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

(Draft, May 21, 1999)

Edited by

Z. Bai, J. Demmel, J. Dongarra,

A. Ruhe, and H. van der Vorst

List of Contributors

Bai, Zhaojun, Department of Mathematics, University of Kentucky, Lexington, KY 40506, U.S.A.
bai@ms.uky.edu

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

Day, David, Sandia National Laboratories PO Box 5800, Albuquerque, NM 87185, U.S.A.
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, U.S.A.
demmel@cs.berkeley.edu

Dongarra, Jack J., Department of Computer Science, University of Tennessee, Knoxville, TN 37996 and Mathematical Science Section Oak Ridge National Laboratory Oak Ridge, TN 37821, U.S.A.
dongarra@cs.utk.edu

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

Ericsson, Thomas, Chalmers Tekniska Hvgskola, Matematiska institutionen, S-412 96 Gvteborg, Sweden
thomas@cs.chalmers.se

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

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

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

Kowalski, Thomas, Department of Mathematics, University of Kentucky, Lexington, KY 40506, U.S.A.
kowalski@ms.uky.edu

Knyazev, Andrew, Department of Mathematics, University of Colorado at Denver, P.O. Box 173364, Campus Box 170, Denver, CO 80217, U.S.A.
aknyazev@math.cudenver.edu

Lehoucq, Richard B., Sandia National Laboratories PO Box 5800, Albuquerque, NM 87185, U.S.A.
rlehoucq@cs.sandia.gov

Li, Ren-cang, Department of Mathematics, University of Kentucky, Lexington, KY 40506, U.S.A.
rcli@ms.uky.edu

Lippert, Ross Sandia National Laboratories, PO Box 5800, Albuquerque, NM 87185, U.S.A.
ripper@cs.sandia.gov

Maschhoff, Kristi Tera Computer Company, Merrill Place, 411 First Ave South, Ste 600, Seattle, WA 98104-2860, U.S.A.
kristyn@tera.com

Meerbergen, Karl, Rutherford Appleton Laboratory, Chilton, Didcot, OX11 0QX, United Kingdom
K.Meerbergen@rl.ac.uk

Morgan, Ronald, Department of Mathematics, Baylor University, ...
morganr@baylor.edu

Ruhe, Axel, Chalmers Tekniska Hvgskola, Matematiska institutionen, S-412 96 Gvteborg, Sweden
ruhe@cs.chalmers.se

Saad, Yousef, Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN 55455, U.S.A.
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, U.S.A.
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

Acknowledgements

- Alan Edelman and Ross Lippert would like to thank Steve Smith for many invaluable discussions.
- Please send your "acknowledgement" statements for the support of your work to Bai, we will edit them and put them here.

Preface

Contents

List of Figures **14**

1 Introduction **17**
1.1 Why Eigenvalue Templates? . . . 17
1.2 Intended Readership . . . 19
1.3 Related Work . . . 19
1.4 How to Use this Book . . . 20

2 A Brief Tour of Eigenvalue Problems **21**
2.1 Introduction . . . 21
2.2 Basic Definitions
J. Demmel . . . 21
2.3 Desired Spectral Information and Eigendecompositions
J. Demmel . . . 23
2.4 Problem Representation and Available Operations
J. Demmel . . . 26
2.5 Overview of Available Algorithms
J. Demmel . . . 29
2.6 Which Algorithm/Template Meets My Needs? . . . 31
2.6.1 Criteria for choice of algorithm . . . 31
2.7 Basic Ideas of Iterative Projection Methods
Y. Saad . . . 33
2.7.1 Introduction . . . 33
2.7.2 Orthogonal Projection Methods . . . 34
2.7.3 Oblique Projection Methods . . . 35
2.7.4 Ritz and Harmonic Ritz Values . . . 36
2.8 Shift-and-invert Spectral Transformation
R. Lehoucq and D. Sorensen . . . 38

3 Hermitian Eigenvalue Problem **41**
3.1 Introduction . . . 41
3.1.1 Overview of Available Algorithms . . . 42
3.1.2 Summary of choices . . . 43
3.2 Transformation Methods . . . 44
3.3 Single and Multiple Vector Iterations
M. Gu . . . 46
3.3.1 Power Method . . . 46

3.3.2 Inverse Iteration . . . 47
3.3.3 Rayleigh Quotient Iteration . . . 48
3.3.4 Orthogonal Iteration . . . 48
3.3.5 Practical Subspace Iteration . . . 49
3.3.6 Software Availability . . . 51
3.4 Lanczos Method
A. Ruhe . . . 51
3.4.1 Algorithm . . . 52
3.4.2 Convergence properties . . . 54
3.4.3 Spectral transformation . . . 55
3.4.4 Reorthogonalization . . . 55
3.4.5 Software Availability . . . 57
3.4.6 Numerical Examples . . . 57
3.5 Implicitly Restarted Lanczos Method
R. Lehoucq and D. Sorensen . . . 61
3.5.1 Implicit Restart . . . 61
3.5.2 Shift selection . . . 64
3.5.3 Lanczos in GEMV form . . . 65
3.5.4 Convergence properties . . . 66
3.5.5 Computational Costs and Tradeoffs . . . 67
3.5.6 Deflation and Stopping Rules . . . 68
3.5.7 An Orthogonal Deflating Transformation . . . 69
3.5.8 Locking or Purging a Single Eigenvalue . . . 71
3.5.9 Stability of Q^*TQ . . . 72
3.5.10 Implementation of Locking and Purging . . . 75
3.5.11 Software Availability . . . 76
3.6 Band Lanczos Method
R. Freund . . . 77
3.6.1 The Need for Deflation . . . 77
3.6.2 Basic Properties . . . 78
3.6.3 The Algorithm . . . 81
3.6.4 Variants . . . 83
3.7 Jacobi-Davidson methods
G. Sleijpen and H. van der Vorst . . . 85
3.7.1 Basic Theory . . . 85
3.7.2 The Basic Algorithm . . . 87
3.7.3 Jacobi-Davidson with restart and deflation . . . 91
3.7.4 Jacobi-Davidson for interior Eigenvalues . . . 96
3.7.5 Software Availability . . . 102

4 Generalized Hermitian Eigenproblem 103
4.1 Introduction . . . 103
4.2 Transformation Methods . . . 103
4.3 Single and Multiple Vector Iterations Methods . . . 103
4.4 Lanczos Method . . . 103
4.5 Implicity Restarted Lanczos Method . . . 103
4.6 Jacobi-Davidson Method . . . 103

5 Non-Hermitian Eigenvalue Problem **105**
5.1 Introduction . . . 105
5.2 Transformation Methods . . . 108
5.3 Single and Multiple Vector Iterations
M. Gu . . . 109
5.3.1 Power Method . . . 109
5.3.2 Inverse Iteration . . . 109
5.3.3 Subspace Iteration with Projection . . . 109
5.3.4 Practical Implementations . . . 110
5.3.5 Software Availability . . . 111
5.4 Arnoldi's Method
Y. Saad . . . 112
5.4.1 The Basic Algorithm . . . 112
5.4.2 Orthogonality . . . 114
5.4.3 Explicit Restarts . . . 114
5.4.4 Deflation . . . 115
5.5 The Implicitly Restarted Arnoldi Method
R. Lehoucq and D. Sorensen . . . 117
5.5.1 Introduction . . . 117
5.5.2 The Arnoldi Process in GEMV Form . . . 118
5.5.3 The Restarting Idea . . . 121
5.5.4 Implicit Restarting . . . 121
5.5.5 Numerical Stability of IRAM . . . 125
5.5.6 Computational Costs and Tradeoffs . . . 126
5.5.7 Deflation and Stopping Rules . . . 127
5.5.8 An Orthogonal Deflating Transformation . . . 128
5.5.9 Locking or Purging a Single Eigenvalue . . . 129
5.5.10 Stability of Q^*HQ . . . 131
5.5.11 Locking and Purging in IRAM . . . 134
5.5.12 Software Availability . . . 137
5.6 Block Arnoldi Method
R. Lehoucq and K. Maschhoff . . . 138
5.6.1 Block Arnoldi Reductions . . . 138
5.6.2 Practical Algorithm . . . 140
5.7 Lanczos Method
Z. Bai and D. Day . . . 142
5.7.1 Algorithm . . . 142
5.7.2 Convergence properties . . . 146
5.7.3 Multiple eigenvalues . . . 148
5.7.4 Software Availability . . . 148
5.7.5 Notes and References . . . 148
5.8 Block Lanczos Methods
Z. Bai and D. Day . . . 149
5.8.1 Basic Algorithm . . . 149
5.8.2 An Adaptively Blocked Lanczos Method . . . 152
5.8.3 Software Availability . . . 157
5.8.4 Notes and References . . . 157

5.9 Band Lanczos Method
R. Freund . . . 158
5.9.1 Deflation . . . 159
5.9.2 Basic Properties . . . 159
5.9.3 The Algorithm . . . 163
5.9.4 Application to Reduced-Order Modeling . . . 167
5.9.5 Variants . . . 168
5.9.6 Software Availability . . . 169
5.10 Lanczos Method for Complex Symmetric Eigenproblems
R. Freund . . . 169
5.10.1 Properties of Complex Symmetric Matrices . . . 169
5.10.2 Properties of the Algorithm . . . 170
5.10.3 The Algorithm . . . 172
5.10.4 Solving the reduced eigenvalue problems . . . 172
5.10.5 Software Availability . . . 173
5.10.6 Notes and References . . . 174
5.11 Jacobi-Davidson methods
G. Sleijpen and H. van der Vorst . . . 174
5.11.1 Generalization of Hermitian Case . . . 174
5.11.2 Schur Form and Restart . . . 174
5.11.3 Interior Eigenvalues . . . 177
5.11.4 Software Availability . . . 181

