumed orthogonal to x_j . We can rewrite this into

$$\begin{aligned} (A - \theta_j B) w_j &= -\epsilon_j B x_j + (A - \theta_j B) x_j - s_j \\ &= (A - (\theta_j + \epsilon_j) B) x_j - s_j, \end{aligned}$$

where $\epsilon_j = -(x_j^*(A - \theta_j B) w_j) / (x_j^* B x_j)$.

In words, the solution w_j of the correction equation is obtained by the action of the Cayley transform $T_C = (A - \theta_j B)^{-1}(A - (\theta_j + \epsilon_j) B)$ to the most recent Ritz vector. Example 11.2.1 in [411] shows that ϵ_j tends to zero on convergence. Both pole and zero of the Cayley transform lie close to the desired eigenvalue. This meets the conditions for good matching between eigenvectors of $Ax = \lambda Bx$ and $T_C u = \eta u$, motivated at the end of §11.2.2.

The following observation is a bit funny. Since $w_j = -\epsilon_j(A - \theta_j B)^{-1}Bx_j + x_j$, when $s_j = 0$, we have from $x_j^* w_j = 0$ that

$$\theta_j + \epsilon_j = \theta_j + \frac{x_j^* x_j}{x_j^*(A - \theta_j B)^{-1}Bx_j}.$$

So, the pole of the Cayley transform is the Rayleigh–Ritz quotient of x_j and the zero is the harmonic Rayleigh–Ritz quotient with target θ_j .

11.2.6 Preconditioned Lanczos Method

Suppose that A and B are symmetric and that we have a positive definite preconditioner M for $A - \mu B$, i.e., $M \approx A - \mu B$. We can use the Arnoldi method applied to $M^{-1}(A -$

νB) for computing eigenvalues near μ . (Again, we use a Ritz value θ as zero.) This leads to the recurrence relation

$$M^{-1}(A - \nu B)V_k = V_k H_k + f_k e_k^T.$$

Since M is positive definite, we have that $x^* M y$ is a valid inner product. When we use the M inner product, $x^* M y$, in the Arnoldi method rather than the standard one, $x^* y$, we have $V_k^* M V_k = I$ and $V_k^* M f_k = 0$, so $H_k = V_k^*(A - \nu B)V_k$ is a symmetric matrix. This implies that the Arnoldi method with the M inner product reduces to the Lanczos method with M inner product applied to the nonsymmetric matrix $M^{-1}(A - \nu B)$. From earlier discussions, we conclude that the preconditioned Lanczos method with M inner product has similar convergence behavior as the spectral transformation Lanczos method applied to a perturbed problem.

Let (ζ, z) satisfy $H_k z = \zeta z$ and define the Ritz vector $x = V_k z$. This vector is obtained from the Lanczos (Arnoldi) recurrence relation, so not from a Galerkin projection. The Ritz value, on the other hand, can be computed via the Rayleigh quotient, i.e.,

$$\begin{aligned}\theta &= x^* A x / x^* B x \\ &= \nu + x^*(A - \nu B)x / x^* B x \\ &= \nu + z^* V_k^*(A - \nu B)V_k z / x^* B x \\ &= \nu + z^* H_k z / x^* B x \\ &= \nu + \zeta / x^* B x,\end{aligned}$$

which is cheap to compute, e.g. when B is a diagonal matrix.

Recall that the Lanczos recurrence relation for the inexact Cayley transform $T_{\text{IC}} = M^{-1}(A - \nu B)$ is

$$T_{\text{IC}} V_k - V_k H_k = f_k e_k^T.$$

The recurrence relation can also be written in the form

$$T_C V_k - V_k H_k = f_k e_k^T + (T_C - T_{\text{IC}})V_k.$$

We know that for a Ritz pair (x, θ) , $(T_C - T_{\text{IC}})x$ can be small, when $\theta \approx \nu$. So, if x is close to an eigenvector of T_{IC} and if θ is close to ν , then x is close to an eigenvector of T_C and so the Ritz value θ is close to an eigenvalue of $Ax = \lambda Bx$.

Morgan and Scott [336] proved the convergence for the following algorithm, for computing eigenvalues of $Ax = \lambda x$, i.e., with $B = I$.

ALGORITHM 11.3: Preconditioned Lanczos Method for GHEP

- (1) given v_1 , $\|v_1\| = 1$ and $\nu_1 = v_1^* A v_1 / v_1^* B v_1$
- (2) for $l = 1, 2, \dots$ do
 - (3) choose a preconditioner M_l for $A - \mu_l B$.
 - (4) run k steps of the Lanczos method with M_l orthogonalization applied to $T_{\text{IC}}^{(l)} = M_l^{-1}(A - \nu_l B)$.
 - (5) compute a Ritz vector $x_l = V_k z_l$ with $H_k z_l = \zeta_l z_l$, and ζ_l the smallest Ritz value of H_k .
 - (6) compute $\nu_{l+1} = \theta_l = \nu_l + \zeta_l / x_l^* B x_l$.
 - (7) if $\|Ax_l - \theta_l Bx_l\| \leq \text{TOL}$, stop

Note that if $\mu = \nu$, then the preconditioner must not be a too-good approximation to $A - \mu B$; otherwise $T_{\text{IC}} \approx I$. The value of k may differ for each l . Morgan and Scott suggest the stopping test $-\zeta_l > \|T_{\text{IC}}^{(l)}x_l - \zeta_l x_l\|$, which is cheap within the Lanczos method. It is shown in [336] that if both M_l and M_l^{-1} are uniformly bounded in norm, θ_l converges to an eigenvalue of A . Moreover, the asymptotic convergence is quadratic.

11.2.7 Inexact Rational Krylov Method

In this section, we combine the ideas of the inexact Cayley transformation, the (Jacobi) Davidson method, and the rational Krylov method. Roughly speaking, we still have the same method as the inexact Cayley transform Arnoldi method or the preconditioned Lanczos method, with the only difference that the zero ν and the pole μ may be updated on each iteration. As with the preconditioned Lanczos method, the Ritz vectors are computed from the Hessenberg matrices. In addition, the Ritz values are also computed from the recurrence relation.

We now discuss the method in detail, using the algorithm of the rational Krylov method, designed for the inexact Cayley transforms, given below.

ALGORITHM 11.4: Inexact Rational Krylov Method for GNHEP

- (1) given v_1 with $\|v_1\| = 1$, and a zero θ_0 .
let the residual be $r_0 = Av_1 - \theta_0 Bv_1$.
- (2) **for** $j = 1, 2, \dots$ **do**
- (3) choose a pole μ_j , let $\nu_j = \theta_{j-1}$, and solve w_j from:

$$(A - \mu_j B)w_j = r_{j-1}$$
- (4) orthonormalize w_j against v_1, \dots, v_j into v_{j+1} . Keep the
Gram–Schmidt coefficients in the j th column h_j of the
Hessenberg matrix $H_{j+1,j}$
- (5) build the j th column of $L_{j+1,j}$ and $K_{j+1,j}$:

$$\left(h_j - \begin{bmatrix} t_j \\ 0 \end{bmatrix} \right), \left(h_j \mu_j - \begin{bmatrix} t_j \\ 0 \end{bmatrix} \nu_j \right)$$

with $t_1 = 1$ and $t_j = L_{j,j-1} z_{j-1}$ for $j > 1$.
- (6) solve the $j \times j$ eigenvalue problem

$$L_{j+1,j}^\dagger K_{j+1,j} z_j = \theta_j z_j$$

and compute the Ritz pair $(\theta_j, x_j = V_{j+1} L_{j+1,j} z_j)$.
- (7) compute the residual $r_j = Ax_j - \theta_j Bx_j$
- (8) convergence check

Let us analyze this algorithm step by step. The solution of the linear system leads to the relations (11.2) where $y_j = x_{j-1}$ is a Ritz vector and ν_j is the associated Ritz value θ_{j-1} from the previous iteration, such that the right-hand side is the residual $r_{j-1} = Ax_{j-1} - \theta_{j-1} Bx_{j-1}$. Since $x_{j-1} = V_j L_{j,j-1} z_{j-1}$, we can also write $x_{j-1} = V_j t_j$, where t_j is called the continuation vector. After the Gram–Schmidt orthogonalization, we have $w_j = V_{j+1} h_j$, so we can rewrite (11.2) as

$$AV_{j+1} \left(h_j - \begin{bmatrix} t_j \\ 0 \end{bmatrix} \right) = BV_{j+1} \left(h_j \mu_j - \begin{bmatrix} t_j \\ 0 \end{bmatrix} \nu_j \right) - s_j.$$

Collecting all equations for $j = 1, \dots, k$ we have

$$AV_{k+1}L_{k+1,k} = BV_{k+1}K_{k+1,k} - S_k , \quad (11.6)$$

which is another way of writing (11.3). Note that when S_k is omitted from this equation, we obtain the usual rational Krylov subspace (RKS) recurrence relation. Also in this case, $L_{k+1,k}$ and $K_{k+1,k}$ are upper Hessenberg matrices and $L_{k+1,k}$ has full rank. Let $L_{k+1,k}^\dagger$ denote the generalized Moore–Penrose inverse of $L_{k+1,k}$. With $E_k = S_k L_{k+1,k}^\dagger V_{k+1}^*$, we can also write

$$(A + E_k)V_{k+1}L_{k+1,k} = BV_{k+1}K_{k+1,k}, \quad (11.7)$$

which corresponds to (11.4). In other words, the RKS coefficient matrices $L_{k+1,k}$ and $K_{k+1,k}$ and the vectors V_{k+1} can be considered as having been computed by the exact rational Krylov method applied to the perturbed problem $(A + E_k)u = \eta Bu$. Similar to the term “inexact” Cayley transform, we talk about “inexact” rational Krylov method.

Before we continue, we must give some properties of the exact rational Krylov method. The following lemma explains how to compute Ritz values in the rational Krylov method.

Lemma 11.1 *Let V_{k+1} , $L_{k+1,k}$, and $K_{k+1,k}$ be such that V_{k+1} and $L_{k+1,k}$ are of full rank. Let $AV_{k+1}L_{k+1,k} = BV_{k+1}K_{k+1,k}$ be satisfied. Assume that B is invertible. Then (θ, x) is a Ritz pair of $B^{-1}Ax = \lambda x$ with respect to the Range($V_{k+1}L_{k+1,k}$), iff $x_k = V_{k+1}L_{k+1,k}z_k$ and*

$$L_{k+1,k}^\dagger K_{k+1,k}z = \theta z .$$

Moreover, the residual is $Ax - \theta Bx = f_k$ with

$$f_k = BV_{k+1}(K_{k+1,k} - \theta L_{k+1,k})z .$$

As with the (Jacobi) Davidson method, the RKS method applies an inexact Cayley transform to a vector. The difference lies in the way the Ritz pairs are computed. In the (Jacobi) Davidson method, the Ritz pairs result from a Galerkin projection of $Ax = \lambda Bx$ on the subspace. In the RKS method, the Ritz pairs are computed from the recurrence relation using Lemma 11.1, assuming that $E_k = 0$.

With the inexact Cayley transform, the transformation can be a large perturbation of the exact Cayley transform, but can still be used to compute one eigenpair, when ν is well chosen. The same is true here. The inexact rational Krylov method delivers an S_k (or E_k) that is not random but small in the direction of the desired eigenpair, as long as the various parameters are properly set.

A Ritz pair (θ, x) computed by Algorithm 11.4 produces residual

$$(A + E_k)x - \theta Bx = f_k$$

defined by Lemma 11.1. The residual related to the original problem can be written as

$$Ax - \theta Bx = f_k - E_k x$$

or, following (11.6),

$$Ax - \theta Bx = f_k - S_k z .$$

In order to have small residuals, $E_k x = S_k z$ must tend to zero. Mathematical arguments for the convergence of the inexact rational Krylov method have been given by

Lehoucq and Meerbergen [291] and De Samblanx [109]. We just give a summary of the theory and some numerical results. In [291] and [109], mathematical and heuristical arguments are given that $S_j z_j \approx s_j$ such that, with $\|s_j\| \leq \tau \|r_{j-1}\|$ (see §11.2.2), the residual satisfies the recurrence relation

$$\|r_j\| \approx \|f_j\| + \tau \|r_{j-1}\|.$$

When the term f_j is dominant to $S_j z_j$ for $j = 1, \dots, k$, then the convergence is dominated by the convergence of the exact rational Krylov process: $Ax - \theta Bx \approx f_j$. When the second term is dominant, we have $\|r_j\| \approx \tau^j \|r_0\|$; i.e., r_j converges towards zero with convergence ratio τ .

- If the underlying exact rational Krylov method converges linearly with a convergence ratio ρ such that $\|f_{k+1}\| \leq \rho \|f_k\|$ and if a relative residual tolerance τ is used for the solution of the linear systems, the effective convergence ratio is approximately $\max(\rho, \tau)$. Linear and superlinear convergence are obtained when μ_j is constant for all j .
- When $\mu_j = \theta_{j-1}$ on each iteration as in the Davidson or Jacobi–Davidson method and $\nu_j = \theta_{j-1} + \epsilon_{j-1}$ (see §11.2.5), then the convergence is quadratic. The term f_j converges quadratically to zero, whereas the second term converges linearly, when τ does not depend on j . Typically, the convergence has a quadratic behavior until $S_j z_j$ becomes the dominant term in r_j . From then on, the convergence becomes linear with convergence ratio τ . In practice, it is advised to pick τ proportional to $\|r_j\|$ in order to preserve the quadratic convergence. With $\tau_j = \varrho \|r_{j-1}\|$, we obtain that

$$\|r_j\| \approx \|f_j\| + \varrho \|r_{j-1}\|^2,$$

which shows quadratic convergence of the second term.

Example 11.2.3. This is now illustrated by the following numerical example. The Olmstead model [343] represents the flow of a layer of viscoelastic fluid heated from below. The equations are

$$\begin{aligned} \frac{\partial u}{\partial t} &= (1 - C) \frac{\partial^2 v}{\partial X^2} + C \frac{\partial^2 u}{\partial X^2} + Ru - u^3, \\ B \frac{\partial v}{\partial t} &= u - v, \end{aligned}$$

where u represents the speed of the fluid and v is related to viscoelastic forces. The boundary conditions are $u(0) = u(1) = 0$ and $v(0) = v(1) = 0$. After discretization with central differences with grid size $h = 1/(n/2)$, the equations may be written as $\dot{x} = f(x)$ with $x = [u_1, v_1, u_2, v_2, \dots, u_{n/2}, v_{n/2}]^*$. We consider the Jacobian matrix $A = \partial f / \partial x$ for $n = 1000$ with the parameter values $B = 2$, $C = 0.1$, and $R = 4.7$, evaluated in the trivial steady state solution.

We solved linear systems by GMRES with 30 iteration vectors. The method was restarted by Morgan's implicitly restarted GMRES [334] keeping the 15 smallest Ritz pairs in the basis until the residual norm satisfied $\|s_j\| \leq \tau \|r_{j-1}\|$ with $\tau = 1 \cdot 10^{-3}$. We used Algorithm 11.4 for computing the eigenvalue nearest 5. We consider two cases. For Case 1, we used a fixed $\mu = 5$. For Case 2, we used two steps with $\mu = 5$ and the

Table 11.2: Residual norms of the inexact rational Krylov method for the Olmstead problem

j	Case 1		Case 2	
	$\ r_j\ $	$\ f_j\ $	$\ r_j\ $	$\ f_j\ $
1	1.3×10^0	1.3×10^0	1.3×10^0	1.3×10^0
2	1.8×10^{-1}	1.8×10^{-1}	1.8×10^{-1}	1.8×10^{-1}
3	8.6×10^{-3}	8.5×10^{-3}	8.6×10^{-3}	8.6×10^{-3}
4	9.8×10^{-4}	9.9×10^{-4}	5.0×10^{-6}	1.7×10^{-6}
5	5.0×10^{-5}	5.0×10^{-5}	3.7×10^{-9}	1.0×10^{-13}
6	1.1×10^{-5}	1.1×10^{-5}	1.8×10^{-11}	2.1×10^{-20}
7	9.3×10^{-7}	9.4×10^{-7}		
8	6.2×10^{-8}	6.3×10^{-8}		
9	2.0×10^{-9}	1.9×10^{-9}		
10	1.2×10^{-10}	1.3×10^{-10}		

remaining steps solved the correction equation (11.5). From the results in Table 11.2 and Figure 11.1, we can see linear convergence for Case 1. Since the linear systems are solved more accurately ($\tau = 10^{-3}$) than the speed of the eigenvalue solver, f_j and r_j converge to zero with the same speed. For Case 2, the convergence is linear for Steps 1 and 2, since μ_j is constant. From the third iteration on, f_j converges quadratically to zero. The $\|r_j\|$ converge linearly in steps 1 and 2, then we have quadratic convergence on steps 3 and 4, and then linear convergence again with convergence ratio $\tau = 10^{-3}$.

The residual norm r_j has two terms. The first term decreases very rapidly when μ is close to an eigenvalue. The decrease of the second term (the residual of the linear system solver) is often much more difficult when the pole lies close to the spectrum. In practice, we may need to balance the number of outer iterations (the eigenvalue solver) and the number of inner iterations (the linear system solver) by selecting the most optimal μ . This comment is also related to the end of §11.2.3.

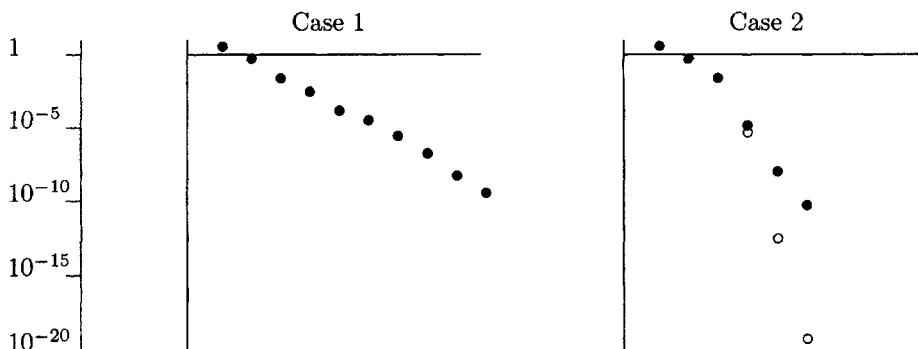


Figure 11.1: Logarithmic plots of residual norms of the inexact rational Krylov method for the Olmstead problem. The circles denote $\|f_j\|$ and the bullets $\|r_j\|$.

11.2.8 Inexact Shift-and-Invert

Recall from §11.2.1 that with exact arithmetic and exact linear system solvers, the SI and the Cayley transform produce the same Krylov spaces and the same Ritz pairs. We have shown in the previous sections by theoretical arguments and numerical examples that the Cayley transform is a good choice when the linear systems are solved with a relative error tolerance τ smaller than 1.

When the Cayley transform is used, the linear system

$$(A - \mu B)w = (A - \theta B)x - s$$

is solved, where $\|s\| \leq \tau \| (A - \theta B)x \|$. This linear system can be written as

$$(A - \mu B)(w - x) = (\mu - \theta)Bx - s.$$

So, with $z = (w - x)/(\mu - \theta)$ and $t = s/(\mu - \theta)$, we obtain the shift-and-invert linear system

$$(A - \mu B)z = Bx - t,$$

where $\|t\| \leq \tau \| (A - \theta B)x \| / (\mu - \theta)$. In other words, we can use the SI, but instead of a relative residual tolerance, we should use a tolerance that is proportional to the norm of the Ritz residual, $\| (A - \theta B)x \|$.

11.3 Preconditioned Eigensolvers

A. Knyazev

11.3.1 Introduction

We consider a generalized *symmetric definite* eigenvalue problem of the form

$$(A - \lambda B)x = 0$$

with real symmetric n by n matrices A and B , assuming that A is positive definite. That describes a regular matrix pencil $A - \lambda B$ with a discrete spectrum, a set of n real, some possibly infinite, eigenvalues $\lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$. If B is nonsingular, all eigenvalues are finite. If B is positive semidefinite (e.g., $B = I$), all eigenvalues are positive, and we consider the problem of computing the smallest m eigenvalues of the pencil $A - \lambda B$. When B is indefinite, it is convenient to consider the pencil $B - \mu A$ with eigenvalues

$$\mu = \frac{1}{\lambda}, \quad \mu_{\min} = \mu_n \leq \dots \leq \mu_1 = \mu_{\max},$$

where we want to compute the largest m eigenvalues, μ_1, \dots, μ_m . The problem of computing m smallest eigenvalues μ_i can be reformulated as the previous one by replacing B with $-B$. In buckling, both largest and smallest eigenvalues μ_i are of interest.

It is well known that (right) eigenvectors x_i , satisfying $(B - \mu_i A)x_i = 0$, can be chosen orthogonal in the following sense:

$$(x_i, Ax_j) = (x_i, Bx_j) = 0, \quad i \neq j.$$

An important class of eigenproblems is the class of *mesh eigenproblems*, arising from discretizations of boundary value problems with self-adjoint differential operators of mathematical physics. The following properties are typical for mesh eigenproblems:

- The size n of A and B is big, and is much larger than the number m of required eigenpairs. Sometimes, a family of similar eigenproblems with increasing n must be solved.
- A and B are sparse. Very large matrices cannot always fit in available memory even when using a sparse representation. In some situations, matrices A and B are not available in an explicit matrix form.
- No cheap factorization of A , or any matrix of the form $A + \beta B$, is available. The only operation we can perform with A is multiplication of a vector by A .
- Cheap factorization of B may sometimes be available, but, in general, the only operation we can do efficiently with B is multiplication of a vector by B .
- The matrix A is ill-conditioned. Usually, B has a smaller condition number than A .
- The number of wanted eigenpairs m can be chosen such that eigenvalues of interest are well separated from the rest of the spectrum; in other words, the ratio

$$\frac{\mu_m - \mu_{m+1}}{\mu_{\max} - \mu_{\min}}$$

is not too small. This means that the invariant subspace of the matrix $A^{-1}B$ spanned by the eigenvectors of interest is well conditioned, thus the problem of computing m eigenpairs is well posed. Such property can be established in many applications when the spectrum of the pencil $B - \mu A$ approximates a spectrum of a compact operator, such that zero is the only point of condensation of the approximate eigenvalues.