6 Generalized Non-Hermitian Eigenproblem 183
6.1 Introduction . . . 183
6.2 Transformation Methods . . . 183
6.3 Single and Multiple Vector Iterations Methods . . . 183
6.4 Arnoldi Method . . . 183
6.5 Rational Krylov Subspace Method . . . 183
6.6 Lanczos Method . . . 183
6.7 Indefinite Symmetric Lanczos Method . . . 183
6.8 Jacobi-Davidson Method . . . 183

7 Singular Value Decomposition and its Generalization 185

8 More on Generalized Eigenvalue Problems 187
8.1 Introduction . . . 187
8.2 Singular Pencils
B. Kågström . . . 188
8.2.1 Regular versus singular problems . . . 188
8.2.2 Kronecker canonical form . . . 189
8.2.3 Generic and non-generic Kronecker structures . . . 191
8.2.4 Ill-conditioning of eigenproblems . . . 191
8.2.5 Generalized Schur-staircase form . . . 193
8.2.6 GUPTRI algorithm . . . 194
8.2.7 Software Availability . . . 199
8.2.8 Notes and References . . . 204

8.3 Quadratic Eigenvalue Problems
Z. Bai, G. Sleijpen and H. van der Vorst . . . 205
8.3.1 Introduction . . . 205
8.3.2 Reduction to Linear Form . . . 206
8.3.3 Spectral Transformations for QEP . . . 207
8.3.4 Numerical Methods for Solving Linearized Problems . . . 209
8.3.5 The Jacobi-Davidson method . . . 210
8.3.6 Notes and References . . . 212
8.4 Nonlinear Eigenvalue Problems with Orthogonality Constraints
A. Edelman and R. Lippert . . . 212
8.4.1 Introduction . . . 212
8.4.2 Matlab Templates . . . 213
8.4.3 Sample Problems and Their Differentials . . . 215
8.4.4 Numerical Examples . . . 219
8.4.5 Modifying the Templates . . . 226
8.4.6 Geometric Technicalities . . . 231

9 Common Issues **237**
9.1 Accuracy Assessment and Numerical Stability
Z. Bai and R. Li . . . 237
9.1.1 Introduction . . . 237
9.1.2 Standard eigenvalue problem . . . 238
9.1.3 Generalized eigenvalue problem, I . . . 240
9.1.4 Generalized eigenvalue problem, II . . . 242
9.1.5 Errors in Computed Eigenvectors . . . 244
9.2 Balancing Matrices
T. Chen and J. Demmel . . . 245
9.2.1 Introduction . . . 245
9.2.2 Direct Balancing . . . 246
9.2.3 Balancing Algorithms Which Use Krylov Information . . . 247
9.2.4 Accuracy of Eigenvalues Computed after Balancing . . . 247
9.3 Data Structures
J. Dongarra . . . 249
9.3.1 Sparse Matrix Storage Formats . . . 250
9.3.2 Matrix vector products . . . 255
9.4 Dense and Sparse BLAS
J. Dongarra . . . 257
9.5 A Brief Survey of Direct Linear Solvers
J. Demmel . . . 258
9.6 A Brief Survey of Iterative Linear Solvers
H. van der Vorst . . . 258
9.7 Parallelism
J. Dongarra . . . 260
9.7.1 Inner products . . . 261
9.7.2 Vector updates . . . 261
9.7.3 Matrix-vector products . . . 261

10 Preconditioning Techniques **263**
10.1 Introduction 263
10.2 Inexact Methods
K. Meerbergen and R. Morgan 265
10.2.1 Matrix Transformations 266
10.2.2 Inexact matrix transformations 266
10.2.3 The inexact Cayley transform Arnoldi method 268
10.2.4 The Davidson method 269
10.2.5 The Jacobi-Davidson method and the Cayley transform 271
10.2.6 The preconditioned Lanczos method 272
10.2.7 The inexact rational Krylov method 274
10.2.8 Inexact shift-and-invert 278
10.3 Preconditioned Eigensolvers
A. Knyazev 278
10.3.1 Introduction 278
10.3.2 General Framework of Preconditioning 280
10.3.3 The Preconditioned Shifted Power Method 282
10.3.4 The Preconditioned Steepest Ascent/Descent Methods 283
10.3.5 Preconditioned Lanczos Methods 284
10.3.6 The Davidson Method 286
10.3.7 Methods with Preconditioned Inner Iterations 287
10.3.8 The Preconditioned Conjugate Gradient Methods 288
10.3.9 Preconditioned Simultaneous Iterations 289
10.3.10 Software Availability 293

11 Of Things Not Treated **295**

List of Figures

3.1 Residual estimates, L-membrane matrix, $n = 2945$. 59
3.2 Ritz values, L-membrane matrix, $n = 2945$. 59
3.3 Residual estimates, Medline SVD example 60
3.4 Residual estimates, Shifted and inverted L-membrane matrix, $n = 2945$. 61
3.5 Jacobi-Davidson for exterior eigenvalues with several strategies for solving the correction equation. 100
3.6 Jacobi-Davidson for exterior eigenvalues (top figures) and interior eigenvalues (bottom figures). The correction equations have been solved with 5 steps of plain GMRES (left figures) and with 5 steps of preconditioned GMRES (right figures). 101

5.1 Jacobi-Davidson for exterior eigenvalues (left figure) and interior eigenvalues (right figure). 180

8.1 Output (in short format) from the `guptri` function in Matlab. 202
8.2 Procrustes Problem 221
8.3 Jordan Problem 222
8.4 Trace Minimization Problem 223
8.5 LDA Toy Problem 224
8.6 Simultaneous Schur Problem 225
8.7 Simultaneous Diagonalization Problem 226
8.8 Our code tree: dependencies of various modules 227
8.9 The unconstrained differential of $F(Y)$ can be projected to the tangent space to obtain the covariant gradient, G, of F. 233
8.10 In a flat space, comparing vectors at nearby points is not problematic since all vectors lie in the same tangent space. 235
8.11 In a curved manifold, comparing vectors at nearby points can result in vectors which do not lie in the tangent space. 235

9.1 Plots examining the relative accuracy of eigenvalues computed with and without direct balancing for the qh768 and tolosa matrices. 248
9.2 Plots examining the relative accuracy of eigenvalues computed with and without Krylov balancing for the qh768 and tolosa matrices. 248
9.3 Plots comparing the relative accuracy of the largest and smallest (in magnitude) eigenvalues computed with the different Krylov-based balancing algorithms for the qh768 matrix. We use the default settings for the number of iterations and the cutoff value. 249

9.4 Plots comparing the relative accuracy of the largest and smallest (in magnitude) eigenvalues computed with the different Krylov-based balancing algorithms for the tolosa matrix. We use the default settings for the number of iterations and the cutoff value. 250
9.5 Profile of a nonsymmetric skyline or variable-band matrix. 255

10.1 Logarithmic plots of residual norms of the inexact rational Krylov method for the Olmstead problem. The circles denote $\|f_j\|$ and the bullets $\|r_j\|$ 277
10.2 CG vs. SA, $\delta_1/\delta_0 = 10$ 290
10.3 CG vs. SA, $\delta_1/\delta_0 = 100$ 291
10.4 CG vs. SA, $\delta_1/\delta_0 = 1000$ 292

List of Notation

Will be updated soon

The following is a short list of conventional notations we should try to use throughout writing.

$A, \ldots, Z$	matrices
$a, \ldots, z$	vectors
$\alpha, \beta, \ldots, \omega$	scalars
A^T	matrix transpose
A^H	conjugate transpose of A
A^{-1}	matrix inverse
A^{-T}	the inverse of A^T
a_{ij}	matrix element of A
$a_{i,:}$	ith row of matrix A
$a_{:,j}$	jth column of matrix A
(x, y), $x^T y$	vector dot product (inner product)
v_j	vector v in the jth iteration
$\mathrm{diag}(A)$	diagonal of matrix A
$\mathrm{diag}(\alpha, \beta \ldots)$	diagonal matrix constructed from scalars $\alpha, \beta \ldots$
$\mathrm{span}(a, b \ldots)$	spanning space of vectors $a, b \ldots$
$\mathcal{R}$, $\mathcal{C}$	sets of real and complex numbers
$\mathcal{R}^n$, $\mathcal{C}^n$	real and complex n-spaces
$\|x\|_p$ $\|A\|_2$	vector and matrix p-norm
$\|x\|_A$	the "A-norm", defined as $(Ax, x)^{1/2}$
$\|A\|_F$	matrix Frobenius-norm
$\lambda(A)$	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
$\kappa_2(A)$	spectral condition number of matrix A
$\overline{\alpha}$	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

Extra

A^* the conjugate transpose of A. If A is real, just transpose

$\mathcal{K}^m(A, v)$ Krylov subspace

$$\mathcal{K}^m = \mathrm{span}\{v, Av, A^2 v, \ldots, A^{m-1} v\}$$

by Axel: For me v_j is the jth column of the basis v while I like to use superscripts to denote which iteratino for a quantity that changes in each iteration.

CHAPTER 1

Introduction

1.1 Why Eigenvalue Templates?

Large scale problems of engineering and scientific computing often require solutions of linear algebra problems. This book specializes on the important subclass consisting of eigenvalue problems. Half a century ago, it might have been sufficient to mumble something about forming a characteristic polynomial and then computing its roots. This book is a testament of how far the field has evolved. The traditional mathematicians' method was never practical numerically. The time is ripe to organize descriptions of the many modern methods which are practical including advice on how to choose among the methods wisely.