Such problems appear, e.g., in structural mechanics, where it is usual to call A a *stiffness* matrix and B a *mass* matrix. A mass matrix is usually positive definite, but in some applications, e.g., in buckling, the matrix B is only nonnegative, or even indefinite, while A is positive definite.

In the rest of the section, for brevity we deal with the pencil $B - \mu A$ only. With $B = I$ and $\lambda = 1/\mu$, our results and arguments are readily applied for the pencil $A - \lambda I$.

Now, we briefly consider two traditional methods for such generalized eigenproblems.

One popular approach is based on multiplication by a shift-and-invert operator:

$$x^{(i+1)} = (B - \mu A)^{-1} A x^{(i)};$$

see §4.3. It lets us quickly compute the eigenvalues closest to the shift μ , assuming that this operation may be implemented with an explicit efficient triangular factorization of $B - \mu A$. With a properly chosen variable shift, e.g., based on the Rayleigh quotient

$$\mu(x) = \frac{(x, Bx)}{(x, Ax)}, \quad (11.8)$$

the convergence is cubic for symmetric eigenproblems. However, for very large problems such factorizations are usually too expensive. Instead of explicit factorization, an approximate solution of the corresponding linear system is possible, e.g., using an

iterative system solver. Then we obtain an inner-outer iterative method, where outer iterations may slow down significantly for an insufficient number of inner steps. If we take into account the total number of inner-outer iterative steps, the convergence is usually only linear.

If a preconditioned iterative solver is used for inner iterations, such a method becomes a *preconditioned eigensolver*. We shall discuss this method later, in §11.3.7.

Another approach can be used if B is efficiently factorizable, so that we can multiply vectors by AB^{-1} , or $B^{-1}A$, and use traditional methods like the Lanczos method; see §4.4. In this case, a single iteration is not too expensive, but the convergence may be slow. If B is indefinite, then we need to find eigenvalues λ_i in the middle of the spectrum using polynomial methods, which is known to be a hard problem. If B is positive definite, then the situation is simpler, as we want to find the extreme (smallest) eigenvalues λ_i . Still, the ratio

$$\frac{\lambda_{m+1} - \lambda_m}{\lambda_{\max} - \lambda_{\min}}$$

that typically characterizes the speed of convergence for λ_m is small because of the large denominator, which is a sign of possibly slow convergence.

Thus, the traditional approaches considered above are usually inefficient for very large eigenproblems. Preconditioning is a key for significant improvement of the performance.

11.3.2 General Framework of Preconditioning

Preconditioned methods are designed to handle the case when the only operation we can perform with matrices A and B of the pencil is multiplication of a vector by A and B . To accelerate the convergence, we introduce a *preconditioner* T . It is also common to call the inverse $T^{-1} = M$ the preconditioner; see, e.g., the previous section. Applying the preconditioner Tx to a vector x usually involves solving a linear system $Mu = f$.

In many engineering applications, preconditioned iterative solvers for linear systems $Ax = b$ are already available, and efficient preconditioners $T \approx A^{-1}$ are constructed. We shall show that the same preconditioner T can be used to solve an eigenvalue problem $Ax = \lambda x$ and $Bx = \mu Ax$. Moreover, existing codes for the system $Ax = b$ can often be just slightly modified to solve the partial eigenvalue problem with A .

We will assume that the preconditioner T is *symmetric positive definite*. As A is also symmetric positive definite, there exist positive constants $\delta_1 \geq \delta_0 > 0$ such that

$$\delta_0(Mx, x) \leq (Ax, x) \leq \delta_1(Mx, x), \quad M = T^{-1}. \quad (11.9)$$

The ratio δ_1/δ_0 can be viewed as the spectral condition number $\kappa(TA)$ of the preconditioned matrix TA and measures how well M approximates, up to a scaling, the original matrix A . A smaller ratio δ_1/δ_0 ensures faster convergence.

Indefinite preconditioners for symmetric eigenproblems are also possible, but not recommended. Iterative methods for nonsymmetric problems, e.g., based on minimization of the residual, should generally be used when the preconditioner is indefinite, which may increase computational costs considerably.

We will *not assume* that the preconditioner T commutes with A , or $A^{-1}B$, despite the fact that such an assumption would greatly simplify the theory of preconditioned methods.

We first define, following [268], a preconditioned single-vector iterative solver for the pencil $B - \mu A$, as a generalized polynomial method of the following kind:

$$x^{(k)} = P_{m_k}(TA, TB)x^{(0)}, \quad (11.10)$$

where P_{m_k} is a polynomial of the m_k th degree of two independent variables, $x^{(0)}$ is an initial guess, and T is a fixed preconditioner.

We only need to choose a polynomial, either a priori or during the process of iterations, and use a recursive formula which leads to an iterative scheme. For an approximation $\mu^{(i)}$ to an eigenvalue of the pencil $B - \mu A$ for a given eigenvector approximation $x^{(i)}$, the Rayleigh quotient $\mu(x)$ (11.8) is typically used:

$$\mu(x^{(i)}) = \frac{(x^{(i)}, Bx^{(i)})}{(x^{(i)}, Ax^{(i)})}.$$

Thus, we have a complete description of a general preconditioned eigensolver for the pencil $B - \mu A$, as shown below:

THE PRECONDITIONED EIGENSOLVER FOR $B - \mu A$

1. Start: select $x^{(0)}$.
2. Iterate m_k steps to compute $x^{(k)} = P_{m_k}(TA, TB)x^{(0)}$.
3. Compute $\mu^{(k)} = (x^{(k)}, Bx^{(k)})/(x^{(k)}, Ax^{(k)})$.

With $B = I$ and $\lambda = 1/\mu$, we can obtain a similar algorithm for $A - \lambda I$. The polynomials can be chosen in a special way to force convergence of $x^{(k)}$ to an eigenvector other than the one corresponding to an extreme eigenvalue.

Similarly, we define general preconditioned *block-iterative methods*, where a group of vectors $x_j^{(k)}$, $j = 1, \dots, m$, is computed simultaneously:

$$x_j^{(k)} = P_{m_k}(TA, TB)x_j^{(0)}, \quad j = 1, \dots, m, \quad (11.11)$$

where P_{m_k} is a polynomial of the m_k th degree of two independent variables, and $x_j^{(0)}$ are initial guesses. The polynomial P_{m_k} does not have to be (and in practice is usually not) the same for different values of j . If it is the same, then (11.11) can be rewritten as the *subspace iterations*:

$$S^{(k)} = P_{m_k}(TA, TB)S^{(0)}, \quad (11.12)$$

where $S^{(0)}$ and $S^{(k)}$ are m -dimensional subspaces. Such a method is easier to analyze theoretically; see an example of a preconditioned subspace iterations based on the power method with shift in [148, 149], but may converge slower in practice.

In (11.11), the iterative subspace $S^{(k)}$ is defined as a span

$$S^{(k)} = \text{span}\{x_1^{(k)}, \dots, x_m^{(k)}\}$$

of individual vectors $x_j^{(k)}$. It is common to use (11.11) recursively by combining it with a procedure for selecting individual vectors in $S^{(k)}$ as initial vectors for the next

recursive step. The Rayleigh–Ritz method is the usual choice for such a procedure. One step of the recursion is shown below:

THE BLOCK PRECONDITIONED EIGENSOLVER FOR $B - \mu A$

1. Start: select $x_j^{(0)}$, $j = 1, \dots, m$.
2. Iterate m_k steps to compute $\hat{x}_j^{(k)} = P_{m_k}(TA, TB)x_j^{(0)}$, $j = 1, \dots, m$.
3. Use the Rayleigh–Ritz method for the pencil $B - \mu A$ in the subspace $\text{span}\{\hat{x}_1^{(k)}, \dots, \hat{x}_m^{(k)}\}$, to compute the Ritz values $\mu_j^{(k)}$ and the corresponding Ritz vectors $x_j^{(k)}$.

In the following sections, we consider particular examples of preconditioned eigensolvers.

11.3.3 Preconditioned Shifted Power Method

The *preconditioned power method with a shift* is the simplest preconditioned iterative method. It can only find the smallest (or largest, for a different shift) eigenvalue and the corresponding eigenvector.

We present the single-vector version of the method for the pencil $B - \mu A$ in Algorithm 11.5. On the output, $\mu^{(k)}$ and $x^{(k)}$ approximate the largest eigenvalue μ_1 and its corresponding eigenvector.

We note that the shift τ requires knowledge of μ_{\min} , or its estimate from below. If B is nonnegative definite, then μ_{\min} may simply be replaced with 0.

ALGORITHM 11.5: Preconditioned Power Method for GHEP

- (1) select a starting vector $x^{(0)}$.
- (2) for $i = 0, \dots$, until convergence do:
 - (3) $\mu^{(i)} := (x^{(i)}, B x^{(i)}) / (x^{(i)}, A x^{(i)})$
 - (4) $r := B x^{(i)} - \mu^{(i)} A x^{(i)}$
 - (5) $w := Tr$
 - (6) $x := w + \tau x$, with the shift $\tau = \delta_1(\mu^{(i)} - \mu_{\min})$
 - (7) $x^{(i+1)} = x / \|x\|_A$

The standard arguments, based on the eigendecomposition of $A^{-1}B$ for the pencil $B - \mu A$, do not allow us to show that the power method converges, as eigenvectors in the eigendecomposition are not necessarily eigenvectors of the iteration operator. This makes convergence theory quite tricky. D'yakonov and his colleagues [150, 146, 147] obtained explicit estimates of linear convergence for iterative Algorithm 11.5 using assumption (11.9). Somewhat simplified (see [264, 265, 268]), the convergence rate estimate for Algorithm 11.5 can be written as

$$\frac{\mu_1 - \mu^{(n)}}{\mu^{(n)} - \mu_2} \leq (1 - \xi)^n \frac{\mu_1 - \mu^{(0)}}{\mu^{(0)} - \mu_2}, \quad \xi = \frac{\delta_0}{\delta_1} \frac{\mu_1 - \mu_2}{\mu_1 - \mu_{\min}}, \quad (11.13)$$

under the assumption that $\mu^{(0)} > \mu_2$.

The estimate shows that the convergence is (at least) linear. Note that condition numbers of A and B do not appear in the estimate.

Similar results can be obtained if $\mu_p \geq \mu^{(0)} > \mu_{p+1}$ for some $p > 1$ holds [147].

In numerical experiments, Algorithm 11.5 usually converges to μ_1 for a random initial guess. When $\mu_p \geq \mu^{(0)} > \mu_{p+1}$, the sequence $x^{(k)}$ needs to pass p eigenvectors, which are saddle points of the Rayleigh quotient, to reach u_1 . The convergence may slow down, in principle, near every saddle point. For a general preconditioner T , it is hard to predict whether this can happen for a given initial guess $x^{(0)}$.

We have collected the dominant costs per iteration for Algorithm 11.5 in terms of storage and floating point operations respectively, in the following two tables.

Item	Storage
Residual	1 n -vector
Approximate eigenvector	1 n -vector
Temporary vectors	1–2 n -vectors

Action	Major cost
Rayleigh quotient $\mu^{(i)}$	2 dots
Residual r	2 matrix-vector products
Preconditioned residual w	Depends on preconditioner T
Approximate eigenvector	1 update

The main advantage of the preconditioned shifted power method is, of course, its algorithmic simplicity and the low costs per iteration. Using an a priori chosen shift provides the total control of the iterative method. The method is very stable and robust. A solid convergence theory exists.

The need of bounds for δ_1 and λ_{\min} to calculate the shift is a clear disadvantage. Also, other preconditioned eigensolvers we consider below converge linearly as well, but typically with a better rate.

11.3.4 Preconditioned Steepest Ascent/Descent Methods

These methods are similar to Algorithm 11.5 in the previous subsection, but the shifts are chosen in the process of iterations and do not require knowledge of any bounds. Historically, preconditioned steepest descent/ascent methods for eigenproblems were suggested earlier (see [225, 393, 365]) than the power method with an a priori choice of the shift of the previous subsection.

Here, we present the single-vector version of the preconditioned steepest ascent for the pencil $B - \mu A$ in Algorithm 11.6. For $B = I$, $\lambda = 1/\mu$, the method becomes a steepest descent method for $A - \lambda I$.

We note that the shift can be determined with the Rayleigh–Ritz method for the pencil $B - \mu A$ on the trial subspace $\text{span}\{x^{(i)}, w\}$. As the trial space is two-dimensional, the projection problem can be solved as a quadratic equation. Thus, one can derive explicit formulas for τ as roots of a quadratic polynomial.

By construction, the steepest ascent (descent) method provides maximization (minimization) of the Rayleigh quotient on every iteration as compared with the corresponding iterative Algorithm 11.5. Therefore, the convergence results from the previous section can be applied without any changes, and similar observations can be made.

ALGORITHM 11.6: Preconditioned Steepest Ascent Method for GHEP

- (1) select a starting vector $x^{(0)}$.
- (2) for $i = 0, \dots$, until convergence do:
- (3) $\mu^{(i)} := (x^{(i)}, B x^{(i)}) / (x^{(i)}, A x^{(i)})$
- (4) $r := B x^{(i)} - \mu^{(i)} A x^{(i)}$
- (5) $w := Tr$
- (6) $x := w + \tau x^{(i)}$, with τ chosen to maximize $\mu(x)$
- (7) $x^{(i+1)} = x / \|x\|_A$

We have collected the dominant costs per iteration for Algorithm 11.6 in terms of storage, and floating point operations in the following two tables, respectively:

Item	Storage
Residual	1 n -vector
Approximate eigenvector	1 n -vector
Temporary vectors	3–5 n -vectors

Action	Major cost
Rayleigh–Ritz method	6 dots and 2 matrix-vector products
Residual r	2 matrix-vector products, 1 update
Preconditioned residual w	Depends on preconditioner T
Approximate eigenvector	1 update

The main advantages of the preconditioned steepest descent/ascent method are its relative simplicity and a minimal cost of every iteration compared to other preconditioned eigensolvers considered below. Using the Rayleigh–Ritz method instead of an a priori chosen shift does not require knowledge of any bounds. On the negative side, every iteration may cost about twice that of Algorithm 11.5. Though convergence is usually better than that of Algorithm 11.5 it is still worse than linear convergence of other preconditioned eigensolvers we consider below.

11.3.5 Preconditioned Lanczos Methods

The preconditioned Lanczos method was suggested by Scott [397], though with $T = I$, and was analyzed by Knyazev [264, 265]. It is based on the following idea of inner/outer iterations, which we describe here for the pencil $B - \mu A$.

Let parameter μ be fixed. We consider the following auxiliary eigenvalue problem:

$$T(B - \mu A)v = \nu v. \quad (11.14)$$

If $\mu = \mu_1$, then there exists a zero eigenvalue ν , and the corresponding eigenvector of (11.14) is also the eigenvector of the original pencil $B - \mu A$ corresponding to μ_1 . This zero eigenvalue is the maximal eigenvalue of the matrix $T(B - \mu A)$ and is separated from the rest of the spectrum of $T(B - \mu A)$, which is negative. Thus, it can be computed using standard polynomial methods applied to $T(B - \mu A)$, in particular, by the Lanczos method. We note that the separation, which determines the speed of convergence, depends on the quality of the preconditioner T ; see [264, 265].

In the method, we choose some initial value of parameter $\mu = \mu^{(0)}$, and then use a few Lanczos inner iterations to solve for the largest eigenvalue of (11.14). Under our assumption that the preconditioner T is symmetric positive definite, the matrix $T(B - \mu A)$ is symmetric with respect to the T^{-1} -based scalar product, therefore, the classical Lanczos method can be used with this scalar product; see §4.4. The new value of $\mu = \mu^{(1)}$ is then calculated as the Rayleigh quotient for the original pencil of the most recent vector iterate of the inner iterations. This vector also serves as an initial guess for the next cycle of inner iteration. Such inner/outer iterative method clearly fits our definition of preconditioned eigensolvers.

We present the single-vector version of the method for the pencil $B - \mu A$ in Algorithm 11.7.

ALGORITHM 11.7: Preconditioned Lanczos Method for GHEP

- (1) select a starting vector $x^{(0)}$.
- (2) for $i = 0, \dots$, until convergence do:
- (3) $\mu^{(i)} := (x^{(i)}, B x^{(i)}) / (x^{(i)}, A x^{(i)})$
- (4) apply l_i steps of the classical Lanczos method, with inner product $(x, T^{-1}y)$, to find the largest eigenvalue of $T(B - \mu^{(i)}A)$ and the corresponding eigenvector using vector $x^{(i)}$ as the initial guess.
- (5) $x^{(i+1)} :=$ the latest vector iterate of the Lanczos method

Asymptotic quadratic convergence of the outer iterations for a *fully converged* inner iteration process was proved by Scott [397]. This does not guarantee much, of course, for the convergence of the method with *limited* number of inner iterations. An explicit convergence rate estimate for an arbitrary number of inner iterations was established in [264, 265]. It shows that, first, *the method converges at least linearly for any fixed number of inner iterations, e.g., only even one*, and, second, *a slow, but unlimited, increase of the number of inner iterations during the process improves the convergence rate estimate, averaged with regard to the number of inner iterations*.

The preconditioned Lanczos method may converge faster than the preconditioned steepest descent/ascent method. Moreover, if the standard three-term recurrence without reorthogonalization is used in inner iterations of the Lanczos method, then the computational costs for each iteration are not much more than for the preconditioned steepest descent/ascent method.

The main disadvantage is that in the Lanczos process a T^{-1} -based scalar product is required, so that it should be easy to compute vectors $T^{-1}x$. For some preconditioners, e.g., domain decomposition-based ones, it may be possible to compute only Tx efficiently (which is also necessary), but not $T^{-1}x$.

In such a situation, however, we may still construct a Krylov subspace for the operator $T(B - \mu^{(i)}A)$ and project the operator $B - \mu A$ onto this subspace using the Rayleigh–Ritz procedure. Schematically:

$$\begin{aligned} \mu(x^{(i+1)}) &= \max_{x \in \mathcal{K}} \mu(x), \\ \mathcal{K} &= \text{span}\{x^{(i)}, T(B - \mu^{(i)}A)x^{(i)}, \dots, [T(B - \mu^{(i)}A)]^{l_i}x^{(i)}\}, \end{aligned} \quad (11.15)$$

where l_i is the number of inner iteration steps for the i th outer iteration step.

This leads to the method of Algorithm 11.8 that can be obtained from Algorithm 11.7 by modifying lines (4) and (5):

ALGORITHM 11.8: Preconditioned Projection Method for GHEP

- (1) select a starting vector $x^{(0)}$.
- (2) for $i = 0, \dots$, until convergence do:
- (3) $\mu^{(i)} := (x^{(i)}, B x^{(i)}) / (x^{(i)}, A x^{(i)})$
- (4) construct $(l_i + 1)$ -dimensional Krylov subspace using matrix $T(B - \mu^{(i)} A)$ and vector $x^{(i)}$ as the initial guess
- (4a) apply the Rayleigh–Ritz method for $B - \mu A$
- (5) $x^{(i+1)} :=$ Ritz vector, corresponding to the smallest Ritz value

The method was suggested in [264, 265] and then was rediscovered in [336]. If only one inner iteration is used, the method is identical to the preconditioned steepest descent/ascent Algorithm 11.6.

Because of the similarities with the preconditioned Lanczos process much of the convergence theory and the main conclusions thereof carry over to this process as well; for details see [264, 265]. The preconditioned projection Algorithm 11.8 converges somewhat faster than the preconditioned Lanczos Algorithm 11.7 with the same number of inner iterations. However, the standard three-term recurrence can now only be used to compute the basis of the Krylov subspace. To implement the Rayleigh–Ritz method we need to store all the vectors of the basis and explicitly compute all scalar products. This makes the cost of computing projection matrices in the Rayleigh–Ritz method higher and significantly increases storage requirements, unless restarts are used.

11.3.6 Davidson Method

In the Davidson method, an attempt is made to accelerate the convergence of the preconditioned projection Algorithm 11.8 by changing the Rayleigh quotient at each inner iteration step.

For simplicity, we describe here only the most naive version of the Davidson method, without restart, for the pencil $B - \mu A$ (cf. [99, 100, 335, 329, 387]):

$$\begin{aligned} \mu(x^{(i+1)}) &= \max_{x \in \mathcal{D}_i} \mu(x), \\ \mathcal{D}_i &= \text{span}\{x^{(0)}, T(B - \mu^{(0)} A)x^{(0)}, T(B - \mu^{(1)} A)x^{(1)}, \dots, T(B - \mu^{(i)} A)x^{(i)}\}. \end{aligned} \quad (11.16)$$

Since the dimension of the subspace \mathcal{D}_i grows with every step, and all basis vectors need to be stored, restarts are usually necessary as in preconditioned projection Algorithm 11.8. A trivial procedure of a restart is to stop the method after a certain number of steps (inner iterations) and then start it again using the last iteration vector as the initial approximation for the next step (of outer iterations).