A computational scientist interested in solving advanced eigenvalue problems might have to search among books, the research literature, and on-line or library software. The search can be daunting. Software may be found in well-maintained libraries available commercially or publicly, other libraries distributed with texts or other books, individual subroutines tested and published by organizations like the ACM, and yet more software available from individuals on individual web pages, or publicly maintained pages such as Netlib[1]. Sometimes the software may be hard to find or come without support. Though we admit that since challenging numerical linear algebra problems still await satisfactory solutions, many excellent methods exist from a plethora of sources.

But the sheer number of algorithms and their implementations makes it hard even for experts, let alone general users, to find the best solution for a given problem. This has led to the development of various on-line search facilities for numerical software. One has been developed by NIST (National Institute of Standards and Technology), and is called GAMS (Guide to Available Mathematical Software) [2]; another is part of Netlib. These facilities permit search based on library names, subroutines names, keywords, and a taxonomy of topics in numerical computing. But for the general user

[1] http://www.netlib.org

[2] `http://gams.nist.gov`

in search of advice as to which algorithm or which subroutine to use for her particular problem, they offer relatively little advice.

Question by Bai: should we mention the Linear system template project here instead of in section 1.3?

Furthermore, many challenging problems cannot be solved with existing "black-box" software packages in a reasonable time or space. This means that more special purpose methods must be used, and tuned for the problem at hand. This tuning is the greatest challenge, since there are a large number of tuning options available, and for many problems it is a challenge to get any acceptable answer at all, or have confidence in what is computed. The expertise regarding which options, or combinations of options, is likely to work in a specific application area, is widely distributed.

Thus, there is a need for tools to help users pick the best algorithm and implementation for their numerical problems, as well as expert advice on how to tune them.

For us a template should include

1. A high-level description of an algorithm.

2. A description of when it 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.

3. A description of available refinements and user-tunable parameters, as well as advice on when to use them.

4. Pointers to complete or partial implementations, perhaps in several languages or for several architectures (such as different parallel architecture). These implementation expose those details suitable for user-tuning, and hide the others.

5. Numerical examples, on a common set of examples, illustrating both easy cases and difficult cases.

6. Troubleshooting advice.

7. Pointers to texts or journal articles for further information.

In addition to individual templates, a *decision tree* is provided to help steer the user to the right algorithm, or subset of possible algorithms, based on a sequence of questions about the nature of the problem to be solved.

We do not intend to write a "cookbook", and have deliberately avoided the words "numerical recipes", because these phrases imply that our algorithms can be used blindly without knowledge of the system of equations. The state of the art in iterative methods does not permit this: some knowledge about the linear system is needed to guarantee convergence of these algorithms, and generally the more that is known the more the algorithm can be tuned. Thus, we have chosen to present an algorithmic outline, with guidelines for choosing a method and implementing it on particular kinds of high-performance machines. We also discuss the use of preconditioners and relevant data storage issues.

1.2 Intended Readership

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

- Students and teachers who want simple but generally effective algorithms, 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 scientists trying to solve the largest, hardest applications in their fields. Such users desire high speed, access to algorithmic details for performance tuning, and reliability for their problem.

It may seem difficult to address the needs of such diverse communities with a single document. Nevertheless, we believe this is possible in the form of *templates*. We will guide the reader confronted with an eigenvalue problem on how to give some consideration to

- the structure of the problem
- what is desired from the solution
- and the kind of computer available to solve it (perhaps).

In return, we provide

- model software to solve the problem,
- some feeling of the length of time to solve the problem
- a way to assess the accuracy (perhaps).

It is our hope that users from all areas will be able to use our templates either as is, or with custom modifications for important science and engineering problems.

1.3 Related Work

Question by Bai: Do we need this section? some of them overlaps with the discussion in section 1

Many excellent numerical linear algebra texts [?, ?, ?, ?, ?] and black-box software libraries [?, ?, ?] already exist. A great deal of more specialized software for eigenproblems also exists, for example, for surveys on the subject see [?, ?].

A book of templates, including software, has already been written for iterative methods for solving linear systems [?], although it does not include all the ingredients mentioned above. In particular, it discusses some important advanced methods, such as preconditioning, domain decomposition and multigrid, relatively briefly, and does not have a comprehensive set of numerical examples to help explain the expected

performance from each algorithm on various problem classes. It has a relatively simple decision tree to help users choose an algorithm.

The linear algrbra chapter in Numerical Recipes [?] contains a brief description of the conjugate gradient algorithm for sparse linear systems and the eigensystem chapter only contains the basic method for solving dense standard eigenvalue problem, all of the methods and available software in the recipes, except the Jacobi method for symmetric eigenvalue problem, are simplified version of algorithms in the EISPACK and LAPACK. Beside on disk, the software in the recipes are actually printed line by line in the book.

Other related work: GAMS, PETSc (no eigensolvers)

Dense Linear Algebra Libraries (EISPACK, LINPACK, LAPACK, ScaLAPACK should be here?)

The following paragraph is sugguested by Henk:

The variety of eigenproblems that is covered in this book is very wide, some algorithms are well-known and have been thouroughly investigated - especially for Hermitian standard problems - other areas are still subject to intensive research and therefore some of the algorithms reflect only current insight. This has led to differences in the style of presentation. On one side of the spectrum we have sections for standard problems that have been written for non-expert readers primarily, whereas at the other extreme we see texts with of a higly research character. For most sections we mention explicitly the contributing authors; this is not only for credit reasons, it also reflects personal responsability for the text. All sections provide sufficient references as to serve as windows to a richer world of detail and expertise on eigenproblems.

1.4 How to Use this Book

- The organization of the rest of book,
- **Suggestion:** For the first time reader, it is strongly suggested to read through Chapter 2 to acquire some basic definitions and classifications of algebraic eigenvalue problems.
- ...

Question: should we point out the different presentation style of the templates in this book? black-box, partial exposure, and full exposure?

CHAPTER 2

A Brief Tour of Eigenvalue Problems

2.1 Introduction

The whole chapter will be revised by Jim.

2.2 Basic Definitions

J. Demmel

In this section, we will talk mostly about eigenproblems of the form

$$(A - \lambda B)x = 0. \tag{2.1}$$

$A - \lambda B$ is called a *matrix pencil*. λ is an indeterminate in this latter expression, and indicates that there are two matrices which define the eigenproblem. A and B need not be square.

For almost all fixed scalar values of λ, the rank of the matrix $A - \lambda B$ will be constant. The discrete set of values of λ for which the rank of $A - \lambda B$ is lower than this constant are the *eigenvalues*. Let $\lambda_1, ..., \lambda_k$ be the discrete set of eigenvalues. Some λ_i may be infinite, in which case we really consider $\mu A - B$ with eigenvalue $\mu_i = 0$. Nonzero vectors x_i and y_i such that

$$Ax_i = \lambda_i B x_i \tag{2.2}$$

and

$$y_i^H A = \lambda_i y_i^H B, \tag{2.3}$$

are called a *right eigenvector* and a *left eigenvector*, respectively, where A^T is the transpose of A and A^H is its conjugate-transpose. The word *eigenvector* alone will mean right eigenvector.

Since there are many kinds of eigenproblems, and associated algorithms, we propose some simple top level categories to help classify them. The ultimate decision

tree presented to the reader will begin with easier concepts and questions about the eigenproblem in an attempt to classify it, and proceed to harder questions. For the purposes of this overview, we will use rather more advanced categories in order to be brief but precise. For background, see [94, 101, 48].

Regular and singular pencils: the pencil $A - \lambda B$ is *regular* if A and B are square and $det(A - \lambda B)$ is not identically zero for all λ; otherwise it is *singular*.

Regular pencils have well-defined sets of eigenvalues which change continuously as functions of A and B; this is a minimal requirement to be able to compute the eigenvalues accurately, in the absence of other constraints on A and B. Singular pencils have eigenvalues which can change *discontinuously* as functions of A and B; extra information about A and B, as well as special algorithms which use this information, are necessary in order to compute meaningful eigenvalues. Regular and singular pencils have correspondingly different *canonical forms* representing their spectral decompositions. The *Jordan Canonical Form* of a single matrix is the best known; the *Kronecker Canonical Form* of a singular pencil is the most general. More will be discussed in section 2.3 below.

Self-adjoint and Non-self-adjoint Eigenproblems: We abuse notation to avoid confusion with the very similar but less general notions of Hermitian and non-Hermitian: We call an eigenproblem $A - \lambda B$ is *self-adjoint* if

1) A and B are both Hermitian, and

2) there is a nonsingular X such that $XAX^H = \Lambda_A$ and $XBX^H = \Lambda_B$ are real and diagonal.

Thus the finite eigenvalues are real, all elementary divisors are linear, and the only possible singular blocks in the Kronecker Canonical Form represent a common null space of A and B. The primary source of self-adjoint eigenproblems is eigenvalue problems in which B is known to be positive definite; in this case $A - \lambda B$ is called a *definite pencil*. These properties lead to generally simpler and more accurate algorithms. We classify the singular value decomposition (SVD) and its generalizations as self-adjoint, because of the relationship between the SVD of A and the eigendecomposition of the Hermitian matrix $\begin{bmatrix} 0 & A \\ A^H & 0 \end{bmatrix}$.

The above description is very terse. The user will see a more extended description of each problem, including common synonyms.