The method was quite intricate to study theoretically; cf. [88, 390]. It is common to use the Davidson method with an indefinite preconditioner (often the diagonal of $B - \mu^{(0)} A$), which further complicates the theoretical analysis.

The Davidson method is popular particularly in the chemistry community (see [232, 275]), where the involved matrices are typically diagonally dominant. For such

problems, the method often converges quickly even with a simple diagonal preconditioner.

11.3.7 Methods with Preconditioned Inner Iterations

We note that there is no analog of the inverse power method in our class of preconditioned eigensolvers, as it would require exact solving linear systems with the matrix $B - \alpha A$. However, if such systems are solved using a preconditioned iterative method as inner iterations, then the method of inner/outer iterations falls into the class of preconditioned eigensolvers. As an example, we consider here the most straightforward version of the RQI, applied to the pencil $B - \mu A$; see §4.3.

ALGORITHM 11.9: RQI with Preconditioned Inner Iterative Solver for GHEP

- (1) select a starting vector $x^{(0)}$.
- (2) for $i = 0, \dots$, until convergence do:
 - (3) $\mu^{(i)} := (x^{(i)}, B x^{(i)}) / (x^{(i)}, A x^{(i)})$
 - (4) use l_i steps of a polynomial with respect to matrix $T(B - \mu^{(i)} A)$ solver with vector $x^{(i)}$ as the initial guess to find an approximate solution of the preconditioned linear system $T(B - \mu^{(i)} A)x = TAx^{(i)}$
 - (5) $x^{(i+1)} :=$ the last iteration vector of inner iterations.

It is common to use MINRES as a choice of the inner iterative solver, as the system matrix $T(B - \mu^{(i)} A)$ is indefinite. When only a fixed number of inner iterations is used, we cannot assume that the equation $(B - \mu^{(i)} A)x = Ax^{(i)}$ is solved any accurately, therefore, well-known convergence properties of the RQIs, like cubic convergence, cannot be used directly to establish convergence of the actual inner/outer iterative method. Similarly, the standard perturbation theory cannot be easily applied unless an assumption is made that the number of inner steps is large enough.

The matrix $B - \mu^{(i)} A$ of the equation is ill-conditioned not only because A is ill-conditioned, but also because $\mu^{(i)}$ is getting closer to an eigenvalue of the pencil $B - \mu A$. It is common to try to improve the condition number of the matrix by projecting out the subspace $\text{span}\{x^{(i)}\}$, using regularization or/and choosing an indefinite preconditioner T . It is not evident, however, whether an improvement of convergence of inner iterations is beneficial to convergence of the actual inner/outer iterative method with only a few inner steps.

There are several similar methods with analogous properties, e.g., different versions of Newton's method for finding eigenvectors as stationary vectors of the Rayleigh quotient (e.g., [461, 468]) or truncated rational Krylov methods (e.g., [378, 291]) or inexact homotopy methods (e.g., [309, 235, 469, 310]); see also the previous section.

The Jacobi–Davidson method [411], described in §5.6 for a generalized symmetric eigenproblem, is one of the most famous in this class. The inexact Jacobi–Davidson method with preconditioned system solver as inner iteration does satisfy our definition of a preconditioned eigensolver.

The convergence behavior of such methods with a small number of inner iterations is not well understood.

11.3.8 Preconditioned Conjugate Gradient Methods

In this subsection, we describe our favorite method, in terms of practical efficiency: the *locally optimal PCG method* [266]. It combines the robustness and simplicity of Algorithm 11.6, the three-term recurrence of the preconditioned Lanczos algorithm 11.7, and, perhaps, the fast convergence of the preconditioned projection algorithm 11.8 and Davidson's method.

A simple variant of a PCG method for the pencil $B - \mu A$ can be written as

$$\begin{aligned} x^{(i+1)} &= w^{(i)} + \tau^{(i)} x^{(i)} + \gamma^{(i)} x^{(i-1)}, \\ w^{(i)} &= T(Bx^{(i)} - \mu^{(i)} Ax^{(i)}), \\ \mu^{(i)} &= \mu(x^{(i)}), \end{aligned} \quad (11.17)$$

with properly chosen scalar iteration parameters $\tau^{(i)}$ and $\gamma^{(i)}$. The easiest and most efficient choice of parameters is based on an idea of *local optimality* [266]; namely, select $\tau^{(i)}$ and $\gamma^{(i)}$ that maximize the Rayleigh quotient $\mu(x^{(i+1)})$ by using the Rayleigh–Ritz method; see Algorithm 11.10.

ALGORITHM 11.10: PCG Method for GHEP

- (1) select a starting vector $x^{(0)}$.
- (2) **for** $i = 0, \dots$, until convergence **do**:
- (3) $\mu^{(i)} := (x^{(i)}, Bx^{(i)}) / (x^{(i)}, Ax^{(i)})$
- (4) $r := Bx^{(i)} - \mu^{(i)}Ax^{(i)}$
- (5) $w := Tr$
- (6) use the Rayleigh–Ritz method for $B - \mu A$ on $\text{span}\{w, x^{(i)}, x^{(i-1)}\}$
- (7) $x^{(i+1)} := \text{Ritz vector corresponding to the maximal Ritz value}$

For the locally optimal version of the PCG method, we can trivially apply convergence rate estimate (11.13) as the method converges not slower than the preconditioned steepest ascent in terms of maximizing the Rayleigh quotient.

We have collected the dominant costs per iteration for Algorithm 11.10 in terms of storage and floating point operations, respectively, in the following tables.

Item	Storage
Residual	1 n -vector
Approximate eigenvector	2 n -vectors
Temporary vectors	3–5 n -vectors

Action	Major cost
Rayleigh–Ritz method	9 dots and 2 matrix–vector products,
Residual r	2 matrix–vector products, 1 update
Preconditioned residual w	Depends on preconditioner T
Approximate eigenvector	1 update

The conjugate gradient methods for eigenvalue problems were suggested by Bradbury and Fletcher [61] and have attracted attention recently; different versions have been suggested, e.g., [433, 394, 429, 186, 464, 185, 167, 48]. They are usually constructed as general conjugate gradient methods, applied to minimization of the Rayleigh

quotient, and do not utilize the specific property of the Rayleigh quotient that local minimization problems can be easily solved using the Rayleigh–Ritz method. They are typically based on (now standard for linear systems) two linked two-term recurrences, where one of the scalar parameters is chosen to make directions conjugate in some sense. The peculiarity of Algorithm 11.10 is that it uses a three-term recurrence, which allows us both to choose scalar parameters by solving a local minimization problem and to not worry about finding conjugate directions.

We finally note that Algorithm 11.10 is somewhat unstable since the basis of the trial subspace is almost linearly dependent. This can be cured by an orthogonalization prior to applying the Rayleigh–Ritz method.

We present a block version of Algorithm 11.10 in the next subsection.

11.3.9 Preconditioned Simultaneous Iterations

A well-known idea of using *simultaneous* or *block iterations* provides an important improvement over single-vector methods and permits us to compute an ($m > 1$)-dimensional invariant subspace, rather than one eigenvector at a time. It can also serve as an acceleration technique over single-vector methods on parallel computers, as convergence for extreme eigenvalues usually increases with the size of the block and every step can be naturally implemented on wide varieties of multiprocessor computers.

A block algorithm is a straightforward generalization of the single-vector method and is typically combined with the Rayleigh–Ritz procedure. For every preconditioned eigensolver discussed earlier, several variants of simultaneous iterations can be easily suggested; e.g., see block versions of the preconditioned power method with shift in [266, 149, 63, 130]. For shortness, we describe here only one method—a simultaneous version of the locally optimal PCG method, suggested in [268]—and only for the pencil $B - \mu A$.

ALGORITHM 11.11: Simultaneous PCG Method for GHEP

- (1) select a starting vector $x_j^{(0)}$, $j = 1, \dots, m$
- (2) **for** $i = 0, \dots$, until convergence **do**:
- (3) $\mu_j^{(i)} := (x_j^{(i)}, B x_j^{(i)}) / (x_j^{(i)}, A x_j^{(i)})$ for $j = 1, \dots, m$
- (4) $r_j := B x_j^{(i)} - \mu_j^{(i)} A x_j^{(i)}$ for $j = 1, \dots, m$
- (5) $w_j := Tr_j$ for $j = 1, \dots, m$
- (6) Use the Rayleigh–Ritz method for $B - \mu A$
 span{ $w_1, \dots, w_m, x_1^{(i)}, \dots, x_m^{(i)}, x_1^{(i-1)}, \dots, x_m^{(i-1)}$ }
- (7) $x_j^{(i+1)} :=$ the j th Ritz vector corresponding to the j th
 from the top Ritz value for $j = 1, \dots, m$

As in the single-vector algorithm, special measures need to be taken to overcome the problem of almost-linear dependent basis in the trial subspace in Algorithm 11.11.

There is no theory available yet to predict accurately the speed of convergence of Algorithm 11.11. However, by analogy with known convergence theory of PCG system solvers, we expect convergence of $\mu_j^{(i)}$ to μ_j to be linear with the ratio

$$\left(\frac{1 - \sqrt{\xi_j}}{1 + \sqrt{\xi_j}} \right)^2, \quad \xi_j = \frac{\delta_0}{\delta_1} \frac{\mu_j - \mu_{m+1}}{\mu_j - \mu_{\min}}. \quad (11.18)$$

We compared numerically (see [268]) a block variant of the steepest ascent Algorithm 11.6 and Algorithm 11.11, with the block size $m = 3$. We plot, however, errors for only two top eigenvalues, leaving the third one out of the picture. The two shades of red represent the block SA method, and the two shades of blue correspond to Algorithm 11.11. It is easy to separate the methods on a black-and-white print, too, as Algorithm 11.11 always converges much faster. Two straight lines correspond to linear convergence predicted by (11.18).

In all tests, $B = I$, A is a diagonal matrix with minimal entries 1, 2, 3 and the maximal entry 10^{10} , and we measure the eigenvalue error as

$$\lambda_i^{(i)} - \lambda_j, \quad j = 1, 2.$$

The size of the problem is changing within 100 and 2000. The initial guess is random for every new run. The preconditioner T is a random symmetric positive definite matrix with the fixed value of $\kappa(TA) = \delta_1/\delta_0$ of (11.9).

Our random initial guess leads to very big initial errors as the matrix A is poorly conditioned, $\kappa_2(A) = 10^{10}$. We observe many initial errors at the level of 10^{10} on all figures below, but both tested methods successfully decrease errors to single digits just after a very few iterations.

We can see that the huge condition number of A , the size of the problem, the distribution of eigenvalues in the unwanted part of the spectrum, and the particular choice of the preconditioner T do not noticeably affect the convergence of Algorithm 11.11 as the elements of a bundle, of convergence history lines, are quite close to each other on all figures. Moreover, our theoretical prediction (11.18) of the rate of convergence of Algorithm 11.11 is relatively accurate, though pessimistic. A 10-fold increase of δ_1/δ_0 leads to the increase of number of iterations—10-fold for the block SA method, but only about 3-fold for Algorithm 11.11—exactly as we would expect. We observe that convergence for the first eigenvalue, in dark colors, is often faster than that for the second one, in lighter colors and dashed. Finally, we also observe a superlinear acceleration when the number of iterations is getting comparable with a half of the size of the problem solved. However, such a large ratio of the number of iterations and the size of the problem is atypical in practice, so we try to avoid it simply by increasing the size of the problem; e.g., on Figure 11.3 the first two runs are performed for the problem of the size 200; after noticing a superlinear convergence we immediately increase the size to 1000.

Figures 11.2 to 11.4 clearly illustrate two statements made in §11.1. First, the *performance of preconditioned solvers depends heavily on the quality of the preconditioner used*. A good preconditioner leads to a fast convergence on Figure 11.2. A bad preconditioner significantly slows the convergence down on Figure 11.4. We note that the size of the problem does not really affect the convergence speed. Second, the *implementation particular to each method can make a big difference even with the same preconditioner*. This is especially noticeable for poor-quality preconditioners, e.g., on Figure 11.4. Algorithm 11.11, the locally optimal PCG method, converges about a hundred times faster than Algorithm 11.6, though both algorithms use the same preconditioner and have similar costs of every iteration.

To conclude, our numerical results suggest that Algorithm 11.11 is a genuine conjugate gradient method. Our most recent numerical results [269] demonstrate that Algorithm 11.11 is practically the *optimal method on the whole class of the preconditioned solvers for symmetric definite eigenvalue problems*.

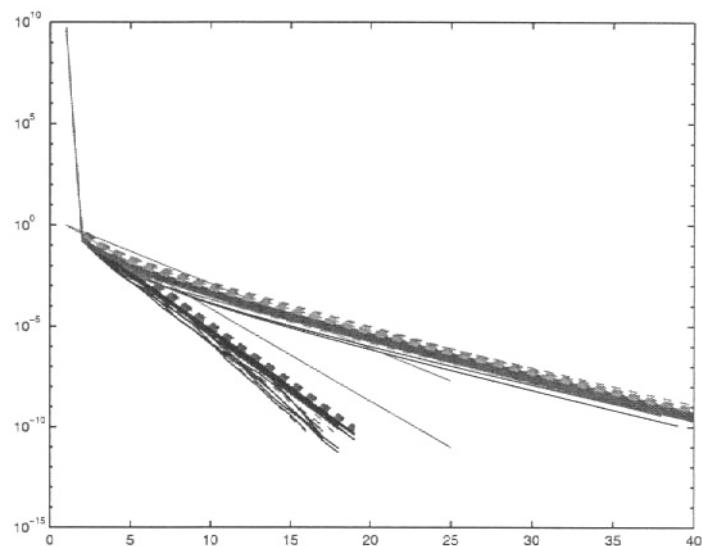


Figure 11.2: Conjugate gradient versus steepest ascent, $\delta_1/\delta_0 = 10$

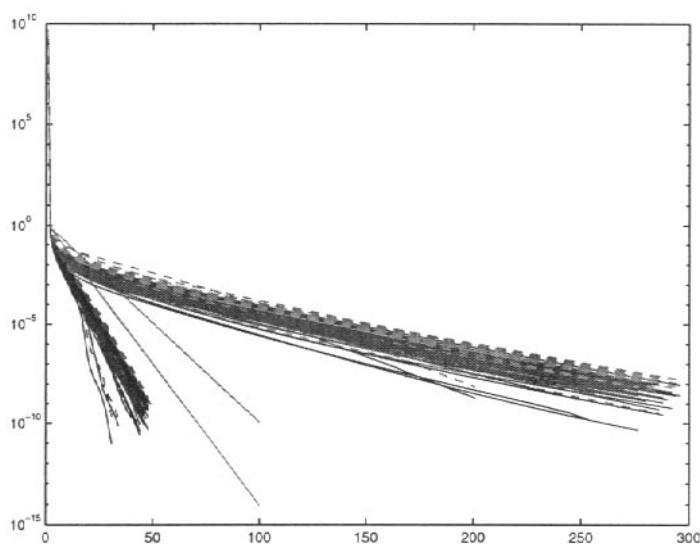


Figure 11.3: Conjugate gradient versus steepest ascent, $\delta_1/\delta_0 = 100$

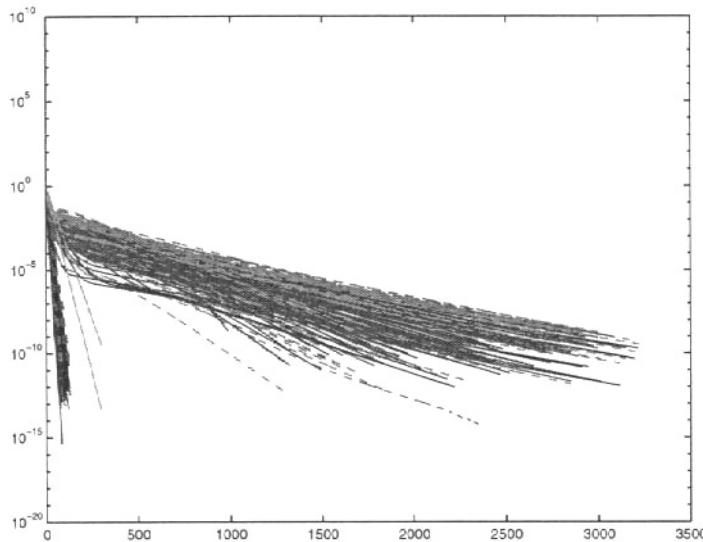


Figure 11.4: Conjugate gradient versus steepest ascent, $\delta_1/\delta_0 = 1000$

There are known [91] block-preconditioned methods for simultaneous computation of *both ends of the spectrum*, but they do not offer much improvement over the standard approach of computing minimal and maximal eigenvalues separately.

Another known idea of constructing block eigensolvers is to use *methods of nonlinear optimization to minimize, or maximize, the trace of the projection matrix* in the Rayleigh–Ritz method (see, e.g., [91, 16, 366] and §9.4), here $T = I$. It leads to methods somewhat similar to Algorithm 11.11 if the conjugate gradient method used for optimization, but Algorithm 11.11 has an advantage of using the Rayleigh–Ritz method.

The use of *locking*, a form of *deflation*, which exploits the unequal convergence rates of the different eigenvectors, enhances the performance of the preconditioned simultaneous iteration methods described above, quite similar to the case of classical methods, without preconditioning; see §4.3. Because of the different rates of convergence of each of the approximate eigenvectors computed by the simultaneous iteration, it is a common practice to extract those already converged and perform a deflation. We freeze extracted approximations and remove them from subsequent iterations such that there is no need to continue to multiply them by any matrices. However, we will still need either to include them in the basis of the trial subspace of the Rayleigh–Ritz method or to perform the subsequent orthogonalizations with respect to the frozen vectors whenever such orthogonalizations are needed.

The former possibility seems more natural and is easier to program than the latter one. It is also quite simple to analyze its influence on the iterative method using known results on accuracy of the Rayleigh–Ritz method; see, e.g., [387, 267]. The cost

of orthogonalization can be somewhat lower, however, as it reduces the dimension of the trial subspace.

We note that locking should also be used in single-vector iterative methods when we try to compute a group of eigenvectors one by one.

Using orthogonalization for locking in preconditioned eigensolvers may not, unfortunately, be simple if we want to be able to investigate the propagation of the resulting error in the process of further iterations.

As an example, let us consider our simplest preconditioned eigensolver, Algorithm 11.5, with additional orthogonalization, defined by an orthogonal projector P^\perp onto an orthogonal complement of the subspace, spanned by already computed eigenvectors, which can be written as

$$x^{(i+1)} = P^\perp w^{(i)} + \tau^{(i)} x^{(i)}, \quad w^{(i)} = T(Bx^{(i)} - \mu^{(i)} Ax^{(i)}). \quad (11.19)$$

We need to figure out what scalar products we want to use in our orthogonal projector P^\perp .

First, we need to choose a scalar product for the orthogonal complement of the subspace, spanned by already computed eigenvectors. The A -based scalar seems to be a natural choice here. When B is positive definite, it is common to use the B -based scalar product as well.

Second, we need to define a scalar product, in which projector P^\perp is orthogonal. A traditional approach is to use the same scalar product as on the first step. It is also trivial to implement in a code.

Unfortunately, with such a choice, the iteration operator in method (11.19) is losing the property of being symmetric, with respect to the T^{-1} -based scalar product. It makes theoretical investigation of the influence of orthogonalization to approximately computed eigenvectors quite complicated (see [147, 149]), where direct analysis of perturbations is done.

To preserve the symmetry, we must use the T^{-1} -orthogonal projector P^\perp in spite of the fact that we use a different, say A -based, scalar product on the first step to define the orthogonal complement. With this choice, we can use the standard and simple backward error analysis [264, 265] instead of direct analysis [147, 149], but the actual computation of $P^\perp w$ for a given w requires special attention.

Following [264, 265], we take the original subspace, spanned by the frozen approximate eigenvectors (let us call it \tilde{X}) and find a T^{-1} -orthogonal basis of the subspace $T\tilde{A}\tilde{X}$. Mathematically, a T^{-1} -orthogonal complement to the latter subspace coincides with an A -orthogonal complement of the original subspace, \tilde{X} . Now, we can use a standard T^{-1} -orthogonal projector onto a T^{-1} -orthogonal complement of $T\tilde{A}\tilde{X}$. Clearly, the T^{-1} -based scalar product must be possible to compute to use this trick.

We also note that the use of a T^{-1} -based scalar product, as well as any scalar product based on an ill-conditioned matrix, may lead to unstable computations.

11.3.10 Software Availability

The author (Knyazev) maintains a Web site, which can be accessed via the book's homepage, ETHOME, for interactive discussion on the subject, a bibliography page, and links to some software. In particular, the following software is available:

- MATLAB implementations of the block steepest ascent and the block conjugate gradient methods, used for numerical experiments in the previous subsection.
- A FORTRAN code that computes eigenvalues of the Laplacian in the L-shaped domain on a uniform mesh, using preconditioned domain decomposition Lanczos-type methods; see [270, 266, 271, 272].

This software is experimental and does not reflect all insights and possibilities described in this section.

Appendix

Of Things Not Treated

In this appendix, we briefly mention some of the subjects not treated in this book, giving references for the reader who is interested in pursuing a particular subject in depth.

Pseudospectra. Pseudospectra are a new tool for the study of matrices and linear operators. They provide an alternative to the traditional eigenvalues or spectra for revealing the behavior of systems, including stability, resonance, and accessibility to the matrix iterations and preconditioners. For an introduction to pseudospectra, see the survey article [441]. For a recent survey article on the computation of pseudospectra, see [440]. More recent work includes the study of structured pseudospectra for polynomial eigenvalue problems; see [437].

Eigenproblems of Structured Matrices. In some applications, matrices of special structure are common, and algorithms are developed which leave such a special structure invariant and return solutions that satisfy the constraints given by the special structure. Some such structures are Hamiltonian, symplectic, and quaternion matrices.

The canonical forms for Hamiltonian and symplectic matrices and pencils are given in [352, 307]. Algorithms for the eigenvalue problems of Hamiltonian and symplectic matrices can be found in [452, 71, 45, 46, 47, 325, 43, 44, 436]. For the eigenvalues of quaternion matrices, see [131, 69, 313, 165].

Unitary Eigenvalue Problems. A QR-style algorithm for the eigenproblem of a unitary matrix is presented in [201, 203]. The divide-and-conquer method is presented in [202, 11]. Other algorithms and applications of unitary eigenproblems can be found in [9, 10, 68].

Eigenproblems of Parameterized Matrices $A(p)$. A typical problem in this subject is to compute “continuous” invariant subspaces $Q(p)$ of a parameterized matrix $A(p)$; i.e., the entries of the matrix are functions of parameters p . See, for example, [125]. Numerical methods for finding multiple eigenvalues of matrices depending on

parameters are presented in [95]. The work [67] extends the singular value decomposition to a path of matrices $A(t)$, the so-called analytic singular value decomposition. An application of eigenproblems of matrices depending on parameters in structural engineering can be found in [1].

Eigenvalue Optimization. Optimization problems involving eigenvalues arise in many different mathematical disciplines. For a survey on the optimization of convex functions of eigenvalues of symmetric matrices subject to linear constraints, see [296]. For the problem of minimizing the largest real part of an eigenvalue of a parametrized square matrix, see [70].

Updating Eigenproblems. If one needs to solve eigenproblems of a sequence of matrices that differ only by quantities of small norm or low rank, it may be possible to update an eigensolution; see, for example, [15, 75, 469].

Updating the SVD. Many problems in signal processing computations are related to the problem of rapidly updating the singular value decomposition. It is possible to stably update the singular value decomposition in time proportional to mn , where m and n are the dimensions of the original given matrix; see [83] and references given there.

For updating the partial singular value decomposition and applications in latent semantic indexing, see [466, 467].

Some Modified Eigenvalue Problems. In some applications, for example, in optimization and data analysis contexts, eigenvalue problems with constraints occur; see [36, 200].

Inverse Eigenvalue Problems. An inverse eigenvalue problem concerns the reconstruction of a matrix from prescribed spectral data. The spectral data involved may consist of complete or only partial information on eigenvalues or eigenvectors. The matrix constructed is required to maintain a certain specific structure as well as that given spectral property. Two fundamental questions associated with any inverse eigenvalue problem are solvability and computability. Early survey papers on the subject include [56, 181]. The book by Zhou and Dai [470] (in Chinese) is an early attempt to survey the subject in greater depth. It is updated and extended significantly in the recent survey paper [84].

High Relative Accuracy Eigensolvers. Most of the well-known algorithms that are discussed here give results that are accurate relative to the size of the larger elements of the matrix. This is good for large eigenvalues, but may not be the best possible for very small eigenvalues. See [40, 118, 229] and references given there for computing eigenvalues and singular values to high relative accuracy.

Homotopy Continuation. A special case of updating is homotopy continuation when a continuous path of eigenproblems are solved. Most often the path is started on a problem whose solution is known and continues to the problem of interest; see [304, 442, 235, 309, 310, 469, 305].

Homotopy continuation methods were studied in the early 1960s by Davidenko [97, 98]. Some continuation methods require solving linear systems with a shifted matrix and may lead to preconditioned eigensolvers of the type discussed in section 11.3.7.

Matrix Sign Function Methods. The matrix sign function is a popular tool for solving the algebraic Riccati equation; see, for example, [72, 287]. It can also be used as a tool to find eigenvalues and the corresponding invariant subspaces of a dense square matrix in a region of the complex plane; see [30, 73, 31] and references given there.

Multilevel, Multigrid, and Domain Decomposition Methods. The multilevel, multigrid, and domain decomposition methods developed for numerical solution of partial differential equations (see, for example, [66, 218, 321, 62, 415, 327]) are also applicable for solving eigenvalue problems.

There are three possibilities. First, an algorithm developed for a linear system can sometimes be modified to solve the corresponding eigenvalue problem directly; see, for example, [215, 320, 64, 216, 217, 315, 74] on multigrid methods and [311, 312, 74, 314] on domain decomposition methods. Second, a multilevel method can be used as an inner solver for linear systems with a shifted matrix that appear in shift-and-invert spectral transformation techniques; see, e.g., [39, 237, 238]. Finally, some multilevel algorithms can serve as a powerful tool for constructing preconditioners to be used in preconditioned eigensolvers such as discussed in Chapter 11. Additional references include [341, 342].

Some multilevel methods allow algebraic interpretation. Many domain decomposition methods with overlap are equivalent to well-known block-relaxation methods for matrices; see [454, 314, 388].

Domain decomposition methods for eigenvalue problems without overlap can often be reduced to iterating a Schur complement of the corresponding shifted matrix. For example, see [2, 401, 404, 399, 282, 400, 264, 270, 271, 272]. A component mode synthesis method (e.g., [57, 58, 59, 60, 164]) is yet another form of the domain decomposition method for eigenvalue problems.

This page intentionally left blank

Bibliography

- [1] M. R. Abdel-Aziz. Safeguarded use of the implicit restarted Lanczos technique for solving non-linear structural eigensystems. *Internat. J. Numer. Methods Engrg.*, 37:3117–3133, 1994.
- [2] A. Abramow and M. Neuhaus. Bemerkungen über Eigenwertprobleme von Matrizen höherer Ordnung. In *Les mathématiques de l'ingénieur*, pages 176–179. Mém. Publ. Soc. Sci. Arts Lett. Hainaut, Vol. hors Série, Maison Léon Losseau, Mons, France, 1958.
- [3] G. Adams, A. Bojanczyk, and F. T. Luk. Computing the PSVD of two 2×2 triangular matrices. *SIAM J. Matrix Anal. Appl.*, 15(2):366–382, 1994.
- [4] L. Ahlfors. *Complex Analysis*. McGraw-Hill, New York, 1966.
- [5] J. I. Aliaga, D. L. Boley, R. W. Freund, and V. Hernández. A Lanczos-type method for multiple starting vectors. *Math. Comp.* 69:1577–1601, 2000.
- [6] P. R. Amestoy and I. S. Duff. Vectorization of a multiprocessor multifrontal code. *Internat. J. Supercomputer Appl.*, 3:41–59, 1989.
- [7] P. R. Amestoy and I. S. Duff. Memory management issues in sparse multifrontal methods on multiprocessors. *Internat. J. Supercomputer Appl.*, 7:64–82, 1993.
- [8] P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, and J. Koster. A fully asynchronous multifrontal solver using distributed dynamic scheduling. Technical Report RAL-TR-1999-059, Rutherford Appleton Laboratory, Oxfordshire, UK, 1999. Software available at <http://www.pallas.de/parasol>.
- [9] G. S. Ammar, W. B. Gragg, and L. Reichel. Downdating Szegő polynomials and data fitting applications. *Linear Algebra Appl.*, 172:315–336, 1992.
- [10] G. S. Ammar and C. He. On an inverse eigenvalue problem for unitary Hessenberg matrices. *Linear Algebra Appl.*, 218:263–271, 1995.
- [11] G. S. Ammar, L. Reichel, and D. C. Sorensen. Algorithm 730: An implementation of a divide and conquer method for the unitary eigenproblem. *ACM Trans. Math. Software*, 20:161–170, 1994.
- [12] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. SIAM, Philadelphia, Third edition, 1999.

- [13] P. J. Anderson and G. Loizou. A Jacobi type method for complex symmetric matrices. *Numer. Math.*, 25:347–363, 1976.
- [14] I. Andersson. Experiments with the conjugate gradient algorithm for the determination of eigenvalues of symmetric matrices. Technical Report UMINF-4.71, University of Umeå, Sweden, 1971.
- [15] P. Arbenz and G. H. Golub. On the spectral decomposition of Hermitian matrices modified by low rank perturbations with applications. *SIAM J. Matrix Anal. Appl.*, 9:40–58, 1988.
- [16] T. Arias, A. Edelman, and S. Smith. Curvature in conjugate gradient eigenvalue computation with applications. In J. G. Lewis, editor, *Proceedings of the 1994 SIAM Applied Linear Algebra Conference*, pages 233–238. SIAM, Philadelphia, 1994.
- [17] M. Arioli, I. S. Duff, and D. Ruiz. Stopping criteria for iterative solvers. Report RAL-91-057, Central Computing Center, Rutherford Appleton Laboratory, Oxfordshire, UK, 1992.
- [18] V. I. Arnold. On matrices depending on parameters. *Russian Math. Surveys*, 26:29–43, 1971.
- [19] W. E. Arnoldi. The principle of minimized iterations in the solution of the matrix eigenvalue problem. *Quart. Appl. Math.*, 9:17–29, 1951.
- [20] E. Artin. *Geometric Algebra*. Interscience, New York, 1957.
- [21] C. Ashcraft and R. Grimes. SPOOLES: An object-oriented sparse matrix library. In *Proceedings of the Ninth SIAM Conference on Parallel Processing*. SIAM, Philadelphia, 1999. Software available at <http://www.netlib.org/linalg/spooles>.
- [22] J. Baglama, D. Calvetti, and L. Reichel. Iterative methods for the computation of a few eigenvalues of a large symmetric matrix. *BIT*, 36(3):400–421, 1996.
- [23] J. Baglama, D. Calvetti, and L. Reichel. Fast Leja points. *Electron. Trans. Numer. Anal.*, 7:124–140, 1998.
- [24] J. Baglama, D. Calvetti, L. Reichel, and A. Ruttan. Computation of a few close eigenvalues of a large matrix with application to liquid crystal modeling. *J. Comput. Phys.*, 146:203–226, 1998.
- [25] Z. Bai. The CSD, GSVD, their applications and computations. Preprint Series 958, Institute for Mathematics and Its Applications, University of Minnesota, Minneapolis, April 1992. Available at <http://www.cs.ucdavis.edu/~bai>.
- [26] Z. Bai. Error analysis of the Lanczos algorithm for the nonsymmetric eigenvalue problem. *Math. Comp.*, 62:209–226, 1994.
- [27] Z. Bai. A spectral transformation block Lanczos algorithm for solving sparse non-Hermitian eigenproblems. In J. G. Lewis, editor, *Proceedings of the Fifth SIAM Conference on Applied Linear Algebra*, pages 307–311. SIAM, Philadelphia, 1994.

- [28] Z. Bai, D. Day, J. Demmel, and J. Dongarra. A test matrix collection for non-Hermitian eigenvalue problems. Technical Report CS-97-355, University of Tennessee, Knoxville, 1997. LAPACK Working Note #123, Software and test data available at <http://math.nist.gov/MatrixMarket/>.
- [29] Z. Bai, D. Day, and Q. Ye. ABLE: An adaptive block lanczos method for non-hermitian eigenvalue problems. *SIAM J. Matrix Anal. Appl.*, 20:1060–1082, 1999.
- [30] Z. Bai and J. Demmel. Design of a parallel nonsymmetric eigenroutine toolbox, Part I. In R. F. Sincovec et al., editors, *Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing*. SIAM, Philadelphia, 1993. Long version available as Computer Science Report CSD-92-718, University of California, Berkeley, 1992.
- [31] Z. Bai and J. Demmel. Using the matrix sign function to compute invariant subspaces. *SIAM J. Matrix Anal. Appl.*, 19:205–225, 1998.
- [32] Z. Bai, J. Demmel, and M. Gu. An inverse free parallel spectral divide and conquer algorithm for nonsymmetric eigenproblems. *Numer. Math.*, 76:279–308, 1997.
- [33] Z. Bai and J. W. Demmel. On swapping diagonal blocks in real Schur form. *Linear Algebra Appl.*, 186:73–95, 1993.
- [34] Z. Bai, P. Feldmann, and R. W. Freund. How to make theoretically passive reduced-order models passive in practice. In *Proceedings of the IEEE 1998 Custom Integrated Circuits Conference*, pages 207–210. IEEE Press, Piscataway, NJ, 1998.
- [35] Z. Bai and R. W. Freund. A band symmetric Lanczos process based on coupled recurrences with applications. Technical Report Numerical Analysis Manuscript, Bell Laboratories, Murray Hill, NJ, USA, 1998.
- [36] Z. Bai and G. Golub. Some unusual matrix eigenvalue problems. In J. Palma, J. Dongarra, and V. Hernandez, editors, *Proceedings of VECPAR'98 - Third International Conference for Vector and Parallel Processing*, Lecture Notes in Computer Science. Vol. 1573, pages 4–19. Springer-Verlag, New York, 1999.
- [37] Z. Bai and G. W. Stewart. Algorithm 776. SRRIT — A FORTRAN subroutine to calculate the dominant invariant subspaces of a nonsymmetric matrix. *ACM Trans. Math. Software*, 23:494–513, 1998.
- [38] S. Balay, W. Gropp, L. C. McInnes, and B. Smith. PETSc 2.0 Users Manual. Technical Report ANL-95/11 - Revision 2.0.28, Argonne National Laboratory, Argonne, IL, 2000. Software available at <http://www.mcs.anl.gov/petsc>.
- [39] R. E. Bank. Analysis of a multilevel inverse iteration procedure for eigenvalue problems. *SIAM J. Numer. Anal.*, 19(5):886–898, 1982.
- [40] J. Barlow and J. Demmel. Computing accurate eigensystems of scaled diagonally dominant matrices. *SIAM J. Numer. Anal.*, 27(3):762–791, 1990.

- [41] R. Barrett, M. Berry, T. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. SIAM, Philadelphia, 1994.
- [42] K.-J. Bathe and E. L. Wilson. *Numerical Methods in Finite Element Analysis*. Prentice Hall, Englewood Cliffs, NJ, 1976.
- [43] P. Benner and H. Faßbender. The symplectic eigenvalue problem, the butterfly form, the SR algorithm, and the Lanczos method. *Linear Algebra Appl.*, 275/276:19–47, 1998.
- [44] P. Benner, H. Fassbender, and D. Watkins. SR and SZ algorithms for the symplectic (butterfly) eigenproblem. *Linear Algebra Appl.*, 287:41–76, 1999.
- [45] P. Benner, V. Mehrmann, and H. Xu. A new method for computing the stable invariant subspace of a real hamiltonian matrix. *J. Comput. Appl. Math.*, 86:17–43, 1997.
- [46] P. Benner, V. Mehrmann, and H. Xu. A numerical stable, structure preserving method for computing the eigenvalues of real Hamiltonian or symplectic pencils. *Numer. Math.*, 78:329–358, 1998.
- [47] P. Benner, V. Mehrmann, and H. Xu. A note on the numerical solution of complex Hamiltonian and skew-Hamiltonian eigenvalue problem. *Electron. Trans. Numer. Anal.*, 8:115–126, 1999.
- [48] L. Bergamaschi, G. Gambolati, and G. Pini. Asymptotic convergence of conjugate gradient methods for the partial symmetric eigenproblem. *Numer. Linear Algebra Appl.*, 4(2):69–84, 1997.
- [49] M. Berry. Large scale singular value computations. *Internat. J. Supercomputer Appl.*, 6(1):13–49, 1992.
- [50] Å. Björck. *Numerical Solutions for Least Squares Problems*. SIAM, Philadelphia, 1996.
- [51] Å. Björck and V. Pereyra. Solution of vandermonde systems of equations. *Math. Comp.*, 24:893–903, 1970.
- [52] L. S. Blackford, J. Choi, A. Cleary, E. D’Azevedo, J. Demmel, I. Dhillon, J. Dongarra, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. Whaley. *ScalAPACK Users’ Guide*. SIAM, Philadelphia, 1997.
- [53] A. Bojanczyk and P. Van Dooren. On propagating orthogonal transformations in a product of 2×2 triangular matrices. In *Numerical Linear Algebra*. de Gruyter, Berlin, 1993.
- [54] A. Bojanczyk, P. Van Dooren, L. M. Ewerbring, and F. T. Luk. An accurate product SVD algorithm. *J. Signal Processing*, 25:189–201, 1991.
- [55] D. Boley. The algebraic structure of pencils and block Toeplitz matrices. *Linear Algebra Appl.*, 279:255–279, April 1998.

- [56] D. Boley and G. H. Golub. A survey of matrix inverse eigenvalue problems. *Inverse Problems*, 3:595–622, 1987.
- [57] F. Bourquin. Analysis and comparison of several component mode synthesis methods on one-dimensional domains. *Numer. Math.*, 58(1):11–33, 1990.
- [58] F. Bourquin. Component mode synthesis and eigenvalues of second order operators: discretization and algorithm. *RAIRO Modél. Math. Anal. Numér.*, 26(3):385–423, 1992.
- [59] F. Bourquin. A domain decomposition method for the eigenvalue problem in elastic multistructures. In *Asymptotic Methods for Elastic Structures (Lisbon, 1993)*, pages 15–29. de Gruyter, Berlin, 1995.
- [60] F. Bourquin and P. G. Ciarlet. Modelling and justification of eigenvalue problems for junctions between elastic structures. *J. Funct. Anal.*, 87(2):392–427, 1989.
- [61] W. W. Bradbury and R. Fletcher. New iterative methods for solution of the eigenproblem. *Numer. Math.*, 9:259–267, 1966.
- [62] J. H. Bramble. *Multigrid Methods*. Longman Scientific & Technical, Harlow, UK, 1993.
- [63] J. H. Bramble, J. E. Pasciak, and A. V. Knyazev. A subspace preconditioning algorithm for eigenvector/eigenvalue computation. *Adv. Comput. Math.*, 6(2):159–189, 1996.
- [64] A. Brandt, S. McCormick, and J. Ruge. Multigrid methods for differential eigenproblems. *SIAM J. Sci. Statist. Comput.*, 4(2):244–260, 1983.
- [65] C. Brezinski, M. Redivo Zaglia, and H. Sadok. Avoiding breakdown and near-breakdown in Lanczos type algorithms. *Numer. Algorithms*, 1:261–284, 1991.
- [66] W. L. Briggs. *A Multigrid Tutorial*. SIAM, Philadelphia, 1987.
- [67] A. Bunse-Gerstner, R. Byers, V. Mehrmann, and N. K. Nichols. Numerical computation of an analytic singular value decomposition of a matrix valued function. *Numer. Math.*, 60:1–39, 1991.
- [68] A. Bunse-Gerstner and C. He. On the Sturm sequence of polynomials for unitary Hessenberg matrices. *SIAM J. Matrix Anal. Appl.*, 16:1043–1055, 1995.
- [69] A. Bunse-Gerstner and V. Mehrmann. The quaternion QR algorithm. *Numer. Math.*, 55:83–95, 1989.
- [70] J. V. Burke, A. S. Lewis, and M. L. Overton. Optimizing matrix stability. *Proc. Amer. Math. Soc.*, 1999, to appear.
- [71] R. Byers. A Hamiltonian QR-algorithm. *SIAM J. Sci. Statist. Comput.*, 7:212–229, 1986.
- [72] R. Byers. Solving the algebraic Riccati equation with the matrix sign function. *Linear Algebra Appl.*, 85:267–279, 1987.

- [73] R. Byers, C. He, and Mehrmann. The matrix sign function method and the computation of invariant subspaces. *SIAM J. Matrix Anal. Appl.*, 18:615–632, 1997.
- [74] Z. Q. Cai, J. Mandel, and S. McCormick. Multigrid methods for nearly singular linear equations and eigenvalue problems. *SIAM J. Numer. Anal.*, 34:178–200, 1997.
- [75] C. Carey, G. H. Golub, and K. H. Law. A Lanczos-based method for structural dynamics re-analysis problems. Manuscript na-93-03, Computer Science Department, Stanford University, Stanford, CA, 1993.
- [76] J. Carrier, L. Greengard, and V. Rokhlin. A fast adaptive multipole algorithm for particle simulations. *SIAM J. Sci. Statist. Comput.*, 9:669–686, 1988.
- [77] F. Chaitin-Chatelin and V. Frayssé. *Lectures on Finite Precision Computations*. SIAM, Philadelphia, 1996.
- [78] T. F. Chan, E. Gallopoulos, V. Simoncini, T. Szeto, and C. H. Tong. A quasi-minimal residual variant of the Bi-CGSTAB algorithm for nonsymmetric systems. *SIAM J. Sci. Comput.*, 15:338–347, 1994.
- [79] F. Chatelin. *Eigenvalues of Matrices*. Wiley, New York, 1993.
- [80] I. Chavel. *Riemannian Geometry—A Modern Introduction*. The Cambridge University Press, Cambridge, UK, 1993.
- [81] T.-Y. Chen. Balancing sparse matrices for computing eigenvalues. Master’s thesis, University of California, Berkeley, May 1998.
- [82] T.-Y. Chen and J. Demmel. Balancing sparse matrices for computing eigenvalues. *Linear Algebra Appl.*, 309:261–287, 2000.
- [83] X. Chi and M. Gu. Updating the SVD. CAM technical report, Department of Mathematics, University of California, Los Angeles, 2000.
- [84] M. T. Chu. Inverse eigenvalue problems. *SIAM Rev.*, 40:1–39, 1998.
- [85] B. D. Craven. Complex symmetric matrices. *J. Austral. Math. Soc.*, 10:341–354, 1969.
- [86] C. R. Crawford. Algorithm 646 PDFIND: A routine to find a positive definite linear combination of two real symmetric matrices. *ACM Trans. Math. Software*, 12:278–282, 1986.
- [87] C. R. Crawford and Y. S. Moon. Finding a positive definite linear combination of two Hermitian matrices. *Linear Algebra Appl.*, 51:37–48, 1983.
- [88] M. Crouzeix, B. Philippe, and M. Sadkane. The Davidson method. *SIAM J. Sci. Comput.*, 15:62–76, 1994.
- [89] J. K. Cullum and W. E. Donath. A block Lanczos algorithm for computing the q algebraically largest eigenvalues and a corresponding eigenspace for large, sparse symmetric matrices. In *Proceedings of the 1994 IEEE Conference on Decision and Control*, pages 505–509. IEEE Press, Piscataway, NJ, 1974.