1. Hermitian eigenproblem $A - \lambda I$, where $A = A^*$ (Chapter 3)

2. Non-Hermitian eigenproblem $A - \lambda I$, where $A \neq A^*$ (Chapter 5)

3. Generalized definite eigenproblem $A - \lambda B$, where $A = A^*$, $B = B^*0$ and one of A, B and $\alpha A + \beta B$ is positive definite. (Chapter 4)

4. Generalized non-Hermitian Eigenproblem $A - \lambda B$, (Chapter 6)

5. Quadratic $\lambda^2 A + \lambda B + C$ or higher degree $\Sigma \lambda^i A_i$ eigenvalue problems (Chapter 7)

6. Singular Value Decomposition (SVD) of a single general matrix A and the generalization (Chapter 8)

7. Other eigenvalue problems (sucha as in Chapter 7),

It is sometimes possible to change the classification of an eigenproblem by simple transformations. There will be cross references to other, better algorithms in cases for which simple transformations permit their use. Here are some examples:

- multiplying a skew-Hermitian matrix A (i.e., $A^H = -A$) by the constant $\sqrt{-1}$ makes it Hermitian, so its eigenproblem becomes self-adjoint. Such simple transformations are very useful in finding the best algorithm for a problem.

- Under "Hermitian eigenproblem $A - \lambda I$": If $A = BB^H$, the user will be advised to consider the SVD of B.

- Under "Generalized Hermitian eigenproblem": If $A = A^H$ and $B = B^H$, and the user knows real constants α and β such that $\alpha A + \beta B$ is positive definite, the user will be advised to consider the definite eigenproblem $A - \lambda(\alpha A + \beta B)$.

- Under "Non-Hermitian eigenproblem $A - \lambda I$": If $B = \alpha SAS^{-1}$ is Hermitian for known, simple α and S, the user will be advised to consider the eigenproblem for B.

- Under "SVD of A": If $A = BC^{-1}$, the user will be advised to consider the quotient SVD of B and C.

- Under "Quadratic $\lambda^2 A + \lambda B + C$ or higher degree $\Sigma \lambda^i A_i$ eigenvalue problems", the user will be advised to consider the linearization of problem to a matrix pencil eigenvalue problem.

- Under "Other eigenvalue problems", the user will see some other related eigenvalue problems, such as

  - Updating eigendecompositions (to be in Chapter 11)
  - Polynomial zero finding
  - Rank revealing QR decomposition (to be in Chapter 11)

2.3 Desired Spectral Information and Eigendecompositions

J. Demmel

For each eigenproblem, the user must choose which spectral information is desired. The available choices differ depending on the mathematical properties of the problem. In general, the more information the user desires, the more it will cost.

Self-adjoint Eigenproblems

We first consider the n-by-n Hermitian matrix A, which has n real eigenvalues $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$. When computing eigenvalues, the following choices are possible:

1. All the eigenvalues, to some specified accuracy.
2. Eigenvalues λ_i for some specified set of subscripts $i \in \mathcal{I}$, 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. 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. Certain number of eigenvalues closest to a given value μ.

For each of these possibilities, the user can also compute the corresponding eigenvectors (in this case left and right eigenvectors are equal). For the eigenvalues that are clustered together, the user may choose to compute the associated *invariant subspace*, i.e. the space spanned by the corresponding eigenvectors, since in this case the individual eigenvectors can be very ill-conditioned, while the invariant subspace may be less so.

The spectral information one can request from a singular value decomposition is similar, except that there are left and right singular vectors and singular subspaces, corresponding to eigenvectors and invariant subspaces. Common requests might include the numerical rank, the null space, and the range space. In order to compute these the user must supply a tolerance $\tau > 0$, which will determine which singular values are considered to be nonzero or zero. Methods can differ greatly depending on whether there is a gap between the singular values larger than τ and those smaller, and whether the user wants an approximate or precise answer.

Self-adjoint matrix pencils $A - \lambda B$, where A and B are n-by-n and Hermitian, and can be simultaneously diagonalized by a congruence, are quite similar. The case $B = I$ corresponds to the standard Hermitian eigenproblem just discussed. $A - \lambda B$ has $k \leq n$ real eigenvalues which we denote $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_k$. If B is nonsingular, $k = n$. The user may request similar subsets of eigenvalues as described above, as well as right and/or left eigenvectors. If the pencil is non-singular, the user may request *right or left deflating subspaces*, which are generalizations of invariant subspaces [**?**]. If $A - \lambda B$ is a singular pencil, the common null-space of A and B can also be computed, as well as the deflating subspaces of the regular part.

Just as the SVD of A is closely related to the eigendecomposition of $A^H A$, the QSVD corresponds to the matrix pencil $A^H A - \lambda B^H B$. The case $B = I$ corresponds to the usual SVD.

Table 2.1 spells out the possible decompositions which display all the information the user could want. We use terminology which will apply to the non-Hermitian problem too. "Schur" form refers to the simplest decomposition possible with complex unitary transformations. (Real orthogonal matrices are also possible, if the original data is real; we leave out the real case for ease of exposition.) "Jordan-Schur" form refers to the most detailed decomposition possible with unitary transformations; this form reduces the matrix or matrices to "staircase" form [248], wherein all the sizes of Jordan blocks are immediately available, and some more detailed invariant (or deflating [**?**] or reducing [**?**]) subspaces are computed. "Jordan" form, which typically

TABLE 2.1: *The Possible "Eigendecompositions" of Self-adjoint Eigenproblems*

Problem Type	"Schur"	"Jordan-Schur"	"Jordan"
$A - \lambda I$	$Q^H A Q = \Lambda$	same as Schur	same as Schur
$A - \lambda B$	$Q^H A Z = T_A$ $Q^H B Z = T_B$	$Q^H A Z = T_{S,A}$ $Q^H B Z = T_{S,B}$	$S^H A S = \Lambda_A$ $S^H B S = \Lambda_B$
SVD of A	$Q^H A Z = \Sigma$	same as Schur	same as Schur
QSVD of A, B	$Q^H A S = \Sigma_A$ $Z^H B S = \Sigma_B$	same as Schur same as Schur	same as Schur same as Schur

requires non-unitary transformations and so can be very ill-conditioned to compute, is the most detailed decomposition. Jordan form is usually called Weierstrass form for regular pencils, and Kronecker form for singular pencils. Q and Z matrices in the table are unitary, S matrices are general nonsingular, Λ is real and diagonal, Σ is real, diagonal, and nonnegative, T matrices are upper triangular, and T_S matrices are in upper triangular staircase form.

The decompositions may also be written with A and B isolated on the left-hand-side, for example $A = Q \Lambda Q^H$ instead of $Q^H A Q = \Lambda$. The reason for writing these decompositions as they are shown, is that they can be *partial decompositions*, if only $k < n$ of the eigenvalues or singular values are of interest to the user. In this case the central matrices (Λ, Λ_A, Λ_B Σ, Σ_A, Σ_B T_A, T_B, $T_{S,A}$, and T_{S_B}) will be k-by-k, and the outer matrices (Q, Z and S) will be n-by-k.

In addition to these decompositions, the user may request *condition numbers* for any of the computed quantities (eigenvalues, means of eigenvalue clusters, eigenvectors, invariant/deflating/reducing subspaces) [?, ?, ?]. Given computed values for eigenvalues, eigenvectors, and/or subspaces, the user may also request an *a posteriori* error bound based on a computed residual.

Non-self-adjoint Eigenproblems

An n-by-n non-Hermitian matrix has n eigenvalues, which can be anywhere in the complex plane. A non-self-adjoint regular pencil has from 0 to n eigenvalues, which can be anywhere in the extended complex plane. Thus some of the choices in the self-adjoint case do not apply here. Instead, the following choices of spectral information are possible:

1. All the eigenvalues.
2. All the eigenvalues in some specified region of the complex plane, such as
   - The peripheral eigenvalues (when the spectrum is seen as a set in the complex plane), such as the 10 rightmost eigenvalues.
   - Eigenvalues in a given compact region, such as the unit disk.
   - Eigenvalues in a given unbounded region, such as the left half plane, or (a region around) the imaginary axis.

As before, the desired accuracy of the eigenvalues may be specified. For each of these choices, the user can also compute the corresponding (left or right) eigenvectors, or Schur vectors. Since the eigenvalues of a non-Hermitian matrix can be very

TABLE 2.2: *The Possible "Eigendecompositions" of Non-self-adjoint Eigenproblems*

Problem Type	"Schur"	"Jordan-Schur"	"Jordan"
$A - \lambda I$	$Q^H A Q = T$	$Q^H A Q = T_S$	$S^{-1} A S = J$
$A - \lambda B$, regular	$Q^H A Z = T_A$ $Q^H B Z = T_B$	$Q^H A Z = T_{S,A}$ $Q^H B Z = T_{S,B}$	$S_L^{-1} A S_R = J_A$ $S_L^{-1} B S_R = J_B$
$A - \lambda B$, singular	$Q^H A Z = U_A$ $Q^H B Z = U_B$	$Q^H A Z = U_{S,A}$ $Q^H B Z = U_{S,B}$	$S_L^{-1} A S_R = K_A$ $S_L^{-1} B S_R = K_B$

ill-conditioned, it is sometimes hard to find all eigenvalues within a given region with certainty. For eigenvalues that are clustered together, the user may choose to estimate the mean of the cluster, or even the ϵ-*pseudospectrum*, the smallest region in the complex plane which contains all the eigenvalues of all matrices B differing from the given matrix A by at most ϵ: $\|A - B\| \leq \epsilon$. The user may also choose to compute the associated invariant (or deflating or reducing) subspaces (left or right) instead of individual eigenvectors. However, due to the potential ill-conditioning of the eigenvalues, there is no guarantee that the invariant subspace will be well-conditioned.