- [90] J. K. Cullum and R. A. Willoughby. Computing eigenvalues of very large symmetric matrices—an implementation of a Lanczos algorithm with no reorthogonalization. *J. Comput. Phys.*, 44:329–358, 1981.
- [91] J. K. Cullum and R. A. Willoughby. *Lanczos Algorithms for Large Symmetric Eigenvalue Computations. Volume 1, Theory*. Birkhäuser, Boston, 1985.
- [92] J. K. Cullum and R. A. Willoughby. *Lanczos Algorithms for Large Symmetric Eigenvalue Computations. Volume 2, Programs*. Birkhäuser, Boston, 1985.
- [93] J. K. Cullum and R. A. Willoughby. A practical procedure for computing eigenvalues of large sparse nonsymmetric matrices. In J. K. Cullum and R. A. Willoughby, editors, *Large Scale Eigenvalue Problems*, pages 193–240. Elsevier Science Publishers, 1986.
- [94] J. K. Cullum and R. A. Willoughby. A QL procedure for computing the eigenvalues of complex symmetric tridiagonal matrices. *SIAM J. Matrix Anal. Appl.*, 17:83–109, 1996.
- [95] H. Dai and P. Lancaster. Numerical methods for finding multiple eigenvalues of matrices depending on parameters. *Numer. Math.*, 76:189–208, 1997.
- [96] J. W. Daniel, W. B. Gragg, L. Kaufman, and G. W. Stewart. Reorthogonalization and stable algorithms for updating the Gram-Schmidt QR factorization. *Math. Comp.*, 30:772–795, 1976.
- [97] D. F. Davidenko. The method of variation of parameters as applied to the computation of eigenvalues and eigenvectors of matrices. *Soviet Math. Dokl.*, 1:364–367, 1960.
- [98] D. F. Davidenko. On the computation of eigenvalues and eigenvectors of matrices. *Dokl. Akad. Nauk SSSR*, 141:277–280, 1961.
- [99] E. R. Davidson. The iterative calculation of a few of the lowest eigenvalues and corresponding eigenvectors of large real symmetric matrices. *J. Comput. Phys.*, 17:87–94, 1975.
- [100] E. R. Davidson. Matrix eigenvector methods. In G. H. F. Diercksen and S. Wilson, editors, *Methods in Computational Molecular Physics*, pages 95–113. Reidel, Boston, 1983.
- [101] C. Davis and W. Kahan. The rotation of eigenvectors by a perturbation. III. *SIAM J. Numer. Anal.*, 7:1–46, 1970.
- [102] G. J. Davis. Numerical solution of a quadratic matrix equation. *SIAM J. Sci. Comput.*, 2:164–175, 1981.
- [103] T. A. Davis and I. S. Duff. A combined unifrontal/multifrontal method for unsymmetric sparse matrices. Technical Report TR-95-020, Computer and Information Sciences Department, University of Florida, Gainesville, 1995. Software available at <http://www.netlib.org/linalg/umfpack2.2.tgz>.

- [104] D. Day. *Semi-Duality in the Two-Sided Lanczos Algorithm*. Ph.D. thesis, University of California, Berkeley, 1993.
- [105] D. Day. An efficient implementation of the nonsymmetric Lanczos algorithm. *SIAM J. Matrix Anal. Appl.*, 18:566–589, 1997.
- [106] I. De Hoyos. Points of continuity of the Kronecker canonical form. *SIAM J. Matrix Anal. Appl.*, 11(2):278–300, April 1990.
- [107] J. de Leeuw and W. Heiser. Theory of multidimensional scaling. In P. R. Krishnaiah and L. N. Kanal, editors, *Handbook of Statistics, Vol. 2*, pages 285–316. North-Holland, Amsterdam, 1982.
- [108] B. De Moor. On the structure of generalized singular value and QR decompositions. *SIAM J. Matrix Anal. Appl.*, 15(1):347–358, 1994.
- [109] G. De Samblanx. *Filtering and restarting projection methods for eigenvalue problems*. PhD Thesis, Katholieke Universiteit Leuven, Department of Computer Science, 3001 Heverlee, Belgium, 1998.
- [110] G. De Samblanx and A. Bultheel. Nested Lanczos: implicitly restarting a Lanczos algorithm. *Numer. Algorithms*, 18:31–50, 1998.
- [111] E.. De Sturler. A parallel restricted version of GMRES(m). Technical Report Tech. Report Preprint 91-085, Delft University of Technology, Delft, The Netherlands, 1992.
- [112] E. De Sturler and H. A. van der Vorst. Communication cost reduction for krylov methods on parallel computers. In W. Gentzsch and U. Harms, editors, *High Performance and Networking Tools, Vol. 2*, Lecture Notes in Computer Science. Vol. 797, pages 190–195. Springer-Verlag, Berlin, 1994.
- [113] R. S. Dembo, S. C. Eisenstat, and T. Steihaug. Inexact Newton methods. *SIAM J. Numer. Anal.*, 19:400–408, 1982.
- [114] J. Demmel. *Applied Numerical Linear Algebra*. SIAM, Philadelphia, 1997.
- [115] J. Demmel. Accurate SVDs of structured matrices. *SIAM J. Matrix Anal. Appl.*, 21(3):562–580, 2000.
- [116] J. Demmel and A. Edelman. The dimension of matrices (matrix pencils) with given Jordan (Kronecker) canonical forms. *Linear Algebra Appl.*, 230:61–87, 1995.
- [117] J. Demmel and W. Gragg. On computing accurate singular values and eigenvalues of matrices with acyclic graphs. *Linear Algebra Appl.*, 185:203–217, 1993.
- [118] J. Demmel, M. Gu, S. Eisenstat, I. Slapničar, K. Veselić, and Z. Drmač. Computing the singular value decomposition with high relative accuracy. *Linear Algebra Appl.*, 299:21–80, 1999.
- [119] J. Demmel and B. Kågström. Computing stable eigendecompositions of matrix pencils. *Linear Algebra Appl.*, 88/89:139–186, 1987.

- [120] J. Demmel and B. Kågström. Accurate solutions of ill-posed problems in control theory. *SIAM J. Matrix Anal. Appl.*, 9(1):126–145, 1988.
- [121] J. Demmel and B. Kågström. The generalized Schur decomposition of an arbitrary pencil $A - \lambda B$: Robust software with error bounds and applications. Part I: Theory and algorithms. *ACM Trans. Math. Software*, 19(2):160–174, 1993.
- [122] J. Demmel and B. Kågström. The generalized Schur decomposition of an arbitrary pencil $A - \lambda B$: Robust software with error bounds and applications. Part II: Software and applications. *ACM Trans. Math. Software*, 19(2):175–201, 1993.
- [123] J. Demmel and W. Kahan. Accurate singular values of bidiagonal matrices. *SIAM J. Sci. Statist. Comput.*, 11:873–912, 1990.
- [124] J. Demmel and K. Veselić. Jacobi's method is more accurate than QR. *SIAM J. Matrix Anal. Appl.*, 13(4):1204–1245, 1992.
- [125] J. W. Demmel, L. Dieci, and M. Friedman. *SIAM J. Sci. Comput.*, 22(1):81–94, 2000.
- [126] J. W. Demmel, S. C. Eisenstat, J. R. Gilbert, X. S. Li, and J. W. H. Liu. A supernodal approach to sparse partial pivoting. *SIAM J. Matrix Anal. Appl.*, 20(3):720–755, 1999. Software available at <http://www.nerc.gov/~xiaoye/SuperLU>.
- [127] J. W. Demmel, J. R. Gilbert, and X. S. Li. An asynchronous parallel supernodal algorithm for sparse Gaussian elimination. *SIAM J. Matrix Anal. Appl.*, 20(4):915–952, 1999. Software available at <http://www.nerc.gov/~xiaoye/SuperLU>.
- [128] I. Dhillon. *A New $O(n^2)$ Algorithm for the Symmetric Tridiagonal Eigenvalue/Eigenvector Problem*. Ph.D. thesis, University of California, Berkeley, 1997.
- [129] I. S. Dhillon. Current inverse iteration software can fail. *BIT*, 38(4):685–704, 1998.
- [130] D. C. Dobson. An efficient method for band structure calculations in 2D photonic crystals. *J. Comput. Phys.*, 149(2):363–376, 1999.
- [131] J. Dongarra, J. Gabriel, D. Kolling, and J. Wilkinson. The eigenvalue problem for Hermitian matrices with time reversal symmetry. *Linear Algebra Appl.*, 60:27–42, 1984.
- [132] J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W. Stewart. *LINPACK Users' Guide*. SIAM, Philadelphia, 1979.
- [133] J. J. Dongarra, J. Du Croz, S. Hammarling, and R. J. Hanson. An extended set of FORTRAN basic linear algebra subprograms. *ACM Trans. Math. Software*, 14:1–32, 1988.
- [134] J. J. Dongarra, J. DuCroz, I. S. Duff, and S. Hammarling. A set of level 3 basic linear algebra subprograms. *ACM Trans. Math. Software*, 16:1–17, 1990.

- [135] J. J. Dongarra, I. S. Duff, D. C. Sorensen, and H. A. van der Vorst. *Numerical Linear Algebra for High-Performance Computers*. SIAM, Philadelphia, PA, 1998.
- [136] Z. Drmač. A posteriori computation of the singular vectors in a preconditioned Jacobi SVD algorithm. *IMA J. Numer. Anal.*, 19:191–213, 1999.
- [137] I. S. Duff. Direct methods. Technical Report RAL-98-056, Rutherford Appleton Laboratory, Oxfordshire, UK, 1998.
- [138] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices*. Clarendon Press, Oxford, UK, 1986.
- [139] I. S. Duff, R. G. Grimes, and J. G. Lewis. Sparse matrix test problems. *ACM Trans. Math. Software*, 15:1–14, 1989.
- [140] I. S. Duff and J. K. Reid. The multifrontal solution of indefinite sparse symmetric linear equations. *ACM Trans. Math. Software*, 9(3):302–325, September 1983.
- [141] I. S. Duff and J. K. Reid. MA47, a Fortran code for direct solution of indefinite sparse symmetric linear systems. Technical Report RAL-95-001, DRAL, Chilton Didcot, UK, 1995.
- [142] I. S. Duff and J. K. Reid. The design of MA48, a code for the direct solution of sparse unsymmetric linear systems of equations. *ACM Trans. Math. Software*, 22:187–226, 1996.
- [143] I. S. Duff and J. A. Scott. Computing selected eigenvalues of large sparse unsymmetric matrices using subspace iteration. *ACM Trans. Math. Software*, 19:137–159, 1993.
- [144] I. S. Duff and J. A. Scott. The design of a new frontal code for solving sparse unsymmetric systems. *ACM Trans. Math. Software*, 22(1):30–45, 1996.
- [145] R. J. Duffin. A minimax theory for overdamped networks. *J. Rational Mech. Anal.*, 4:221–233, 1955.
- [146] E. G. D'yakonov. Iteration methods in eigenvalue problems. *Math. Notes*, 34:945–953, 1983.
- [147] E. G. D'yakonov. *Optimization in solving elliptic problems*. CRC Press, Boca Raton, FL, 1996. Translated from the 1989 Russian original; translated, edited, and with a preface by Steve McCormick.
- [148] E. G. D'yakonov and A. V. Knyazev. Group iterative method for finding lower-order eigenvalues. *Moscow University, Ser. 15, Computational Math. and Cybernetics*, 2:32–40, 1982.
- [149] E. G. D'yakonov and A. V. Knyazev. On an iterative method for finding lower eigenvalues. *Russian J. Numer. Anal. Math. Modelling*, 7(6):473–486, 1992.
- [150] E. G. D'yakonov and M. Yu. Orehkov. Minimization of the computational labor in determining the first eigenvalues of differential operators. *Math. Notes*, 27(5–6):382–391, 1980.

- [151] A. Edelman, T. A. Arias, and S. T. Smith. The geometry of algorithms with orthogonality constraints. *SIAM J. Matrix Anal. Appl.*, 20:303–353, 1999.
- [152] A. Edelman, E. Elmroth, and B. Kågström. A geometric approach to perturbation theory of matrices and matrix pencils. Part I: Versal deformations. *SIAM J. Matrix Anal. Appl.*, 18(3):653–692, 1997.
- [153] A. Edelman, E. Elmroth, and B. Kågström. A geometric approach to perturbation theory of matrices and matrix pencils. Part II: A stratification-enhanced staircase algorithm. *SIAM J. Matrix Anal. Appl.*, 20(3):667–699, 1999.
- [154] A. Edelman and Y. Ma. Staircase failures explained by orthogonal versal forms. *SIAM J. Matrix Anal. Appl.*, 21(3):1004–1025, 2000.
- [155] A. Edelman and S. T. Smith. On conjugate gradient-like methods for eigen-like problems. *BIT*, 36:494–508, 1996. See also Loyce Adams and J. L. Nazareth, editors, *Proc. Linear and Nonlinear Conjugate Gradient-Related Methods*, SIAM, Philadelphia, 1996.
- [156] V. Eijkhout. Distributed sparse data structures for linear algebra operations. Technical Report CS 92-169, Computer Science Department, University of Tennessee, Knoxville, TN, 1992. LAPACK Working Note #50, <http://www.netlib.org/lapack/lawns/lawn50.ps>.
- [157] S. C. Eisenstat and I. C. F. Ipsen. Relative perturbation techniques for singular value problems. *SIAM J. Numer. Anal.*, 32:1972–1988, 1995.
- [158] L. Eldén. Algorithms for the regularization of ill-conditioned least-squares problems. *BIT*, 17:134–145, 1977.
- [159] E. Elmroth, P. Johansson, and B. Kågström. Computation and presentation of graphs displaying closure hierarchies of Jordan and Kronecker structures. Technical Report UMINF-99.12, Department of Computing Science, Umeå University, Umeå, Sweden, 1999.
- [160] E. Elmroth and B. Kågström. The set of 2-by-3 matrix pencils—Kronecker structures and their transitions under perturbations. *SIAM J. Matrix Anal. Appl.*, 17(1):1–34, 1996.
- [161] T. Ericsson. A generalised eigenvalue problem and the Lanczos algorithm. In J. K. Cullum and R. A. Willoughby, editors, *Large Scale Eigenvalue Problems*, pages 95–119. Elsevier Science Publishers (North-Holland), Amsterdam, 1986.
- [162] T. Ericsson and A. Ruhe. The spectral transformation Lanczos method for the numerical solution of large sparse generalized symmetric eigenvalue problems. *Math. Comp.*, 35:1251–1268, 1980.
- [163] V. Faber and T. A. Manteuffel. Necessary and sufficient conditions for the existence of a conjugate gradient method. *SIAM J. Numer. Anal.*, 21(2):352–362, 1984.

- [164] C. Farhat and M. Geradin. On a component mode synthesis method and its application to incompatible substructures. *Comput. & Structures*, 51(5):459–473, 1994.
- [165] H. Faßbender, D. S. Mackey, and N. Mackey. Hamilton and Jacobi come full circle: Jacobi algorithms for structured Hamiltonian eigenproblems. *To appear in Linear Algebra Appl.*, 2000.
- [166] P. Feldmann and R. W. Freund. Reduced-order modeling of large linear subcircuits via a block Lanczos algorithm. In *Proceedings of the 32nd Design Automation Conference*, pages 474–479. ACM, New York, 1995.
- [167] Y. T. Feng and D. R. J. Owen. Conjugate gradient methods for solving the smallest eigenpair of large symmetric eigenvalue problems. *Internat. J. Numer. Methods Engrg.*, 39(13):2209–2229, 1996.
- [168] K. V. Fernando and B. N. Parlett. Accurate singular values and differential qd algorithms. *Numer. Math.*, 67:191–229, 1994.
- [169] R. Fletcher. Conjugate gradient methods for indefinite systems. *Lecture Notes in Mathematics*, Vol. 506, pages 73–89. Springer-Verlag, Berlin, 1976.
- [170] R. Fletcher. *Practical Methods of Optimization*. Wiley, New York, second edition, 1987.
- [171] D. R. Fokkema. *Subspace Methods for Linear, Nonlinear, and Eigen Problems*. Ph.D. thesis, Utrecht University, Utrecht, the Netherlands, 1996.
- [172] D. R. Fokkema, G. L. G. Sleijpen, and H. A. van der Vorst. Jacobi-Davidson style QR and QZ algorithms for the partial reduction of matrix pencils. *SIAM J. Sci. Comput.*, 20:94–125, 1998.
- [173] R. W. Freund. Conjugate gradient-type methods for linear systems with complex symmetric coefficient matrices. *SIAM J. Sci. Statist. Comput.*, 13:425–448, 1992.
- [174] R. W. Freund. A transpose-free quasi-minimal residual algorithm for non-Hermitian linear systems. *SIAM J. Sci. Comput.*, 14:470–482, 1993.
- [175] R. W. Freund. Computing minimal partial realizations via a Lanczos-type algorithm for multiple starting vectors. In *Proceedings of the 36th IEEE Conference on Decision and Control*, pages 4394–4399. IEEE Press, Piscataway, NJ, 1997.
- [176] R. W. Freund. Reduced-order modeling techniques based on Krylov subspaces and their use in circuit simulation. In *Applied and Computational Control, Signals, and Circuits*, Vol. 1, pages 435–498. Birkhäuser, Boston, 1999.
- [177] R. W. Freund and P. Feldmann. Reduced-order modeling of large linear passive multi-terminal circuits using matrix-Padé approximation. In *Proceedings of the Design, Automation and Test in Europe Conference 1998*, pages 530–537. IEEE Computer Society Press, Los Alamitos, CA, 1998.
- [178] R. W. Freund, M. H. Gutknecht, and N. M. Nachtigal. An implementation of the look-ahead Lanczos algorithm for non-Hermitian matrices. *SIAM J. Sci. Comput.*, 14:137–158, 1993.

- [179] R. W. Freund and N. M. Nachtigal. QMR: A quasi-minimal residual method for non-Hermitian linear systems. *Numer. Math.*, 60:315–339, 1991.
- [180] R. W. Freund and N. M. Nachtigal. QMRPACK: A package of QMR algorithms. *ACM Trans. Math. Software*, 22:46–77, 1996.
- [181] S. Friedland, J. Nocedal, and M. L. Overton. The formulation and analysis of numerical methods for inverse eigenvalue problems. *SIAM J. Numer. Anal.*, 24:634–667, 1987.
- [182] C. Fu, X. Jiao, and T. Yang. Efficient sparse LU factorization with partial pivoting on distributed memory architectures. *IEEE Trans. Parallel and Distributed Systems*, 9(2):109–125, 1998. Software available at <http://www.cs.ucsb.edu/research/S+>.
- [183] Z. Fu and E. M. Dowling. Conjugate gradient eigenstructure tracking for adaptive spectral estimation. *IEEE Trans. Signal Processing*, 43(5):1151–1160, 1995.
- [184] K. Gallivan, E. Grimme, and P. Van Dooren. A rational Lanczos algorithm for model reduction. *Numer. Algorithms*, 12:33–64, 1996.
- [185] G. Gambolati, G. Pini, and M. Putti. Nested iterations for symmetric eigenproblems. *SIAM J. Sci. Comput.*, 16(1):173–191, 1995.
- [186] G. Gambolati, F. Sartoretto, and P. Florian. An orthogonal accelerated deflation technique for large symmetric eigenproblems. *Comput. Methods Appl. Mech. Engrg.*, 94(1):13–23, 1992.
- [187] F. Gantmacher. *The Theory of Matrices, Vols. I and II* (transl.). Chelsea, New York, 1959.
- [188] I. Garcia-Planas. Kronecker stratification of the space of quadruples of matrices. *SIAM J. Matrix Anal. Appl.*, 19(4):872–885, 1998.
- [189] G. Geist, G. Howell, and D. Watkins. The BR eigenvalue algorithm. *SIAM J. Matrix Anal. Appl.*, 20(4):1083–1098, 1999.
- [190] M. Genseberger and G. L. G. Sleijpen. Alternative correction equations in the Jacobi-Davidson method. Preprint 1073, Department of Mathematics, Utrecht University, Utrecht, the Netherlands, 1998.
- [191] A. George and J. Liu. *Computer Solution of Large Sparse Positive Definite Systems*. Prentice-Hall, Englewood Cliffs, NJ, 1981.
- [192] P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization*. Academic Press, New York, second edition, 1981.
- [193] I. Gohberg, T. Kailath, and V. Olshevsky. Fast gaussian elimination with partial pivoting for matrices with displacement structure. *Math. Comp.*, 64(212):1557–1576, 1995.
- [194] I. Gohberg, P. Lancaster, and L. Rodman. *Matrix Polynomials*. Academic Press, New York, 1982.