A singular pencil has a more complicated eigenstructure, as defined by the Kronecker Canonical Form, a generalization of the Jordan Canonical Form [95, 248]. Instead of invariant or deflating subspaces, we say a singular pencil has reducing subspaces.

Table 2.2 spells out the possibilities. In addition to the notation used in the last section, U matrices denote generalized upper triangular (singular pencils only), U_S matrices are generalized upper triangular in staircase form (singular pencils only), J is in Jordan form, and K is in Kronecker form. As before, these can be partial decompositions, when Q, Z, S_L and S_R are n-by-k instead of n-by-n.

In addition to these decompositions, the user may request condition numbers for any of the computed quantities (eigenvalues, means of eigenvalue clusters, eigenvectors, invariant/deflating/reducing subspaces). Given computed values for eigenvalues, eigenvectors, and/or subspaces, the user may also request an a posteriori error bound based on a computed residual.

2.4 Problem Representation and Available Operations

J. Demmel

First we discuss the *type* of individual matrix or operator entries. The simplest distinction is between *real* and *complex* data. The issue s are speed (real arithmetic is faster than complex), storage (real numbers requir e half the storage of complex numbers), code complexity (some algorithms simplify greatly if we can do complex arithmetic), and stability (real arithmetic can guarantee complex conjugate eigenvalues of real matrices). The distinction between *single* and *double* precision, which clearly affects space, will be discussed again as an accuracy issue.

Given the type of individual operator entries, we must distinguish the overall structure of the operator. Here is our list. Two letter abbreviations (like GE) refer to matrix types supported by LAPACK [**?**, Table 2.1].

- Dense matrices
  - General (GE in LAPACK notation)
  - Hermitian, with only upper or lower half defined (SY or HE)
  - Hermitian, packed into half the space (SP or HP)
- Matrices depending systematically on $O(n)$ data.
  - Band matrices
    - Bidiagonal matrices, stored as two arrays (BD)
    - Tridiagonal matrices, stored as two or three arrays (ST, GT)
    - Wider band matrices (GB, SB, HB)
    - Band matrices with bulges ("look ahead" matrices)
  - Arrow, Toeplitz, Hankel, Circulant, and Companion matrices
  - Diagonal + rank-1 matrices, band + low rank matrices
- Sparse matrices, or those depending less systematically on $o(n^2)$ data. It turns out that the relevant classification is by the operations that can be performed on the operators represented, not the details of the representation. This will be clearer in Section ??.
  - Triangular factorization $A = PLU$ is possible, perhaps on the "shifted matrix" $A - \mu B = PLU$
  - Matrix-vector multiplication possible $y = Ax$ (and $y = A^H x$) (this includes, for example, "dense" integral operators which have fast algorithms for performing $y = Ax$.)
  - Possible to factor AA^H or $A^H A$, perhaps shifted (for SVD)
  - Possible to factor $\begin{bmatrix} 0 & A \\ A^H & 0 \end{bmatrix}$, perhaps shifted (for SVD)
- Hierarchical operators (built out of above building blocks)
- Sequences of operators of the sort described above, where each operator is "close" to one or more previous ones. A common example is where the norm, or the rank, of the difference of consecutive matrices is small.

There are many more special structures, often arising from control theory, such as unitary orthogonal matrices represented by their Schur parameters. Until we hear of a large demand for such problems, we do not plan to include them, other than as literature references. For the same reason we also do not plan to include algorithms for quaternion matrices, or matrices with rational entries, which are more properly in the domain of symbolic computation.

All eigenvalue algorithms are built up around matrix transformations and matrix and vector operations. The choice between different methods very much depends on which of these operations one has available at a reasonable cost. We list the major ones in decreasing order of power:

- General purpose methods, which do not generally require user tuning

  - Some special matrices, such as bidiagonals or diagonal + rank-1 matrices, have special algorithms designed especially for them. Many good implementations are available [?].
  - Similarity transformations $S^{-1}AS$ can be applied to matrices A of reasonable size, stored as two dimensional arrays, or in a dense band structure. (For pencils $A - \lambda B$, we would use equivalence transformations $S_L^{-1}AS_R - \lambda S_L^{-1}BS_R$ instead). Many of these algorithms have good implementations available [?]. Some of them (such as sign function based algorithms [?]) are currently being implemented. When good "black-box" implementations exist, they will not be described in detailed templates. We will however indicate when they are to be preferred over the iterative methods, which are the main theme of this collection.

- Specialized methods, which generally require user tuning

  - Multiplication of a vector x by a shifted-and-inverted operator, $y = (A - \mu I)^{-1}x$ or $y = (A - \mu B)^{-1}Bx$ for a given vector x, lets us quickly compute the eigenvalues closest to the shift μ. This operation may be implemented with an explicit triangular factorization

    $$A - \mu B = PLU$$

    e.g. as a part of an FEM (finite element method) package, but any other implementation may also be used, such as those based on multigrid or more specialized methods. It is also possible to use an iterative solver which only uses multiplication by A (or A and B) internally, but this may not be superior to other methods discussed below.
  - If the shift μ is restricted, say to 0, then we will be restricted to efficiently finding eigenvalues closest to zero, and will need more iterations to find any but the few smallest eigenvalues.
  - If only B is factorizable, e.g. if $B = I$, so that we can only multiply vectors by AB^{-1}, we will be restricted to efficiently finding the *peripheral* eigenvalues, i.e. those eigenvalues near the "edges" of the spectrum, when the spectrum is seen as a set in the complex plane.

If, in a special purpose method, it is possible to multiple a vector by A^H in addition to A, or possibly $y = (A-\mu I)^{-H}x$ or $y = B^H(A-\mu B)^{-H}x$, depending on the situation, then a broader, more powerful class of algorithms are available.

Sometimes it is significantly cheaper to apply the operator to a set of vectors rather than just 1 vector at a time. This can happen because of memory hierarchy effects. There are algorithms to exploit this behavior.

The worst case is when the only operation we can perform is multiplication of a vector by A (and possibly B). But algorithms are available in this case too.

Not only the matrix-vector operations but also the vector algebra needs special consideration. In all the algorithms that we will consider, only two different vector operations have to be applied to vectors of length n, dot products and the addition of a multiple of one vector to another, `axpy` to use terminology from the BLAS [?]. These operations may be implemented on distributed processors in a way that is appropriate

for the application or the computer at hand. In some cases a vector is not even represented as an array of numbers, but stands for a function represented in e.g. a finite element or wavelet basis. Then user-supplied routines for dot products and axpy will replace the basic vector operations in our templates.

2.5 Overview of Available Algorithms

J. Demmel

Direct (transformation) Algorithms vs. Iterative Algorithms definition for the direct and iterative methods, ...

Classification of Iteration Algorithms The following basic algorithms for sparse eigenproblems will be included.

- Simultaneous iteration methods
- Arnoldi methods
- Lanczos methods
- Davidson methods
- Refinement methods
- Trace minimization (self-adjoint case only)

This list is not exhaustive, and we are actively looking for other algorithms. Also, some common methods may classified in several ways. For example, simultaneous iteration and block Arnoldi with an immediate restart are identical. These categories are not meant to be mutually exclusive, but to be helpful to the user. We will include some older but commonly used methods, just to be able to advise the user to use more powerful alternatives, and for experimental purposes.

Arnoldi and Lanczos methods are Krylov subspaces based techniques. These methods may converge very fast in combination with shifted-and-inverted operators, which means that $(A-\mu I)^{-1}$ has to be used in matrix vector products in each iteration step. If only approximations for $(A-\mu I)^{-1}$ are available then Davidson's method can be used as an acceleration technique for the inexact shift-and-invert operations. Approximations for $(A-\mu I)^{-1}$ can be computed from a preconditioner for $A-\mu I$ or by a few steps of a (preconditioned) iterative method [**?**].

Trace minimization is suitable for self-adjoint problems, and uses optimization techniques like conjugate gradients to find the k smallest eigenvalues.

There is unfortunately no simple way to identify the best algorithm and choice of options to the user. The more the user discovers about the problem (such as approximate eigenvalues), the better a choice can be made. In the common situation where the user is solving a sequence of similar problems, this is quite important.

There is also unfortunately no inexpensive way to provide a guarantee that all eigenvalues in a region have been found, when the problem is not self-adjoint. For Hermitian eigenvalue problems by factoring certain translations of A by the identity,

it is possible to guarantee that all eigenvalues in a region have been found. In the non-Hermitian case, this same task is accomplished by a vastly more expensive technique called a Nyquist plot (i.e. compute the winding number). However, depending on the problem, there are methods to help increase one's confidence that all eigenvalues have been found.

Cost Issues As stated above, the user would ideally like a simple formula, or perhaps a program, that would predict the running time as a function of a few simple facts about the problem to be solved, and the computer to be used to solve it. This implies that we need to build performance models for all the algorithms we provide. Realistically, for some dense algorithms we will be able to give operation counts, dependent on the size and mathematical properties of the problem to be solved, the information desired by the user, and perhaps some rough properties of the operator, like the clustering of the spectrum. For sparse problems, we can do the same for the inner loop of the iteration, counting operations like matrix-factorizations, matrix-vector multiplies, dot products, saxpys, and so on.

For particular machines, we can provide Matlab scripts to actually estimate running times in terms of a few machine parameters, like megaflop rate (for the 3 levels of BLAS), number of processors and communication costs (for parallel machines), matrix dimension, layout (on parallel machines), information desired by the user, and so on. There are two basic approaches to producing such performance models (hybrids are possible too). The first is *intrinsic*, which uses operation counts plus simpler models for the costs of each operation (BLAS operations and communication), and the second is *extrinsic*, which simply does curve fitting of benchmark runs. Intrinsic models are more difficult to produce, may be less accurate in certain extreme cases, but are more flexible and illuminating than extrinsic models.