- [195] I. Gohberg and V. Olshevsky. Complexity of multiplication with vectors for structured matrices. *Linear Algebra Appl.*, 202:163–192, 1994.
- [196] I. Gohberg and V. Olshevsky. Fast algorithms with preprocessing for matrix-vector multiplication problems. *J. Complexity*, 10(4):411–427, 1994.
- [197] G. Golub and R. Underwood. The block Lanczos method for computing eigenvalues. In J. Rice, editor, *Mathematical Software III*, pages 364–377. Academic Press, New York, 1977.
- [198] G. Golub and C. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, third edition, 1996.
- [199] G. Golub and J. H. Wilkinson. Ill-conditioned eigensystems and the computation of the Jordan canonical form. *SIAM Rev.*, 18(4):578–619, 1976.
- [200] G. H. Golub, Z. Zhang, and H. Zha. Large sparse symmetric eigenvalue problems with homogeneous linear constraints: The Lanczos process with inner-outer iterations. *Linear Algebra Appl.*, 309:289–306, 2000.
- [201] W. B. Gragg. The QR algorithm for unitary Hessenberg matrices. *J. Comput. Appl. Math.*, 16:1–8, 1986.
- [202] W. B. Gragg and L. Reichel. A divide and conquer method for unitary and orthogonal eigenproblems. *Numer. Math.*, 57:695–718, 1990.
- [203] W. B. Gragg and T.-L. Wang. Convergence of the shifted QR algorithm for unitary Hessenberg matrices. Technical Report NPS-53-90-007, Naval Postgraduate School, Monterey, CA, 1990.
- [204] J. F. Grcar. *Analyses of the Lanczos Algorithm and the Approximation Problem in Richardson’s Method*. Ph.D. thesis, University of Illinois at Urbana-Champaign, 1981.
- [205] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *J. Comput. Phys.*, 73:325–348, 1987.
- [206] R. G. Grimes, J. G. Lewis, and H. D. Simon. A shifted block Lanczos algorithm for solving sparse symmetric generalized eigenproblems. *SIAM J. Matrix Anal. Appl.*, 15:228–272, 1994.
- [207] E. Grimme, D. Sorensen, and P. Van Dooren. Model reduction of state space systems via an implicitly restarted Lanczos method. *Numer. Algorithms*, 12:1–32, 1996.
- [208] M. Gu, J. Demmel, and I. Dhillon. Efficient computation of the singular value decomposition with applications to least squares problems. Computer Science Dept. Technical Report CS-94-257, University of Tennessee, Knoxville, 1994. LAPACK Working Note #88, <http://www.netlib.org/lapack/lawns/lawn88.ps>.
- [209] J.-S. Guo, W.-W. Lin, and C.-S. Wang. Numerical solutions for large sparse quadratic eigenvalue problems. *Linear Alg. Appl.*, 225:57–89, 1995.

- [210] A. Gupta, G. Karypis, and V. Kumar. Highly scalable parallel algorithms for sparse matrix factorization. *IEEE Trans. Parallel and Distributed Systems*, 8:502–520, 1997. Software available at <http://www.cs.umn.edu/~mjoshi/pspases>.
- [211] A. Gupta, E. Rothberg, E. Ng, and B. W. Peyton. Parallel sparse Cholesky factorization algorithms for shared-memory multiprocessor systems. In R. Vichnevetsky, D. Knight, and G. Richter, editors, *Advances in Computer Methods for Partial Differential Equations-VII*, pages 622–628. IMACS, New Brunswick, NJ, 1992.
- [212] I. Gustafsson. A class of first order factorization methods. *BIT*, 18:142–156, 1978.
- [213] M. H. Gutknecht. A completed theory of the unsymmetric Lanczos process and related algorithms, Part I. *SIAM J. Matrix Anal. Appl.*, 13:594–639, 1992.
- [214] M. H. Gutknecht. A completed theory of the unsymmetric Lanczos process and related algorithms, Part II. *SIAM J. Matrix Anal. Appl.*, 15:15–58, 1994.
- [215] W. Hackbusch. On the computation of approximate eigenvalues and eigenfunctions of elliptic operators by means of a multi-grid method. *SIAM J. Numer. Anal.*, 16(2):201–215, 1979.
- [216] W. Hackbusch. Multigrid solutions to linear and nonlinear eigenvalue problems for integral and differential equations. *Rostock. Math. Kolloq.*, (25):79–98, 1984.
- [217] W. Hackbusch. Multigrid eigenvalue computation. In *Advances in Multigrid Methods (Oberwolfach, 1984)*, pages 24–32. Vieweg, Braunschweig, Germany, 1985.
- [218] W. Hackbusch. *Multigrid Methods and Applications*. Springer-Verlag, Berlin, 1985.
- [219] M. Heath, E. Ng, and B. Peyton. Parallel algorithms for sparse linear systems. *SIAM Rev.*, 33:420–460, 1991.
- [220] M. T. Heath and P. Raghavan. Performance of a fully parallel sparse solver. *Internat. J. Supercomputer Appl.*, 11(1):49–64, 1997. Software available at <http://www.netlib.org/scalapack>.
- [221] R. Heeg. *Stability and Transition of Attachment-Line Flow*. Ph.D. thesis, Universiteit Twente, Enschede, the Netherlands, 1998.
- [222] S. Helgason. *Differential Geometry, Lie Groups, and Symmetric Spaces*. Academic Press, New York, 1978.
- [223] B. Hendrickson and R. Leland. The Chaco User’s Guide: Version 2.0. Technical Report SAND94-2692, Sandia National Laboratories, 1994.
- [224] P. Henon, P. Ramet, and J. Roman. A mapping and scheduling algorithm for parallel sparse fan-in numerical factorization. In P. Amestoy, P. Berger, M. Daydé,

- I. Duff, V. Frayssé, L. Giraud, and D. Ruiz, editors, *EuroPar'99 Parallel Processing*, Lecture Notes in Computer Science, Vol. 1685, pages 1059–1067. Springer-Verlag, New York, 1999.
- [225] M. R. Hestenes and W. Karush. Solutions of $Ax = \lambda Bx$. *J. Res. Nat. Bur. Standards*, 47:471–478, 1951.
- [226] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *J. Res. Nat. Bur. Standards*, 49:409–436, 1954.
- [227] V. Heuveline and M. Sadkane. Arnoldi-Faber method for large non-Hermitian eigenvalue problems. *Electron. Trans. Numer. Anal.*, 7:62–76, 1997.
- [228] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, Philadelphia, 1996.
- [229] N. J. Higham. QR factorization with complete pivoting and accurate computation of the SVD. *Linear Algebra Appl.*, 309:153–174, 2000.
- [230] N. J. Higham. Stability analysis of algorithms for solving confluent Vandermonde-like systems. *SIAM J. Matrix Anal. Appl.*, 11:23–41, 1990.
- [231] D. Hinrichsen and J. O'Halloran. Orbit closures of singular pencils. *J. Pure and Applied Algebra*, 81:117–137, 1992.
- [232] K. Hirao and H. Nakatsui. A generalization of the Davidson's method to large nonsymmetric eigenvalue problem. *J. Comput. Phys.*, 45(2):246–254, 1982.
- [233] R. A. Horn and C. R. Johnson. *Matrix Analysis*. Cambridge University Press, Cambridge, UK, 1985.
- [234] A. S. Householder. *The Theory of Matrices in Numerical Analysis*. Blaisdell, New York, 1964. Dover edition, 1975.
- [235] L. J. Huang and T.-Y. Li. Parallel homotopy algorithm for symmetric large sparse eigenproblems. *J. Comput. Appl. Math.*, 60(1-2):77–100, 1995.
- [236] S. A. Hutchinson, L. V. Prevost, J. N. Shadid, and R. S. Tuminaro. Aztec user's guide, version 2.0 Beta. Technical Report SAND95-1559, Sandia National Laboratories, Albuquerque, NM, 1998.
- [237] T. Hwang and I. D. Parsons. A multigrid method for the generalized symmetric eigenvalue problem. I. Algorithm and implementation. *Internat. J. Numer. Methods Engrg.*, 35(8):1663–1676, 1992.
- [238] T. Hwang and I. D. Parsons. A multigrid method for the generalized symmetric eigenvalue problem. II. Performance evaluation. *Internat. J. Numer. Methods Engrg.*, 35(8):1677–1696, 1992.
- [239] E.-J. Im. *Automatic Optimization of Sparse Matrix - Vector Multiplication*. Ph.D. thesis, University of California, Berkeley, May 2000.

- [240] E.-J. Im and K. A. Yelick. Optimizing sparse matrix vector multiplication on SMPs. In *Proceedings of the Ninth SIAM Conference on Parallel Processing for Scientific Computing*, SIAM, Philadelphia, 1999.
- [241] C. G. J. Jacobi. Ueber ein leichtes Verfahren, die in der Theorie der Säcularstörungen vorkommenden Gleichungen numerisch aufzulösen. *J. Reine Angew. Math.*, 30:51–94, 1846.
- [242] A. Jennings. *Matrix Computation for Engineers and Scientists*. Wiley, New York, 1977.
- [243] Z. Jia. A block incomplete orthogonalisation method for large nonsymmetric eigenproblems. *BIT*, 35:516–539, 1995.
- [244] Z. Jia. Polynomial characterizations of the approximate eigenvectors by the refined Arnoldi method and an implicitly restarted refined Arnoldi algorithm. *Linear Algebra Appl.*, 287:191–214, 1998.
- [245] Z. Jia. A refined iterative algorithm based on the block Arnoldi process for large unsymmetric eigenproblems. *Linear Algebra Appl.*, 270:171–189, 1998.
- [246] Z. Jia and G. W. Stewart. An analysis of the Rayleigh-Ritz method for approximating eigenspaces. Technical Report TR-4015, Department of Computer Science, University of Maryland, College Park, 1999.
- [247] Z. Jia and G. W. Stewart. On the convergence of Ritz values, Ritz vectors and refined Ritz vectors. Technical Report TR-3986, Department of Computer Science, University of Maryland, College Park, 1999.
- [248] P. Johansson. Stratigraph users' guide. version 1.1. Technical Report UMINF-99.11, Department of Computing Science, Umeå University, Umeå, Sweden, 1999.
- [249] M. T. Jones and M. L. Patrick. The Lanczos algorithm for the generalized symmetric eigenproblem on shared-memory architectures. *Appl. Numer. Math.*, 12:377–389, 1993.
- [250] B. Kågström. How to compute the Jordan normal form — the choice between similarity transformations and methods using the chain relations. Technical Report UMINF-91.81, Department of Numerical Analysis, Institute of Information Processing, University of Umeå, Umeå, Sweden, 1981.
- [251] B. Kågström. RGSVD—an algorithm for computing the Kronecker canonical form and reducing subspaces of singular $A - \lambda B$ pencils. *SIAM J. Sci. Statist. Comput.*, 7(1):185–211, 1986.
- [252] B. Kågström and A. Ruhe. ALGORITHM 560: An algorithm for the numerical computation of the Jordan normal form of a complex matrix [F2]. *ACM Trans. Math. Software*, 6(3):437–443, 1980.
- [253] B. Kågström and A. Ruhe. An algorithm for the numerical computation of the Jordan normal form of a complex matrix. *ACM Trans. Math. Software*, 6(3):389–419, 1980.

- [254] B. Kågström and P. Wiberg. Extracting partial canonical structure for large scale eigenvalue problems. Technical Report UMINF-98.13, Department of Computing Science, Umeå University, Umeå, Sweden, 1998. Submitted to *Numerical Algorithms*.
- [255] W. Kahan. Accurate eigenvalues of a symmetric tridiagonal matrix. Technical Report CS41, Computer Science Department, Stanford University, Stanford, CA, 1966 (revised June 1968).
- [256] W. Kahan, B. N. Parlett, and E. Jiang. Residual bounds on approximate eigen-systems of nonnormal matrices. *SIAM J. Numer. Anal.*, 19:470–484, 1982.
- [257] T. Kailath and A.H. Sayed, editors. *Fast Reliable Algorithms for Matrices with Structure*. SIAM, Philadelphia, 1999.
- [258] W. Karush. An iterative method for finding characteristics vectors of a symmetric matrix. *Pacific J. Math.*, 1:233–248, 1951.
- [259] G. Karypis and V. Kumar. Metis, Version 4.0. University of Minnesota/Army HPC Research Center, Mineeapolis, 1998.
- [260] H. M. Kim and R. R. Craig, Jr. Structural dynamics analysis using an unsymmetric block Lanczos algorithm. *Internat. J. Numer. Methods Engrg.*, 26:2305–2318, 1988.
- [261] H. M. Kim and R. R. Craig, Jr. Computational enhancement of an unsymmetric block Lanczos algorithm. *Internat. J. Numer. Methods Engrg.*, 30:1083–1089, 1990.
- [262] S. Kim and A. Chronopoulos. An efficient nonsymmetric Lanczos method on parallel vector computers. *J. Comput. Appl. Math.*, 42:357–374, 1992.
- [263] D. R. Kincaid, J. R. Respess, D. M. Young, and R. G. Grimes. Algorithm 586 – ITPACK 2C: A Fortran package for solving large sparse linear systems by adaptive accelerated iterative methods. *ACM Trans. Math. Software*, 8(3):302–322, 1982.
- [264] A. V. Knyazev. Computation of eigenvalues and eigenvectors for mesh problems: Algorithms and error estimates. Dept. of Numerical Math., USSR Academy of Sciences, Moscow, 1986. (In Russian.)
- [265] A. V. Knyazev. Convergence rate estimates for iterative methods for mesh symmetric eigenvalue problem. *Soviet J. Numer. Anal. Math. Modelling*, 2(5):371–396, 1987.
- [266] A. V. Knyazev. A preconditioned conjugate gradient method for eigenvalue problems and its implementation in a subspace. In *International Ser. Numerical Mathematics, v. 96, Eigenwertaufgaben in Natur- und Ingenieurwissenschaften und ihre numerische Behandlung, Oberwolfach*, 1990, pages 143–154, Birkhauser, Basel, 1991.
- [267] A. V. Knyazev. New estimates for Ritz vectors. *Math. Comp.*, 66(219):985–995, 1997.

- [268] A. V. Knyazev. Preconditioned eigensolvers - an oxymoron? *Electron. Trans. Numer. Anal.*, 7:104–123, 1998.
- [269] A. V. Knyazev. Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method. Technical Report UCD-CCM 149, Center for Computational Mathematics, University of Colorado, Denver, 2000. Available at <http://www-math.cudenver.edu/ccmreports/rep149.ps.gz>.
- [270] A. V. Knyazev and A. L. Skorokhodov. Preconditioned iterative methods in subspace for solving linear systems with indefinite coefficient matrices and eigenvalue problems. *Soviet J. Numer. Anal. Math. Modelling*, 4(4):283–310, 1989.
- [271] A. V. Knyazev and A. L. Skorokhodov. The preconditioned gradient-type iterative methods in a subspace for partial generalized symmetric eigenvalue problem. *Soviet Math. Dokl.*, 45(2):474–478, 1993.
- [272] A. V. Knyazev and A. L. Skorokhodov. The preconditioned gradient-type iterative methods in a subspace for partial generalized symmetric eigenvalue problem. *SIAM J. Numer. Anal.*, 31(4):1226–1239, 1994.
- [273] S. Kobayashi and K. Nomizu. *Foundations of Differential Geometry*. Wiley, New York, 1969.
- [274] L. Komzsik. *MSC/NASTRAN Numerical Methods User's Guide, Version 70.5*. The MacNeal-Schwendler Corporation, Los Angeles, 1998.
- [275] N. Kosugi. Modification of the Liu-Davidson method for obtaining one or simultaneously several eigensolutions of a large real symmetric matrix. *J. Comput. Phys.*, 55(3):426–436, 1984.
- [276] L. Kronecker. *Algebraische Reduction der Scharen Bilinearärer Formen*. S. B. Akad., Berlin, 1890.
- [277] V. N. Kublanovskaja. On an application to the solution of the generalized latent value problem for λ -matrices. *SIAM J. Numer. Anal.*, 7:532–537, 1970.
- [278] V. N. Kublanovskaya. On a method of solving the complete eigenvalue problem for a degenerate matrix (in Russian). *Zh. Vychisl. Mat. Mat. Fiz.*, 6:611–620, 1966. USSR Comput. Math. Phys., 6(4):1–16, 1968.
- [279] V. N. Kublanovskaya. An approach to solving the spectral problem of $A - \lambda B$. In B. Kågström and A. Ruhe, editors, *Matrix Pencils*, Lecture Notes in Mathematics, Vol. 973, pages 17–29. Springer-Verlag, Berlin, 1983.
- [280] V. N. Kublanovskaya. AB-algorithm and its modifications for the spectral problem of linear pencils of matrices. *Numer. Math.*, 43:329–342, 1984.
- [281] K. Kundert. Sparse matrix techniques. In Albert Ruehli, editor, *Circuit Analysis, Simulation and Design*. North-Holland, Amsterdam, 1986. Software available at <http://www.netlib.org/sparse>.

- [282] Yu. A. Kuznetsov. Iterative methods in subspaces for eigenvalue problems. In A. V. Balakrishnan, A. A. Dorodnitsyn, and J. L. Lions, editors, *Vistas in Applied Math., Numerical Analysis, Atmospheric Sciences, Immunology*, pages 96–113. Optimization Software, New York, 1986.
- [283] Y.-L. Lai, K.-Y. Lin, and W.-W. Lin. An inexact inverse iteration for large sparse eigenvalue problems. *Numer. Linear Algebra Appl.*, 4:425–437, 1997.
- [284] P. Lancaster. *Lambda-Matrices and Vibrating Systems*. Pergamon Press, Oxford, UK, 1966.
- [285] C. Lanczos. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *J. Res. Nat. Bur. Standards*, 45:255–282, 1950.
- [286] C. Lanczos. Solution of systems of linear equations by minimized iterations. *J. Res. Nat. Bur. Standards*, 49:33–53, 1952.
- [287] A. Laub. Invariant subspace methods for the numerical solution of Riccati equations. In S. Bittanti A. Laub, and J. C. Willems, editors, *Riccati Equations*. Springer-Verlag, New York, 1990.
- [288] C. Lawson, R. Hanson, D. Kincaid, and F. Krogh. Basic linear algebra subprograms for FORTRAN usage. *ACM Trans. Math. Software*, 5:308–325, 1979.
- [289] R. B. Lehoucq. *Analysis and Implementation of an Implicitly Restarted Arnoldi Iteration*. Ph.D. thesis, Rice University, Houston, TX, 1995.
- [290] R. B. Lehoucq and K. J. Maschhoff. Implementation of an implicitly restarted block Arnoldi method. Preprint MCS-P649-0297, Argonne National Laboratory, Argonne, IL, 1997.
- [291] R. B. Lehoucq and K. Meerbergen. Using generalized Cayley transformations within an inexact rational Krylov sequence method. *SIAM J. Matrix Anal. Appl.*, 20(1):131–148, 1998.
- [292] R. B. Lehoucq and J. A. Scott. An evaluation of software for computing eigenvalues of sparse nonsymmetric matrices. Technical Report MCS-P547-1195, Argonne National Laboratory, Argonne, IL, 1995.
- [293] R. B. Lehoucq and J. A. Scott. Implicitly restarted Arnoldi methods and eigenvalues of the discretized Navier-Stokes equations. Technical Report SAND97-2712J, Sandia National Laboratory, Albuquerque, NM, 1997.
- [294] R. B. Lehoucq and D. C. Sorensen. Deflation techniques within an implicitly restarted Arnoldi iteration. *SIAM J. Matrix Anal. Appl.*, 17:789–821, 1996.
- [295] R. B. Lehoucq, D. C. Sorensen, and C. Yang. *ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*. SIAM, Philadelphia, 1998.
- [296] A. S. Lewis and M. L. Overton. Eigenvalue optimization. In A. Iserles, editor, *Acta Numerica, Volume 5*, pages 149–190. Cambridge University Press, Cambridge, UK, 1996.

- [297] C.-K. Li and R. Mathias. The Lidskii-Mirsky-Wielandt theorem – additive and multiplicative versions. *Numer. Math.*, 81:377–413, 1999.
- [298] H. Li, P. Aitchison, and A. Woodbury. Methods for overcoming breakdown problems in the unsymmetric Lanczos reduction method. *Internat. J. Numer. Methods Engrg.*, 42:389–408, 1998.
- [299] R.-C. Li. On perturbations of matrix pencils with real spectra. *Math. Comp.*, 62:231–265, 1994.
- [300] R.-C. Li. Relative perturbation theory III: More bounds on eigenvalue variation. *Linear Algebra Appl.*, 266:337–345, 1997.
- [301] R. C. Li. Relative perturbation theory I: Eigenvalue and singular value variations. *SIAM J. Matrix Anal. Appl.*, 19:956–982, 1998.
- [302] R. C. Li. Relative perturbation theory II: Eigenspace and singular subspace variations. *SIAM J. Matrix Anal. Appl.*, 20:471–492, 1999.
- [303] R. C. Li. Relative perturbation theory IV: $\sin 2\theta$ theorems. *Linear Algebra Appl.*, 311:45–60, 2000.
- [304] T.-Y. Li and Z. Zeng. The Laguerre iteration in solving the symmetric tridiagonal eigenproblem, revisited. *SIAM J. Sci. Comput.*, 15:1145–1173, 1994.
- [305] T.-Y. Li and Z. Zeng. Homotopy continuation algorithm for the real nonsymmetric eigenproblem: Further development and implementation. *SIAM J. Sci. Comput.*, 20:1627–1651, 1999.
- [306] X. S. Li and J. W. Demmel. A scalable sparse direct solver using static pivoting. In *Proceedings of the Ninth SIAM Conference on Parallel Processing for Scientific Computing*, SIAM, Philadelphia, 1999. Software available at <http://www.nerc.gov/~xiaoye/SuperLU>.
- [307] W.-W. Lin, V. Mehrmann, and H. Xu. Canonical forms for Hamiltonian and symplectic matrices and pencils. *Linear Algebra Appl.*, 301/303:469–533, 1999.
- [308] R. Lucas. Private communication, 2000. Contact rflucas@lbl.gov.
- [309] S. H. Lui and G. H. Golub. Homotopy method for the numerical solution of the eigenvalue problem of self-adjoint partial differential operators. *Numer. Algorithms*, 10(3-4):363–378, 1995.
- [310] S. H. Lui, H. B. Keller, and T. W. C. Kwok. Homotopy method for the large, sparse, real nonsymmetric eigenvalue problem. *SIAM J. Matrix Anal. Appl.*, 18(2):312–333, 1997.
- [311] J.-C. Luo. Solving eigenvalue problems by implicit decomposition. *Numer. Methods Partial Differential Equations*, 7(2):113–145, 1991.
- [312] J.-C. Luo. A domain decomposition method for eigenvalue problems. In *Fifth International Symposium on Domain Decomposition Methods for Partial Differential Equations (Norfolk, VA, 1991)*, pages 306–321. SIAM, Philadelphia, 1992.