Accuracy Issues As usual in numerical linear algebra, we will use the backward error analysis model to assess the accuracy of the results. We will first give perturbation theorems that tell how much a perturbation of the data, in this case matrix elements, will change the results, eigenvalues, singular values and vectors. We will also provide a posteriori methods to measure the backward error for computed solutions.

In the simplest cases, the perturbation theorems give a bound for the perturbation of the eigenvalues as a multiple of the perturbation of the matrix elements. The multiplier is called a *condition number*. For eigenvectors we also need information about the distance between the different eigenvalues, as well as the angles between left and right eigenvectors. In degenerate cases, the best we can get is an asymptotic series, possibly involving fractional powers of the perturbation.

We will discuss what to do when a simple condition number based bound is not practical. If we do not have good bounds for individual eigenvectors, then a better conditioned invariant (or deflating or reducing) subspace of higher dimension may be available. We can also derive a bound for the norm of the resolvent and find *pseudospectra*.

For iterative algorithms *a posteriori* methods are used to compute the backward error, often using computed residuals. It is possible to compute, or at least estimate, the residual during the computation as part of monitoring for convergence.

We will show when stronger results as e.g. small relative error bounds for small eigenvalues exist. This is of importance, specially when an ill conditioned matrix comes

from the discretization of a well conditioned continuous problem.

2.6 Which Algorithm/Template Meets My Needs?

Subtitle: Criteria for the Choice of an Algorithmic Template

The eigenvalue problem is far more varied than the also important linear algebra problem of solving systems of equations. It is useful to categorize the methods along five (mostly independent) axes to clarify the nature of the problems.

1. Mathematical properties: such as Hermitian or non-Hermitian eigenproblem.
2. Desired spectral information: such as all or partial eigenvalues and eigenvectors, a few eigenvalues in either end of the spectrum or all eigenvalues in an interval or many eigenvalues
3. Problem representation: such as dense or sparse matrix.
4. Available tools to manipulate the matrix in question: Can you store the full matrix as an array and make a similarity transformation, can you solve a linear system by a direct matrix factorization routine, or do you have a preconditioner available to solve a linear system associated matrices in question?
5. Costs (including dependence on accuracy, computer type): high or low accuracy, lots of memory, supercomputer, workstation, or PC

   Do you have storage space enough to store one, a few or many vectors?

Ideally, the user would supply a "coordinate" for each of these five axes, thus uniquely identifying a problem class for which we could identify a "best algorithm," software implementing it, a performance model to predict its execution time, and a method to assess the accuracy.

Realistically, only a few regions of this five-dimensional space are populated with interesting or useful algorithms, and we expect to have a more decision-tree like approach to guide the user's expectations.

Axel's version on the choice of algorithm:

2.6.1 Criteria for choice of algorithm

The choice of an algorithm for solving an eigenvalue problem is governed by three main factors,

1. Properties of the *matrix*, how large is the size n, how sparse is the matrix and what is the distribution of the eigenvalues?
2. Which part of the *solution* is needed. Eigenvalues and/or eigenvectors, a few eigenvalues in either end of the spectrum, all eigenvalues in an interval or many eigenvalues?
3. Which *tools* do you have available to manipulate the matrix? Can you store the full matrix as an array and make a similarity transformation, can you solve a linear system by a direct matrix factorization routine, or do you have a preconditioner available to solve a linear system? Do you have storage space enough to store one, a few or many vectors?

The reader is adviced to consider these questions carefully. Even though the theory of standard Hermitian eigenvalue problems is simple, it is in general too much to ask from an algorithm to produce many eigenvalues of a matrix of large size, of which we have no further information. On the other hand, if the matrix is small enough, so that we can store a full matrix of that size in our computer, we may call *off the shelf* software to get a complete solution. It is the intermediate cases that are of interest in this book, and these cover many of those eigenvalue problems that are important in practical applications.

A general advice is that the more powerful tools you have available to manipulate the matrix, the better solution can you get at a moderate expenditure in computer effort. If your only way of accessing the matrix is to compute $y = Ax$ for a given vector x, the best you can hope for is to get a moderate set of eigenvalues in the ends of the spectrum. On the other hand, if you have a direct linear system solver available to solve $(A - \sigma I)x = b$ for x, given b and a shift σ, there are algorithms that give many eigenvalues in an interval around σ, at the cost of the solution of a moderate number of shifted linear systems. There are other cases, like when you can solve an unshifted system $Ax = b$, and when you have a preconditioner available, which are somehow in between.

Another important factor is the availability of storage space. If your matrix is so large, compared to your memory resources, that you can store only one or two vectors, your choice is limited. If you can store a moderate set of vectors you have a choice between a couple of subspace based algorithms which may compute several eigenvalues with fewer matrix vector multiplies than needed for a single vector algorithm.

2.7 Basic Ideas of Iterative Projection Methods

Y. Saad

Since most algorithms presented in detail in this book are iteration projection methods designed for solving large sparse eigenvalue problems, in this section, we discuss a general framework of these iterative projection methods. The basic idea of a projection method is to extract an approximate eigenvector from a certain 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.

2.7.1 Introduction

Most of the standard eigenvalue algorithms exploit projection processes in order to extract approximate eigenvectors from a given subspace. 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 following:

(1) Start: select $u^{(0)}$.
(2) For $i = 0, \ldots,$ until convergence do:
(3) $\quad u := Au^{(i)}$
(4) $\quad$ Let k such that $|u_k| = \max_{j=1,\ldots,n} |u_j|$ and $\alpha_k \equiv u_k$
(5) $\quad u^{(i+1)} = u/u_k$
(6) End do

In the simplest case when the dominant eigenvalue, i.e., the eigenvalue with largest modulus, is real and simple then under mild assumptions on the initial vector, the above algorithm will converge to an eigenvector associated with this eigenvalue. 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 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:

$$\widetilde{u} = \eta_1 v_1 + \eta_2 v_2$$

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

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

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

$$A\widetilde{u} - \widetilde{\lambda}\widetilde{u} \perp v_1, \qquad A\widetilde{u} - \widetilde{\lambda}\widetilde{u} \perp v_2.$$

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 $\widetilde{u} = Vy$ and the above equations give rise to

$$(A - \widetilde{\lambda} I)Vy \perp v_1, \qquad (A - \widetilde{\lambda} I)Vy \perp v_2$$

or equivalently,

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

This yields the 2×2 generalized eigenvalue problem:

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

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, v_2.

The above idea can now be generalized from 2 dimensions to m dimensions. Thus, a projection method consists of approximating the exact eigenvector u, by a vector $\widetilde{u}$ 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 $\widetilde{u}$ 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.

2.7.2 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 $\mathbb{C}$ such that

$$Au \ = \ \lambda u. \tag{2.4}$$

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

$$A\widetilde{u} - \widetilde{\lambda}\widetilde{u} \perp \mathcal{K}\ , \tag{2.5}$$

or, equivalently,

$$v^*(A\widetilde{u} - \widetilde{\lambda}\widetilde{u}) \ = \ 0\ , \ \ \forall\ v \in \mathcal{K}. \tag{2.6}$$

To translate this into a matrix problem, assume that an orthonormal basis $\{v_1, v_2, \ldots, v_m\}$ of $\mathcal{K}$ is available. Denote by V the matrix with column vectors $v_1, v_2, \ldots, v_m$, i.e., $V = [v_1, v_2, \ldots, v_m]$, and let

$$\widetilde{u} \ = \ Vy, \tag{2.7}$$

Then, equation (2.7) becomes

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

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

$$B_m y \;=\; \widetilde{\lambda} y \tag{2.8}$$

with

$$B_m \;=\; V^* A V. \tag{2.9}$$

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

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

RAYLEIGH-RITZ PROCEDURE:

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

The values $\widetilde{\lambda}_i$ computed from this procedure are referred to as *Ritz values* and the vectors $\widetilde{u}_i$ 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 of LAPACK. Another important note is that in step 4 one can replace eigenvectors by Schur vectors to get approximate Schur vectors $\widetilde{u}_i$ 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. See section 2.x for the discussion on Schur vectors.

2.7.3 Oblique Projection Methods

In an oblique projection method we are given two subspaces $\mathcal{L}$ and $\mathcal{K}$ and seek an approximation $\widetilde{u} \in \mathcal{K}$ and an element $\widetilde{\lambda}$ of $\mathbb{C}$ that satisfy the Petrov-Galerkin condition,

$$v^*(A - \widetilde{\lambda} I)\widetilde{u} = 0 \quad \forall\, v \in \mathcal{L}\ . \tag{2.10}$$

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 by again translating in matrix form the approximate eigenvector $\widetilde{u}$ in some basis and expressing the Petrov-Galerkin condition (2.10). 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 $\widetilde{u} = V y$ as before, the above Petrov-Galerkin condition yields the same approximate problem as (2.8) except that the matrix B_m is now defined by

$$B_m \;=\; W^* A V.$$

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,*

$$\det(W^*V) \neq 0 \quad . \tag{2.11}$$

As can be seen this condition does not depend on the bases selected and is equivalent to requiring that no vector of $\mathcal{K}$ is 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 allow 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.