- [313] N. Mackey. Hamilton and Jacobi meet again - quaternions and the eigenvalue problem. *SIAM J. Matrix Anal. Appl.*, 16:421–435, 1995.
- [314] S. Yu. Maliassov. On the Schwarz alternating method for eigenvalue problems. *Russian J. Numer. Anal. Math. Modelling*, 13(1):45–56, 1998.
- [315] J. Mandel and S. McCormick. A multilevel variational method for $A\mathbf{u} = \lambda B\mathbf{u}$ on composite grids. *J. Comput. Phys.*, 80(2):442–452, 1989.
- [316] T. Manteuffel. The Tchebyshev iteration for nonsymmetric linear systems. *Numer. Math.*, 28:307–327, 1977.
- [317] R. Mathias. Accurate eigensystem computations by Jacobi methods. *SIAM J. Matrix Anal. Appl.*, 16:977–1003, 1995.
- [318] The MathWorks. *Partial Differential Equation Toolbox User's Guide*, The MathWorks, Natick, MA, 1995.
- [319] The MathWorks. *MATLAB User's Guide*, The MathWorks, Natick, MA, 1996.
- [320] S. F. McCormick. A mesh refinement method for $Ax = \lambda Bx$. *Math. Comp.*, 36(154):485–498, 1981.
- [321] S. F. McCormick. *Multilevel Projection Methods for Partial Differential Equations*. SIAM, Philadelphia, 1992.
- [322] K. Meerbergen. The rational Lanczos method for Hermitian eigenvalue problems. Technical Report RAL-TR-1999-025, Rutherford Appleton Laboratory, Chilton, UK, 1999. Available at <http://www.numerical.rl.ac.uk/reports/reports.html>.
- [323] K. Meerbergen and D. Roose. The restarted Arnoldi method applied to iterative linear system solvers for the computation of rightmost eigenvalues. *SIAM J. Matrix Anal. Appl.*, 18:1–20, 1997.
- [324] K. Meerbergen, A. Spence, and D. Roose. Shift-invert and Cayley transforms for detection of rightmost eigenvalues of nonsymmetric matrices. *BIT*, 34:409–423, 1994.
- [325] V. Mehrmann and D. Watkins. Structure-preserving methods for computing eigenpairs of large sparse skew-Hamiltonian/Hamiltonian pencils. Technical Report SFB393/00-02, Technische Universitaet Chemnitz, Germany, 2000.
- [326] J. Meijerink and H. A. van der Vorst. An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix. *Math. Comp.*, 31:148–162, 1977.
- [327] G. Meurant. *Computer solution of large linear systems*. North-Holland, Amsterdam, 1999.
- [328] C. B. Moler and G. W. Stewart. An algorithm for generalized matrix eigenvalue problems. *SIAM J. Numer. Anal.*, 10:241–256, 1973.

- [329] R. B. Morgan. Davidson's method and preconditioning for generalized eigenvalue problems. *J. Comput. Phys.*, 89:241–245, 1990.
- [330] R. B. Morgan. Theory for preconditioning eigenvalue problems. In *Proceedings of the Copper Mountain Conference on Iterative Methods*, 1990.
- [331] R. B. Morgan. Computing interior eigenvalues of large matrices. *Linear Algebra Appl.*, 154/156:289–309, 1991.
- [332] R. B. Morgan. Generalisations of Davidson's method for computing eigenvalues of large nonsymmetric matrices. *J. Comput. Phys.*, 101:287–291, 1992.
- [333] R. B. Morgan. On restarting the Arnoldi method for large nonsymmetric eigenvalue problems. *Math. Comp.*, 65:1213–1230, 1996.
- [334] R. B. Morgan. Implicitly restarted GMRES and Arnoldi methods for nonsymmetric systems of equations. *SIAM J. Matrix Anal. Appl.*, 21:1112–1135, 2000.
- [335] R. B. Morgan and D. S. Scott. Generalizations of Davidson's method for computing eigenvalues of sparse symmetric matrices. *SIAM J. Sci. Statist. Comput.*, 7:817–825, 1986.
- [336] R. B. Morgan and D. S. Scott. Preconditioning the Lanczos algorithm for sparse symmetric eigenvalue problems. *SIAM J. Sci. Comput.*, 14:585–593, 1993.
- [337] R. B. Morgan and M. Zeng. Harmonic projection methods for large non-symmetric eigenvalue problems. *Numer. Linear Algebra Appl.*, 5:33–55, 1998.
- [338] S. G. Nash and A. Sofer. *Linear and Nonlinear Programming*. McGraw-Hill, New York, 1995.
- [339] E. G. Ng and B. W. Peyton. Block sparse Cholesky algorithms on advanced uniprocessor computers. *SIAM J. Sci. Comput.*, 14(5):1034–1056, 1993.
- [340] B. Nour-Omid, B. N. Parlett, T. Ericsson, and P. S. Jensen. How to implement the spectral transformation. *Math. Comp.*, 48:663–673, 1987.
- [341] S. Oliveira. A convergence proof of an iterative subspace method for eigenvalues problems. In *Foundations of Computational Mathematics (Rio de Janeiro, 1997)*, pages 316–325. Springer-Verlag, Berlin, 1997.
- [342] S. Oliveira. On the convergence rate of a preconditioned subspace eigensolver. *Computing*, 63(3):219–231, 1999.
- [343] W. E. Olmstead, W. E. Davis, S. H. Rosenblat, and W. L. Kath. Bifurcation with memory. *SIAM J. Appl. Math.*, 40:171–188, 1986.
- [344] J. Olsen, P. Jørgensen, and J. Simons. Passing the one-billion limit in full configuration-interaction (FCI) calculations. *Chem. Phys. Lett.*, 169:463–472, 1990.
- [345] T.C. Oppe, W. Joubert, and D. Kincaid. NSPCG's user's guide: A package for solving large linear systems by various iterative methods. Technical report, University of Texas, Austin, TX, 1988.

- [346] E. E. Osborne. On pre-conditioning of matrices. *J. Assoc. Comput. Mach.*, 7:338–345, 1960.
- [347] C. C. Paige. *The Computation of Eigenvalues and Eigenvectors of Very Large Sparse Matrices*. Ph.D. thesis, London University, London, England, 1971.
- [348] C. C. Paige. Properties of numerical algorithms related to computing controllability. *IEEE Trans. Automat. Control*, AC-26(1):130–138, 1981.
- [349] C. C. Paige, B. N. Parlett, and H. A. van der Vorst. Approximate solutions and eigenvalue bounds from Krylov subspaces. *Numer. Linear Algebra Appl.*, 2:115–133, 1995.
- [350] C. C. Paige and M. A. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM J. Numer. Anal.*, 12:617–629, 1975.
- [351] C. C. Paige and M. A. Saunders. LSQR: An algorithm for sparse linear equations and sparse least squares. *ACM Trans. Math. Software*, 8:43–71, 1982.
- [352] C. C. Paige and C. F. Van Loan. A Schur decomposition for Hamiltonian matrices. *Linear Algebra Appl.*, 14:11–32, 1981.
- [353] B. N. Parlett. *The Symmetric Eigenvalue Problem*. Prentice-Hall, Englewood Cliffs, NJ, 1980. Reprinted as Classics in Applied Mathematics 20, SIAM, Philadelphia, 1997.
- [354] B. N. Parlett. Reduction to tridiagonal form and minimal realizations. *SIAM J. Matrix Anal. Appl.*, 13(2):567–593, 1992.
- [355] B. N. Parlett. The new qd algorithms. In *Acta Numerica*, pages 459–491. Cambridge University Press, Cambridge, UK, 1995.
- [356] B. N. Parlett. Invariant subspaces for tightly clustered eigenvalues of tridiagonals. *BIT*, 36:542–562, 1996.
- [357] B. N. Parlett and H. C. Chen. Use of indefinite pencils for computing damped natural modes. *Linear Algebra Appl.*, 140:53–88, 1990.
- [358] B. N. Parlett and I. S. Dhillon. Fernando’s solution to Wilkinson’s problem: an application of double factorization. *Linear Algebra Appl.*, 267:247–279, 1997.
- [359] B. N. Parlett and J. Le. Forward instability of tridiagonal QR. *SIAM J. Matrix Anal. Appl.*, 14:279–316, 1993.
- [360] B. N. Parlett and O. A. Marques. An implementation of the dqds algorithm (positive case). *Linear Algebra Appl.*, 309:217–259, 2000.
- [361] B. N. Parlett and C. Reinsch. Balancing a matrix for calculation of eigenvalues and eigenvectors. *Numer. Math.*, 13:293–304, 1969.
- [362] B. N. Parlett and Y. Saad. Complex shift and invert strategies for real matrices. *Linear Algebra Appl.*, 88/89:575–595, 1987.

- [363] B. N. Parlett and D. Scott. The Lanczos algorithm with selective orthogonalization. *Math. Comput.*, 33:217–238, 1979.
- [364] B. N. Parlett, D. R. Taylor, and Z. S. Liu. A look-ahead Lanczos algorithm for nonsymmetric matrices. *Math. Comp.*, 44:105–124, 1985.
- [365] W. V. Petryshyn. On the eigenvalue problem $Tu - \lambda Su = 0$ with unbounded and non-symmetric operators T and S . *Philos. Trans. Roy. Soc. Math. Phys. Sci.*, 262:413–458, 1968.
- [366] B. G. Pfrommer, J. Demmel, and H. Simon. Unconstrained energy functionals for electronic structure calculations. *J. Comput. Phys.*, 150(1):287–298, 1999.
- [367] A. Pokrzywa. On perturbations and the equivalence orbit of a matrix pencil. *Linear Algebra Appl.*, 82:99–121, 1986.
- [368] E. Polak. *Computational Methods in Optimization*. Academic Press, New York, 1971.
- [369] C. Pommerell. *Solution of Large Unsymmetric Systems of Linear Equations*. Ph.D. thesis, Swiss Federal Institute of Technology, Zürich, Switzerland, 1992.
- [370] E. Rothberg. *Exploiting the Memory Hierarchy in Sequential and Parallel Sparse Cholesky Factorization*. Ph.D. thesis, Dept. of Computer Science, Stanford University, Stanford, CA, 1992.
- [371] A. Ruhe. An algorithm for numerical determination of the structure of a general matrix. *BIT*, 10:196–216, 1970.
- [372] A. Ruhe. Algorithms for the nonlinear eigenvalue problem. *SIAM J. Numer. Anal.*, 10:674–689, 1973.
- [373] A. Ruhe. SOR-methods for the eigenvalue problem with large sparse matrices. *Math. Comp.*, 28:695–710, 1974.
- [374] A. Ruhe. Iterative eigenvalue algorithms based on convergent splittings. *J. Comput. Phys.*, 19:110–120, 1975.
- [375] A. Ruhe. Implementation aspects of band Lanczos algorithms for computation of eigenvalues of large sparse symmetric matrices. *Math. Comp.*, 33:680–687, 1979.
- [376] A. Ruhe. The rational Krylov algorithm for nonsymmetric eigenvalue problems, III: Complex shifts for real matrices. *BIT*, 34:165–176, 1994.
- [377] A. Ruhe. Eigenvalue algorithms with several factorizations – a unified theory yet? Technical Report 1998:11, Department of Mathematics, Chalmers University of Technology, Göteborg, Sweden, 1998.
- [378] A. Ruhe. Rational Krylov: A practical algorithm for large sparse nonsymmetric matrix pencils. *SIAM J. Sci. Comput.*, 19(5):1535–1551, 1998.

- [379] A. Ruhe and D. Skoogh. Rational Krylov algorithms for eigenvalue computation and model reduction. In B. Kågström, J. Dongarra, E. Elmroth, and J. Waśniewski, editors, *Applied Parallel Computing. Large Scale Scientific and Industrial Problems.*, volume 1541 of *Lecture Notes in Computer Science*, pages 491–502, 1998.
- [380] A. Ruhe and T. Wiberg. The method of conjugate gradients used in inverse iteration. *BIT*, 12:543–554, 1972.
- [381] H. Rutishauser. Computational aspects of F. L. Bauer’s simultaneous iteration method. *Numer. Math.*, 13:4–13, 1969.
- [382] H. Rutishauser. Simultaneous iteration method for symmetric matrices. *Numer. Math.*, 16:205–223, 1970. Also in [458, pp. 284–301].
- [383] Y. Saad. Chebyshev acceleration techniques for solving nonsymmetric eigenvalue problems. *Math. Comp.*, 42(166):567–588, 1984.
- [384] Y. Saad. Least squares polynomials in the complex plane and their use for solving nonsymmetric linear systems. *SIAM J. Numer. Anal.*, 24(1):155–169, 1987.
- [385] Y. Saad. Numerical solution of large nonsymmetric eigenvalue problems. *Comp. Phys. Comm.*, 53:71–90, 1989.
- [386] Y. Saad. SPARSKIT: A basic tool-kit for sparse matrix computation, version 2, 1994. Software available at <http://www.cs.umn.edu/~saad>
- [387] Y. Saad. *Numerical Methods for Large Eigenvalue Problems*. Halsted Press, New York, 1992.
- [388] Y. Saad. *Iterative Methods for Linear Systems*. PWS Publishing, Boston, 1996.
- [389] Y. Saad and M. H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 7:856–869, 1986.
- [390] M. Sadkane. Block-Arnoldi and Davidson methods for unsymmetric large eigenvalue problems. *Numer. Math.*, 64:195–211, 1993.
- [391] M. Sadkane. A block Arnoldi-Chebyshev method for computing the leading eigenpairs of large sparse unsymmetric matrices. *Numer. Math.*, 64:181–193, 1993.
- [392] A. H. Sameh and J. A. Wisniewski. A trace minimization algorithm for the generalized eigenvalue problem. *SIAM J. Numer. Anal.*, 19:1243–1259, 1982.
- [393] B. A. Samokish. The steepest descent method for an eigenvalue problem with semi-bounded operators. *Izv. Vuzov Math.*, 5:105–114, 1958. (In Russian.)
- [394] G. V. Savinov. Investigation of the convergence of a generalized method of conjugate gradients for determining the extremal eigenvalues of a matrix. *Zap. Nauchn. Sem. Leningrad. Otdel. Mat. Inst. Steklov. (LOMI)*, 111:145–150, 1981. (In Russian.)

- [395] O. Schenk, K. G  rtner, and W. Fichtner. Efficient sparse LU factorization with left-right looking strategy on shared memory multiprocessors. *BIT*, 40(1):158–176, 2000.
- [396] W. H. A. Schilders. Personal communication, September 1997.
- [397] D. S. Scott. Solving sparse symmetric generalised eigenvalue problems without factorisation. *SIAM J. Numer. Anal.*, 18:102–110, 1981.
- [398] J. A. Scott. An Arnoldi code for computing selected eigenvalues of sparse unsymmetric matrices. *ACM Trans. Math. Software*, 21:432–475, 1995.
- [399] N. S. Sehmi. A Newtonian procedure for the solution of the Kron characteristic value problem. *J. Sound Vibration*, 100(3):409–421, 1985.
- [400] N. S. Sehmi. *Large order structural eigenanalysis techniques*. Ellis Horwood Series: Mathematics and Its Applications. Ellis Horwood, Chichester, UK, 1989.
- [401] V. A. Shishov. A method for partitioning a high order matrix into blocks in order to find its eigenvalues. *USSR Comput. Math. and Math. Phys.*, 1(1):186–190, 1961.
- [402] H. Simon. Analysis of the symmetric Lanczos algorithm with reorthogonalization methods. *Linear Algebra Appl.*, 61:101–132, 1984.
- [403] H. Simon. The Lanczos algorithm with partial reorthogonalization. *Math. Comp.*, 42:115–142, 1984.
- [404] A. Simpson and B. Tabarrok. On Kron's eigenvalue procedure and related methods of frequency analysis. *Quart. J. Mech. Appl. Math.*, 21:1–39, 1968.
- [405] J. P. Singh, W.-D. Webber, and A. Gupta. Splash. Stanford parallel applications for shared-memory. *Computer Architecture News*, 20(1):5–44, 1992. Software available at <http://www-flash.stanford.edu/apps/SPLASH>.
- [406] I. Slapnicar. *Accurate Symmetric Reduction by a Jacobi Method*. Ph.D. thesis, Fernuniversit  t Gesamthochschule Hagen, Germany, 1992.
- [407] I. Slapnicar. Accurate computation of singular values and eigenvalues of symmetric matrices. *Math. Commun.*, 1:153–168, 1996.
- [408] G. L. G. Sleijpen, G. L. Booten, D. R. Fokkema, and H. A. van der Vorst. Jacobi Davidson type methods for generalized eigenproblems and polynomial eigenproblems. *BIT*, 36:595–633, 1996.
- [409] G. L. G. Sleijpen and D. R. Fokkema. Bi-CGSTAB(ℓ) methods for linear equations involving matrices with complex spectrum. *Electron. Trans. Numer. Anal.*, 1:11–32, 1993.
- [410] G. L. G. Sleijpen, D. R. Fokkema, and H. A. van der Vorst. BiCGSTAB(ℓ) and other hybrid Bi-CG methods. *Numer. Algorithms*, 7:75–109, 1994.
- [411] G. L. G. Sleijpen and H. A. van der Vorst. A Jacobi–Davidson iteration method for linear eigenvalue problems. *SIAM J. Matrix Anal. Appl.*, 17:401–425, 1996.

- [412] G. L. G. Sleijpen, H. A. van der Vorst, and E. Meijerink. Efficient expansion of subspaces in the Jacobi-Davidson method for standard and generalized eigenproblems. *Electron. Trans. Numer. Anal.*, 7:75–89, 1998.
- [413] G. L. G. Sleijpen, H. A. van der Vorst, and M. B. van Gijzen. Quadratic eigenproblems are no problem. *SIAM News*, 29:8–9, 1996.
- [414] P. Smit and M. H. C. Paardekooper. The effects of inexact linear solvers in algorithms for symmetric eigenvalue problems. *Linear Algebra Appl.*, 287:337–357, 1998.
- [415] B. F. Smith, P. E. Bjørstad, and W. D. Gropp. *Domain decomposition*. Cambridge University Press, Cambridge, UK, 1996.
- [416] S. T. Smith. Optimization techniques on Riemannian manifolds. *Fields Inst. Commun.*, 3:113–146, 1994.
- [417] E. Snapper and R. Troyer. *Metric Affine Geometry*. Academic Press, New York, 1971.
- [418] P. Sonneveld. CGS: A fast Lanczos-type solver for nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 10:36–52, 1989.
- [419] D. C. Sorensen. Implicit application of polynomial filters in a k -step Arnoldi method. *SIAM J. Matrix Anal. Appl.*, 13:357–385, 1992.
- [420] D. C. Sorensen. Deflation for implicitly restarted Arnoldi methods. Technical Report TR98-12, Department of Computational and Applied Mathematics, Rice University, Houston, TX, 1998.
- [421] A. Stathopoulos, Y. Saad, and K. Wu. Dynamic thick restarting of the Davidson, and the implicitly restarted Arnoldi methods. *SIAM J. Sci. Comput.*, 19:227–245, 1998.
- [422] G. W. Stewart. Simultaneous iteration for computing invariant subspaces of non-Hermitian matrices. *Numer. Math.*, 25:123–136, 1976.
- [423] G. W. Stewart. Perturbation bounds for the definite generalized eigenvalue problem. *Linear Algebra Appl.*, 23:69–86, 1979.
- [424] G. W. Stewart. Computing the CS decomposition of a partitioned orthogonal matrix. *Numer. Math.*, 40:297–306, 1982.
- [425] G. W. Stewart and J.-G. Sun. *Matrix Perturbation Theory*. Academic Press, New York, 1990.
- [426] W. J. Stewart and A. Jennings. Algorithm 570: LOPSI a simultaneous iteration method for real matrices. *ACM Trans. Math. Software*, 7:230–232, 1981.
- [427] W. J. Stewart and A. Jennings. A simultaneous iteration algorithm for real matrices. *ACM Trans. Math. Software*, 7:184–198, 1981.

- [428] I. Štich, R. Car, M. Parrinello, and S. Baroni. Conjugate gradient minimization of the energy functional: A new method for electronic structure calculation. *Phys. Rev. B.*, 39:4997–5004, 1989.
- [429] E. Suetomi and H. Sekimoto. Conjugate gradient like methods and their application to eigenvalue problems for neutron diffusion equation. *Annals of Nuclear Energy*, 18(4):205, 1991.
- [430] J.-G. Sun. Perturbation bounds for eigenspaces of a definite matrix pair. *Numer. Math.*, 41:321–343, 1983.
- [431] J. G. Sun. Stability and accuracy: Perturbation analysis of algebraic eigenproblems. Technical Report UMINF 98.07, Department of Computing Science, Umeå University, Umeå, Sweden, 1998.
- [432] J. G. Sun. Perturbation analysis of quadratic eigenvalue problems. *BIT*, 1999, submitted.
- [433] D. B. Szyld and O. B. Widlund. Applications of conjugate gradient type methods to eigenvalue calculations. In *Advances in Computer Methods for Partial Differential Equations, III (Proc. Third IMACS Internat. Sympos., Lehigh Univ., Bethlehem, Pa., 1979)*, pages 167–173. IMACS, New Brunswick, NJ, 1979.
- [434] D. R. Taylor. *Analysis of the Look-Ahead Lanczos Algorithm*. Ph.D. thesis, University of California, Berkeley, 1982.
- [435] F. Tisseur. Backward error and condition of polynomial eigenvalue problems. *Linear Algebra Appl.*, 309:339–361, 2000.
- [436] F. Tisseur. Stability of structured Hamiltonian eigensolvers. Numerical Analysis Report No. 357, Manchester Centre for Computational Mathematics, Manchester, UK, February 2000.
- [437] F. Tisseur and N. J. Higham. Structured pseudospectra for polynomial eigenvalue problems, with applications. Numerical Analysis Report No. 359, Manchester Centre for Computational Mathematics, Manchester, UK, 2000.
- [438] S. Toledo. Improving instruction-level parallelism in sparse matrix-vector multiplication using reordering, blocking, and prefetching. In *Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing*. SIAM, Philadelphia, 1997.
- [439] S. Toledo. Improving the memory-system performance of sparse-matrix vector multiplication. *IBM J. Res. Develop.*, 41(6):711–726, 1997.
- [440] L. N. Trefethen. Computation of pseudospectra. In A. Iserles, editor, *Acta Numerica*, Volume 8, pages 247–295. Cambridge University Press, Cambridge, MA, 1999.
- [441] L. N. Trefethen. Spectra and pseudospectra: The behavior of non-normal matrices and operators. In J. Levesley, M. Ainsworth, and M. Marletta, editors, *The Graduate Student’s Guide to Numerical Analysis*, Volume 26. Springer-Verlag, Berlin, 2000.

- [442] C. Trefftz, C. C. Huang, P. K. Mckinley, T.-Y. Li, and Z. Zeng. A scalable eigenvalue solver for symmetric tridiagonal matrices. *Parallel Computing*, 21:1213–1240, 1995.
- [443] H. van der Veen and C. Vuik. Bi-Lanczos with partial orthogonalization. *Computers and Structures*, 56:605–613, 1995.
- [444] H. A. van der Vorst. A generalized Lanczos scheme. *Math. Comp.*, 39:559–561, 1982.
- [445] H. A. van der Vorst. Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of non-symmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 13:631–644, 1992.
- [446] P. Van Dooren. The computation of Kronecker’s canonical form of a singular pencil. *Linear Algebra Appl.*, 27:103–141, 1979.
- [447] P. Van Dooren. The generalized eigenstructure problem in linear system theory. *IEEE Trans. Automat. Control*, AC-26(1):111–129, 1981.
- [448] P. Van Dooren. A generalized eigenvalue approach for solving Riccati equations. *SIAM J. Sci. Comput.*, 2:121–135, 1981.
- [449] P. Van Dooren. Algorithm 590, DUSBSP and EXCHQZ: FORTRAN subroutines for computing deflating subspaces with specified spectrum. *ACM Trans. Math. Software*, 8:376–382, 1982.
- [450] P. Van Dooren. Reducing subspaces: Computational aspects and applications in linear systems theory. In *Proceedings of the 5th Int. Conf. on Analysis and Optimization of Systems*, 1982, Lecture Notes on Control and Information Sciences. Volume 44. Springer-Verlag, New York, 1983.
- [451] P. Van Dooren. Reducing subspaces: Definitions, properties and algorithms. In B. Kågström and A. Ruhe, editors, *Matrix Pencils*, Lecture Notes in Mathematics, Volume 973, pages 58–73. Springer-Verlag, Berlin, 1983.
- [452] C. F. Van Loan. A symplectic method for approximating all the eigenvalues of a Hamiltonian matrix. *Linear Algebra Appl.*, 61:233–251, 1984.
- [453] C. F. Van Loan. *Computational Frameworks for the Fast Fourier Transform*. SIAM, Philadelphia, 1992.
- [454] E. L. Wachspress. *Iterative solution of elliptic systems, and applications to the neutron diffusion equations of reactor physics*. Prentice-Hall, Englewood Cliffs, NJ, 1966.
- [455] H. F. Walker. Implementation of the GMRES method using Householder transformations. *SIAM J. Sci. Statist. Comput.*, 9:152–163, 1988.
- [456] W. Waterhouse. The codimension of singular matrix pairs. *Linear Algebra Appl.*, 57:227–245, 1984.
- [457] J. H. Wilkinson. *The Algebraic Eigenvalue Problem*. Clarendon Press, Oxford, UK, 1965.

- [458] J. H. Wilkinson and C. Reinsch. *Handbook for Automatic Computation. Vol. II, Linear Algebra*. Springer-Verlag, New York, 1971.
- [459] Y.-C. Wong. Differential geometry of Grassmann manifolds. *Proc. Nat. Acad. Sci. USA*, 57:589–594, 1967.
- [460] M. Wonham. *Linear Multivariable Control Theory: A Geometric Approach*. Springer-Verlag, New York, second edition, 1979.
- [461] K. Wu, Y. Saad, and A. Stathopoulos. Inexact Newton preconditioning techniques for eigenvalue problems. Technical Report Technical Report LBNL-41382, Lawrence Berkeley National Laboratory, Berkeley, CA, 1998. Also published as Minnesota Super Computer Centre report number UMSI 98-10, Minneapolis.
- [462] K. Wu and H. D. Simon. A parallel Lanczos method for symmetric generalized eigenvalue problems. Technical Report LBNL-41284, National Energy Research Scientific Computing Division, Lawrence Berkeley National Laboratory, Berkeley, CA, 1997. Software available at <http://www.nerc.gov/research/SIMON/planso.html>.
- [463] K. Wu and H. D. Simon. Dynamic restarting schemes for eigenvalue problems. Technical Report LBNL-42982, National Energy Research Scientific Computing Division, Lawrence Berkeley National Laboratory, Berkeley, CA, 1999.
- [464] H. Yang. Conjugate gradient methods for the Rayleigh quotient minimization of generalized eigenvalue problems. *Computing*, 51(1):79–94, 1993.
- [465] Q. Ye. A breakdown-free variation of the nonsymmetric Lanczos algorithms. *Math. Comp.*, 62:179–207, 1994.
- [466] H. Zha and H. Simon. On updating problems in latent semantic indexing. *SIAM J. Sci. Comput.*, 21:782–791, 1999.
- [467] H. Zha and Z. Zhang. On matrices with low-rank-plus-shift structures: partial SVD and latent semantic indexing. *SIAM J. Matrix Anal. Appl.*, 21:522–280, 1999.
- [468] T. Zhang, G. H. Golub, and K. H. Law. Subspace iterative methods for eigenvalue problems. *Linear Algebra Appl.*, 294(1-3):239–258, 1999.
- [469] T. Zhang, K. H. Law, and G. H. Golub. On the homotopy method for perturbed symmetric generalized eigenvalue problems. *SIAM J. Sci. Comput.*, 19(5):1625–1645, 1998.
- [470] S. Zhou and H. Dai. *Dai Shu Te Zheng Zhi Fan Wen Ti (The Algebraic Inverse Eigenvalue Problems)*. Henan Science and Technology Press, Zhengzhou, China, 1991. (In Chinese.)
- [471] Z. Zlatev, J. Waśniewski, P. C. Hansen, and Tz. Ostromsky. PARASPAR: A package for the solution of large linear algebraic equations on parallel computers with shared memory. Technical Report 95-10, Technical University of Denmark, Lyngby, September 1995.

- [472] P. I. Davies, N. J. Higham, and F. Tisseur. Analysis of the Cholesky Method with Iterative Refinement for Solving the Symmetric Definite Generalized Eigenproblem. Manchester Centre for Computational Mathematics, Manchester, England, Numerical Analysis Report 360, 2000.
- [473] D. J. Higham and N. J. Higham. Structured backward error and condition of generalized eigenvalue problems. *SIAM J. Matrix Anal. Appl.*, 20:493–512, 1998.

Index

- ABLE, 199
- accuracy assessment
 - of GHEP, 127
 - of GNHEP, 277
 - of HEP, 105
 - of NHEP, 228
 - of SVD, 137
- adaptive blocking, 202
- Arnoldi factorization, 167
- Arnoldi method
 - basic, 161
 - block, 185
 - implicitly restarted, 166
 - with inexact Cayley transform, 343
- Arnoldi procedure, 161, 167
- Arnoldi vector, 161, 167
- ARPACK
 - for GHEP, 122
 - for HEP, 80
 - for NHEP, 184
 - for SVD, 145
- backward error analysis, 10
 - of GHEP, 129, 132
 - of GNHEP, 278
 - of HEP, 105
 - of NHEP, 229
- balancing
 - accuracy of eigenvalues, 155
 - dense matrix, 153
 - direct, dense matrix, 153
 - direct, sparse matrix, 153
 - iterative, 154
- bidiagonalization
 - for dense matrices, 138
 - for sparse matrices, 142
- biorthogonality
 - full, 199
- loss of, 193
- semi-, 199
- bisection method
 - for GHEP, 113
 - for HEP, 50
 - for SVD, 139
- BLAS, 320
 - for structured matrices, 323
 - sparse, 321
- breakdown
 - of band Lanczos method, 207
 - of block Lanczos method, 199
 - of complex symmetric Lanczos method, 218
 - of Lanczos method, 192
 - of symmetric indefinite Lanczos method, 255
- Cayley transform, 340
 - for QEP, 286
- characteristic polynomial, 11, 15, 23
 - 28
- chordal metric, 31, 131, 278
- companion matrix, 27
 - block, 27, 289
- complex symmetric eigenproblem, 216
- condition number, 8, 10
 - of eigenvalue, GNHEP, 279
 - of eigenvalue, NHEP, 230
- conditioning
 - of GHEP, 16
 - of GNHEP, 31, 35
 - of HEP, 12, 26
 - of SVD, 20
- congruence transformation, 15
- conjugate gradient method
 - covariant, 307
 - preconditioned, for GHEP, 362

- convergence properties
 - Davidson method, 343
 - of inexact Cayley, for GNHEP, 343
 - of inverse iteration, for HEP, 53
 - of inverse iteration, for NHEP, 159
 - of IRAM, 172
 - of IRLM, 71
 - of Lanczos method, for GHEP, 121
 - of Lanczos method, for HEP, 59
 - of Lanczos method, for NHEP, 193
 - of power method, for HEP, 52
 - of power method, for NHEP, 159
 - of preconditioned power method, 356
 - of preconditioned subspace iteration, 363
 - of subspace iteration, for HEP, 54
- covariant
 - conjugate gradient method, 307
 - differentiation, 312
 - Newton's method, 307
 - Stiefel–Grassmann gradient, 310
- Crawford number, 131
- CS (cosine/sine) decomposition, 22
- Davidson method, 89
 - generalized, 344
- definite (matrix) pencil, 14
- deflating subspace, 29
- deflation
 - band Lanczos method, for HEP, 81
 - band Lanczos method, for NHEP, 206
 - block Arnoldi method, 188
 - IRAM, 175
 - IRLM, 73
 - Jacobi–Davidson method, for GNHEP, 240
 - Jacobi–Davidson method, for HEP, 95
 - Jacobi–Davidson method, for NHEP, 224
 - subspace iteration, for HEP, 55
 - subspace iteration, for NHEP, 159
- diagonal form
 - of GNHEP, 30
 - of NHEP, 24
- differentiation
 - covariant, 312
- direct methods
 - for GHEP, 113
 - for GNHEP, 234
 - for HEP, 49
 - for NHEP, 157
 - for SVD, 138
- divide-and-conquer method
 - for GHEP, 114
 - for HEP, 50
 - for SVD, 139
- domain decomposition methods, 371
- DQDS algorithm, 139
- eigendecompositions
 - of GHEP, 15
 - of GNHEP, 29, 35
 - of HEP, 12
 - of NHEP, 24
- eigenproblems
 - inverse, 370
 - modified, 370
 - of parameterized matrices, 369
 - of structured matrices
 - Hamiltonian, 369
 - quaternion, 369
 - symplectic, 369
 - of unitary matrices, 369
 - updating, 370
- eigenspace, 8
- eigenvalue optimization, 370
- equivalence transformations, 19, 29, 34
- error bound
 - of eigenvalue, GHEP, 129, 132
 - of eigenvalue, GNHEP, 279
 - of eigenvalue, HEP, 106
 - of eigenvalue, NHEP, 230
 - of eigenvalues, singular pencil, 272
 - of eigenvector, GHEP, 130, 132
 - of eigenvector, GNHEP, 279
 - of eigenvector, HEP, 106
 - of eigenvector, NHEP, 230

Galerkin condition, 39
gap, 13, 106, 129, 137, 271
generalized Schur-staircase form, 35
generalized Hermitian eigenproblem
(GHEP), 14, 109
generalized non-Hermitian
eigenproblem (GNHEP), 28,
233
generalized Schur decomposition, 31
generalized Schur-staircase form, 266
Golub–Kahan–Lanczos method, 142
gradient
Stiefel–Grassmann, 310
Gram–Schmidt process
classical, 168
modified, 92
modified, with refinement, 93
two-sided, 193, 213
GUPTRI, 261

harmonic Ritz value, 41, 100
Hermitian definite pencil, 109, 131
Hermitian eigenproblem (HEP), 11, 45
Hessenberg matrix, 162
Hessenberg reduction, 158
high relative accuracy eigensolvers, 370
homotopy continuation, 370

ill-conditioning, 9
of singular pencil, 264
implicitly restarted Arnoldi method
(IRAM), 170
implicitly restarted Lanczos method
(IRLM), 68
inexact method
Arnoldi method, 343
Lanczos method, 346
rational Krylov method, 348
invariant subspace, 12
inverse eigenproblems, 370
inverse iteration
for GHEP, 114
for HEP, 50, 52
for NHEP, 159
for SVD, 139

Jacobi method
for HEP, 51
for SVD, 140

Jacobi–Davidson method
Cayley transform, 346
for GHEP, 123
for GNHEP, 238
for HEP, 88
for NHEP, 221
for QEP, 287
Jordan (canonical) form, 25
Jordan structure
nearest, 295
nearest, sg_min example, 299
Jordan–Schur form, 26

Kronecker (canonical) form, 35, 262
Krylov subspace, 39

Lanczos factorization
for HEP, 67
for NHEP, 190
Lanczos method
band, for HEP, 80
band, for NHEP, 205
basic, for HEP, 56
basic, for NHEP, 189
block, for NHEP, 196
for complex symmetric
eigenproblem, 216
for GHEP, 116
for SVD, 142
implicitly restarted, 67
in GEMV form, 70
preconditioned, 346
preconditioned, for GHEP, 359
symmetric indefinite, for GNHEP,
249
Lanczos vector, 67, 190
LAPACK
for GHEP, 113
for GNHEP, 234
for HEP, 49
for NHEP, 157
for SVD, 138
linear least squares problem, 147
linear solver
direct, 326
for band matrices, 328
for dense matrices, 328
for sparse matrices, 329

- linear solver (*continued*)
 - direct (*continued*)
 - for structured matrices, 331
 - iterative, 332
- linearization
 - of PEP, 289
 - of QEP, 283
- locking
 - IRAM, 176
 - IRLM, 74
 - subspace iteration, 54
- look-ahead technique, 192, 195, 220
- manifolds, 309
- MATLAB
 - for GHEP, 113
 - for GNHEP, 234
 - for HEP, 49
 - for NHEP, 157
 - for SVD, 138
- matrix sign function, 371
- multigrid methods, 371
- multilevel methods, 371
- nearest Jordan structure, 295
 - by sg_min, 295
 - sg_min example, 299
- Newton's method
 - covariant, 307
- non-Hermitian eigenproblem (NHEP), 23, 149
- nonlinear eigenproblem (NLEP), 36, 281
- numerical rank, 271
- numerical stability, 10
 - of IRAM, for NHEP, 173
 - of Lanczos method, for NHEP, 189
 - of QR algorithm, 158
- numerical stability assessment
 - of GHEP, 127
 - of GNHEP, 277
 - of HEP, 105
 - of NHEP, 228
- oblique projection, 40
- optimization
 - sg_min, 291
- orthogonal projection, 39
- orthogonality
 - full, 58
 - local, 58
 - loss of, 58
 - of Arnoldi vectors, 163, 168
 - selective, 58
- orthogonality constraints, 290
- parallelism, 334–336
 - inner products, 334
 - matrix-vector products, 335
 - solver, 336
 - vector updates, 335
- Petrov–Galerkin condition, 38, 40
- polynomial acceleration, 55
- polynomial eigenproblem (PEP), 289
- power method
 - for GHEP, 114
 - for HEP, 51
 - for NHEP, 159
 - preconditioned, 356
- preconditioned
 - Lanczos method, 359
 - power method, 356
 - RQI, 361
 - steepest ascent method, 358
 - subspace iteration, 363
- preconditioned eigensolvers, 337
- preconditioning
 - left-, 93
 - right-, 93
- Procrustes problem, 294
 - by sg_min, 294
 - sg_min example, 298
- projection method, 38
 - approximate problem, 39
 - oblique, 39
 - orthogonal, 39
 - refined, 43
- projection process, 37
- pseudospectra, 369
- purging
 - in IRAM, 176
 - in IRLM, 74
- QR algorithm
 - for GHEP, 113
 - for HEP, 50

- for NHEP, 157
- for SVD, 139
- quadratic eigenproblem (QEP), 281
- quadratic Rayleigh quotient, 282
- quotient SVD (QSVD), 21, 146
- QZ algorithm, 234
- rational Krylov method, 246
 - inexact, 348
- Rayleigh quotient, 53
 - Stiefel–Grassmann optimization, 295
 - by sg_min, 295
 - sg_min example, 300
- Rayleigh quotient iteration (RQI), 53
 - for GHEP, 115
 - preconditioned, 361
- Rayleigh–Ritz procedure, 39
- reduced-order modeling, 214
- reducing subspace, 34
- refined projection method, 43
- regular matrix pencil, 261
- relatively robust representation
 - algorithm
 - for HEP, 51
- reorthogonalization
 - of Lanczos method, for GHEP, 118
 - of Lanczos method, for HEP, 61
- residual vector
 - of GHEP, 129, 131
 - of GNHEP, 278
 - of HEP, 105
 - of NHEP, 229
 - of QEP, 282
 - of SVD, 137
- restart
 - of Arnoldi method, explicit, 163
 - of Arnoldi method, implicit, 169
 - of block Arnoldi method, 188
 - of Jacobi–Davidson method, for GNHEP, 240
 - of Jacobi–Davidson method, for HEP, 95
 - of Jacobi–Davidson method, for NHEP, 221
 - of Lanczos method, implicit, 67
- Ritz
 - harmonic, value, 41
 - value, 40
 - vector, 40
- roots of polynomial, 27
- ScalAPACK
 - for HEP, 49
 - for NHEP, 157
 - for SVD, 138
- Schur decomposition (form), 25
 - generalized, 31
 - generalized, partial, 239
 - partial, 166, 225
 - simultaneous, 296
 - simultaneous, sg_min example, 302
- sg_min
 - modifying, 304
 - nearest Jordan structure problem, 295
- Procrustes problem, 294
- Rayleigh quotient, 295
 - simultaneous Schur decomposition, 296
- shift selection
 - for IRLM, 69
- shift-and-invert, 44, 340
 - for GNHEP, 237
 - for QEP, 285
 - inexact, 352
- IRAM, for NHEP, 183
- Lanczos method, for GHEP, 119
- Lanczos method, for HEP, 60
- Lanczos method, for NHEP, 193
- rational Krylov, 246
 - symmetric indefinite Lanczos, 252
- similarity transformation, 12, 24
- simultaneous Schur decomposition
 - by sg_min, 296
- singular (matrix) pencil, 34, 261
- singular subspace, 19
- singular value decomposition (SVD), 18, 135
 - compact, 19, 135
 - generalized, 21, 146
 - more generalized, 146
- partial, 20

- singular value decomposition (SVD)
 - (*continued*)
 - thin, 19, 135
 - truncated, 19, 135
 - updating, 370
- sparse matrix storage, 315–319
 - BCRS, 316–317
 - CCS, 316
 - CDS, 317
 - CRS, 315–316
 - JDS, 318–319
 - SKS, 319
- spectral transformation, 43
 - for QEP, 284
 - Lanczos method, for HEP, 60
- Stiefel–Grassmann optimization, 290
 - sg_min, 291
- subspace iteration
 - for HEP, 54
- for NHEP, 159
- preconditioned, 363
- subspace of approximants, 38
- symmetric definite pencil, 109
 - preconditioned solver, 352
- symmetric indefinite matrix pencil, 249
- transfer function, 214
- transformation to standard problem
 - GHEP, 112
 - GNHEP, 235
- updating
 - eigenproblems, 370
 - the SVD, 370
- Weierstrass form, 30
- Weierstrass–Schur form, 31