2.7.4 Ritz and Harmonic Ritz Values

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 u to be extracted from the subspace $\mathcal{K}$ can be expressed in the form

$$u = Vy$$

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

$$(AV)^*(A - \widetilde{\lambda}I)Vy = 0, \quad \text{or} \quad V^*A^*AVy = \widetilde{\lambda}\ V^*A^*Vy \tag{2.12}$$

This gives a generalized eigenvalue problem of size m for which one of 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, (2.12) becomes

$$V^*A^*Vy = \frac{1}{\widetilde{\lambda}}y \tag{2.13}$$

This resembles the equations obtained for the orthogonal projection methods except that V is not orthogonal but A^HA-orthogonal. To make the above system equivalent to an orthogonal projection method (Galerkin) we can re-introduce the orthonormal basis $W = AV$ and rewrite (2.13) as follows,

$$W^H\ A^{-1}W\ y\ =\ \widetilde{\lambda}^{-1}\ y \tag{2.14}$$

Since the system 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 suspace of approximants in this case is $A\mathcal{K}$. For this reason the approximate eigenvalues are referred to as Harmonic Ritz values. This technique was considered by Morgan [?]. Note that though the method involves the inverse of A, it does not require it explicitly. In particular, one should not expect this method to do as well as shift-and-invert [computing the eigenvalues of A^{-1} via some projection method]. In essence the action of A^{-1} is cancelled out by the fact that the

subspace of approximants is in the range of A. However, the method seems to perform slightly better than a standard approach based on the same subspace $\mathcal{K}$.[1]

On the computational side Harmonic Ritz Values can be obtained in one of two ways: (1) by solving the genarlized eigenvalue problem (2.12), where V is any basis of $\mathcal{K}$, or (2) by obtaining a basis which is $A^H A$-orthonormal and solving the standard problem (2.13). In the methods to be seen later, there are relations which can be exploited to simplify this process.

Let us rewrite the condition (2.10) as $[A - \widetilde{\lambda} I]\widetilde{u} \perp A\mathcal{K}$ and define $\widetilde{z} = A^{-1}\widetilde{u}$. Then this condition becomes

$$[\widetilde{\lambda}^{-1} I - A^{-1}]\widetilde{z} \perp \mathcal{L} \qquad \widetilde{z} \in \mathcal{L}$$

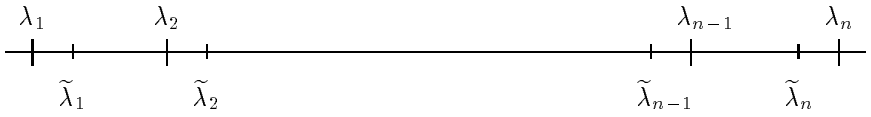
Therefore, the method can be viewed as an orthogonal projection technique for the matrix A^{-1}. It is important to realize however, that since there is no real inversion of the matrix A, this should not be seen as a way of performing a shift-and-invert strategy. This is because the matrix A and its inverse cancel each other out in the formulation of the problem since the the space of approximants is of the form $A\mathcal{K}$.

As it turns out these observations will have interesting consequences in the Hermitian case. Consider the special case of Hermitian matrices and assume that the eigenvalues are labeled increasingly:

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

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

Recall that Ritz values approximate eigenvalues of a Hermitian matrix A 'inside out', in the sense that the rightmost eigenvalues are approximation from below and the leftmost ones are approximated from above as is illustrated in the following diagram.



This principle can be appplied 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 $1/\widetilde{\lambda}_i$ obtained from the process will approximate corresponding eigenvalues $1/\lambda_i$ of A in an inside-out fashion. For positive eigenvalues we have,

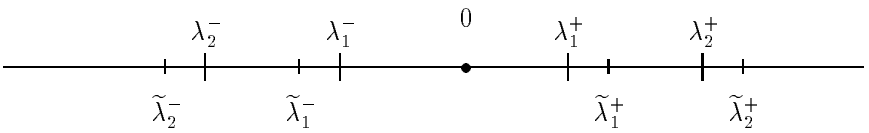
$$\widetilde{\lambda}_i^{-1} \leq \lambda_i^{-1}; \quad i = 1, 2, \ldots \qquad \longrightarrow \qquad \widetilde{\lambda}_i \geq \lambda_i; \quad i = 1, 2, \ldots$$

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

[1] **Comments by Henk: The Harmonic Ritz values does not perform better: they converge more systematically to interior eigenvalues and therefore the corresponding Harmonic Ritz vectors are more appropriate for restarting purposes. This has been pointed out by Morgan, and for non-Hermitian matrices by Sleijpen and van der Vorst (SIMAX 1996).**

to contain a given eigenvalue. [2] 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 must be cautious when applying this principle for negative eigenvalues that the order is reversed. Therefore, we label positive and negative eigenvalues separately. The above inequalities must be reversed for the negative eigenvalues, labeled from the (algebraically) smallest to the largest negative eigenvalues ($\lambda_1^- \leq \lambda_2^- \leq ...$). The result is summarized in the following diagram.



Assume for simplicity that A is SPD and define:

$$\mathcal{K}^{(i)} = \{x \in \mathcal{K} \quad | \quad Ax \perp A\widetilde{u}_1, A\widetilde{u}_2, \ldots A\widetilde{u}_{i-1}\}$$

with $\mathcal{K}^{(0)} \equiv K$. Then the Courant characterization becomes,

$$\widetilde{\lambda}_i^{-1} = \max_{x \in \mathcal{K}^{(i)}} \frac{(Ax, x)}{(Ax, Ax)}$$

Extra comments:

- **Harmonic Ritz values (by that name) have been formally introduced, with proper reference to Morgan and others, in Paige, Parlett and van der Vorst(PPvdV). There also the monotonic behavior of the Harmonic Ritz values is proved as well as their relation to Lehmann intervals. This should be properly mentioned in this section.**
- **section 2.7 ends slightly abrupt. The final relation and other useful relations, such as how to compute the Harmonic Ritz values from the Ritz representation, has been pointed out in PPvdV**

2.8 Shift-and-invert Spectral Transformation

R. Lehoucq and D. Sorensen

The general (linear) eigenvalue problem that may be solved for several eigenvalues and corresponding eigenvectors is

$$Ax = \lambda Bx \tag{2.15}$$

[2] **comments by Henk: here a reference to the PPvdV paper would be appropriate for their relations wrt Lehmann intervals.**

where A and B are real or complex $n \times n$ matrices. This section quickly reviews the fundamentals associated with spectral transformations. The reader is also referred to [169, 167, 160, 212, 255] for further information and references.

Iterative eigensolvers rapidly provide approximations to well-separated extremal eigenvalues. Quite often, the sought-after eigenvalues are generally not well separated. Instead, the sought-after eigenvalues lie in the interior of the convex hull of eigenvalues. In these situations, the iterative eigensolver slowly converges, if at all, to these eigenvalues. Therefore, a spectral transformation of (2.15) should be chosen so that the sought-after eigenvalues of (A, B) are transformed to well-separated extremal eigenvalues. Additionally, the eigenvalues of (A, B) should be easily recoverable from the eigenvalues of the transformation, and the transformation should be easily applied to a vector.

We remark that the notion of well separated (or non-clustered) 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 $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$ while for non-Hermitian matrices, $\text{Real}(\lambda_1) \leq \text{Real}(\lambda_2) \leq \ldots \leq \text{Real}(\lambda_n)$.

If (x, λ) is an eigen-pair for (A, B) and $\sigma \neq \lambda$ then

$$(A - \sigma B)^{-1} B x = \nu x \quad \text{where} \quad \nu = \frac{1}{\lambda - \sigma}. \tag{2.16}$$

This shift-invert spectral transformation is effective for finding eigenvalues near σ because the nearby eigenvalues ν_j of $C \equiv (A - \sigma B)^{-1} B$ that are largest in magnitude correspond to the nearby eigenvalues λ_j of the original problem that are nearest to the shift σ in absolute value. 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

$$\lambda_j = \sigma + 1/\nu_j,$$

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. Another possible spectral transformations is the generalized Cayley transformation; see [169] for further details.

In general, C will be non-Hermitian even if A and B are both Hermitian. However, this is easily remedied. The assumption that B is Hermitian positive definite implies that the bi-linear form

$$\langle x, y \rangle \equiv x^* B y$$

is a B-orthogonal inner product. If B is positive semi-definite and singular, then a semi-inner product results. It is easy to show that if A is Hermitian then so is C (with respect to the inner product defined by B). Therefore, a Hermitian eigenvalue problem results when the computed basis vectors are unitary in this (semi) inner product.

Implementing this B orthogonal inner product requires the user to also provide a matrix-vector product $w \leftarrow Bv$ on request along with each application of C. One should use B-inner products with shift-invert transformations if possible. The resulting numerical method is substantially more robust when $\|B\|$ is small (relative to $\|A\|$) and B is singular. The reason is that perturbations induced by B prevents corruption of the computed eigenvectors by roundoff-error associated with the presence of *infinite*

eigenvalues. Infinite eigenvalues are associated with a singular B. See the paper [170] for further details.

Shift-invert spectral transformations are also extremely effective on standard problems ($B = I$) and should be used whenever possible. This is particularly true when interior eigenvalues are sought, when the desired eigenvalues are signifigantly smaller in magnitude than the largest in magnitude eigenvalues, or when the desired eigenvalues are clustered.

The potential drawback to a spectral transformation is that a means to solve linear systems involving $A - \sigma B$ must be provided, either with a matrix factorization or with an iterative method. The user should use a sparse direct method to factor the appropriate matrix whenever possible. 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 slightly more stringent tolerance is needed for the iterative linear system solves (relative to the desired accuracy of the eigenvalue calculation). See [230, 150, 156] for details.

The main drawback with using the shift-invert spectral transformation is when an iterative method is used for solving linear systems with coefficient matrix $A-\sigma B$. This is because $A-\sigma B$ is typically indefinite in the Hermitian case and has 0 in the interior of the convex hull of the spectrum in the non-Hermitian case. These are typically the most difficult situations for iterative methods.

CHAPTER 3

Hermitian Eigenvalue Problem

3.1 Introduction

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

$$Ax = \lambda x\,, \tag{3.1}$$

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.

Let us first briefly recall some basic theory about the Hermitian eigenvalue problem (3.1). It is known that the problem (3.1) has n real eigenvalues λ_i, which we may order increasingly so that $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$. Note that several eigenvalues may coincide. In many applications the matrix A is positive definite, $\lambda_1 > 0$ (or positive semidefinite, $\lambda_1 \geq 0$). We choose to treat Hermitian A with a general distribution of eigenvalues in this chapter. However, there are some cases where one may use knowledge about positive definiteness to advantage.

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

$$A = X\Lambda X^*, \tag{3.2}$$

where $\Lambda = \text{diag}\,(\lambda_i)$, $X = [x_1, x_2, \ldots, x_n]$ 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 a couple of 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 produces eigenvector approximations that are orthogonal to each other, even if the matrix has multiple eigenvalues.

There is a very satisfactory perturbation theory for eigenvalues of Hermitian matrices. If a perturbation E is added to the matrix A, then each of the eigenvalues are

perturbed at most as far as the spectral norm $\|E\|_2$,

$$|\lambda_i(A+E) - \lambda_i(A)| \leq \|E\|_2 \,. \tag{3.3}$$

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. In many cases, one is interested in eigenvalues that are small compared to the norm $\|A\|$, and for such eigenvalues the inequality (3.3) 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, the reader is referred to section 9.1 for some further details.

There is no result as simple as (3.3) 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 [239]. If the eigenvalue λ_i is simple and separated from the other eigenvalues by the distance δ, and if we have a vector z of unit length $\|z\|_2 = 1$, such that the residual $r = Az - z\mu$ for some real number μ, then

$$\sin\angle(z, x_i) \leq \frac{\|r\|_2}{\delta} \,. \tag{3.4}$$

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

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

The first alternative in this minimum is equivalent to the previous bound (3.3), since μ is an eigenvalue of the perturbed matrix $A+E$ with $E = -rz^H$ is a rank 1 perturbation of norm $\|E\|_2 = \|r\|_2$ (it is not necessary for E to be Hermitian in the bound (3.3)).

3.1.1 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 *transformation methods* where similarity transformations are applied until the eigenvalues can be easily read out. We give a short review of some current transformation methods in section 3.2.

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 section 3.3, is the basic iterative method. It takes a starting vector and lets the matrix operate on it until we get vectors that are parallel 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 here.

Simultaneous iteration method, also described in section 3.3, 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, section 3.4, 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 then the Lanczos algorithm is economical in storage space. We will describe some variants that are useful in different cases in section 3.4.

Implicitly restarted Lanczos method, section 3.5, 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 specially useful if one needs eigenvectors.

Band Lanczos method, section 3.6, 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 [104]. In multiple input multiple output control problems, one is actually interested in reduced order models based on a given set of starting vectors, this is the main reason to develop the variant described here.

Jacobi-Davidson method, section 3.7, is a development of an algorithm that is popular in Quantum Chemistry. It updates a sequence of subspaces, but now by operating with a preconditioned matrix. If you use preconditioned iterations to solve related linear systems, this is the way to get eigenvalues!

3.1.2 Summary of choices

Let us now turn to Table 3.1, 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.

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 just let the matrix multiply a vector in each step. It is the simplest variant to apply, you can store the matrix in any compact way you like. On the other hand most algorithms need

many matrix vector multiplications to converge and you are restricted to seek eigenvalues at the ends of the spectrum.

SI is shift 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. Needs less space than shift 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 3 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 is the computation of a few leading singular values of a data matrix.

CE stands for one or several eigenvalues in the end of the spectrum possibly clustered, a typical example is computing the lowest eigenmodes of a vibrating mechanical system, modeled by the FEM (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 the table 3.1, we give an estimate of the storage needed.

#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 says if we need extra matrix storage. "LU" means a sparse LU factorization, "ILU" an incomplete factorization. It is supposed to be more compact than LU.

3.2 Transformation Methods

In this section, we briefly discuss methods for the computation of eigenvalues of symmetric matrices that can be stored in the computer as full matrices. These methods are usually called direct methods and are built up around similarity transformations. These are the methods implemented in LAPACK [2] and in the `eig` command in MATLAB.

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 variants of these methods have two phases:

TABLE 3.1: **Summary of algorithms for Hermitian eigenvalue problem**

	Appl	Orth	IE	CE	M	# vec	Fact
Power	Dir		Yes	Very slow	No	2	-
	SI		-	Yes	Yes	2	LU
Sim power	Dir	FO	Yes	Slow	No	Moderate	-
	SI	FO	-	Yes	Yes	Moderate	LU
Lanczos	Dir	local	Yes	No	No	2	-
	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	Dir	FO	Yes	Yes	No	Many	-
Lanczos	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

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 costs $\frac{4}{3}n^3$ flops, 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. 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 algorithms finds all the eigenvalues, and optionally all the eigenvectors. Implemented efficiently, it is currently the fastest direct method to find all the eigenvalues of a tridiagonal matrix, take $O(n^2)$ flops. Since reducing a dense matrix to tridiagonal form costs $\frac{4}{3}n^3$ flops, $O(n^2)$ is negligible for large enough n. But for finding all the eigenvectors as well, QR iteration takes a little over $6n^3$ flops on average and is only the fastest algorithm for small matrices, up to about $n = 25$. This the algorithm underlying the Matlab command `eig`[1] and the LAPACK routines `xSYEV` and `xHEEV` (for real and complex dense matrices) and `xSTEV` (for tridiagonal matrices).

Divide-and-conquer method This is currently the fastest method to find all the eigenvalues and eigenvectors of symmetric tridiagonal matrices larger than $n = 25$. 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. For a large tridiagonal matrix this is repeated recursively until the parts are smaller than $n = 25$, when the QR algorithm is called. It is implemented in LAPACK as `xSTEVD`.

[1] Matlab checks to see whether the argument of `eig` is symmetric or not and uses the symmetric QR algorithm when appropriate.

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 $O(nk)$ flops, where k is the number of eigenvalues desired. Thus the bisection method could be much faster than the QR method when $k \ll n$.

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 $O(nk)$ flops. This is much less than either QR or divide-and-conquer, even when all eigenvalues and eigenvectors are desired ($k = n$). On the 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 $O(nk^2)$ flops 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`.

There is a lively research on methods to get orthogonal eigenvectors for tridiagonal matrices without the need for Gram Schmidt orthogonalization, see works by Parlett and coworkers [193, 194, 55].

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 diagonal 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 ($s = 3 - 5$ usually) which is more than the $4/3n^3$ needed for tridiagonal reduction.

There is one important advantage of the Jacobi algorithm. Properly implemented it can deliver eigenvalue approximations with a small error in the relative sense, this is in contrast to algorithms based on tridiagonalization which only guarantee that the error is bounded relative to the norm of the matrix (3.3). See works by Demmel, Veselic and coworkers [54].

3.3 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.

3.3.1 Power Method

The simplest eigenvalue problem is to compute just the dominating eigenvalue along with its eigenvector. The *power method* presented in Algorithm 3.1 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 3.1: **The Power Method**

(1) $v := z/||z||_2$ *for the starting vector* z
(2) **for** $k = 2, 3, ...$
(3) $\quad y := A\,v$
(4) $\quad \theta := v^*\,y.$
(5) $\quad$ *if* $||y - \theta\,v||_2 \leq \epsilon$, *stop*
(6) $\quad v := y/||y||_2.$

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 eigenvectorx_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 converence. For detailed discussions on the power method, see Demmel [48, Ch. 4], Golub and Van Loan [101], and Parlett [192].

3.3.2 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 $\lambda_{\max}$. This method is called *inverse iteration*, or the *inverse power method*.

ALGORITHM 3.2: **The Inverse Power Method**

(1) $v := z/||z||_2$ *for the starting vector* z
(2) **for** $k = 2, 3, ...$
(3) $\quad y := (A - \sigma)^{-1}\;v,$
(4) $\quad v := y/||y||_2,$
(5) $\quad w := A\,v,$
(6) $\quad \theta := v^*\,w$
(7) $\quad$ *if* $||w - \theta\,v||_2 \leq \epsilon$, *stop*

Note that there is no need to compute the invese $(A - \sigma)^{-1}$ explicitly at the step (3). We only have to solve system like $(A - \sigma)\,y = z$ for y efficiently, which formally gives $y := (A - \sigma)^{-1}\;v$.

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. 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.

3.3.3 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 $\rho(x) \equiv x^* A x / x^* x$.

ALGORITHM 3.3: **The RQI**

(1) $v := z/||z||_2$ *and* $\theta := \rho(v)$ *for the starting vector* z
(2) **for** $k = 2, 3, \ldots$
(3) $\quad y := (A - \theta I)^{-1} v,$
(4) $\quad v := y/\|y\|_2,$
(5) $\quad w := A v,$
(6) $\quad \theta := v^* w$
(7) $\quad$ *if* $\|w - \theta v\|_2 \leq \epsilon$, *stop*

In general RQI will need fewer iterations to find an eigenvalue than inverse iteration with a constant shift, it has ultimately cubical convergence, while inverse iteration converges linearly. However, it is not obvious how to choose the starting vector z 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 $\rho(z)$ and to an eigenvector which is not closest to the starting vector z. Furthermore, there is the tiny but nasty possibility that it may not converge to an eigenvalue/eigenvector pair. RQI is more expensive than inverse iteration, requiring a factorization of $A - \theta I$ at every iteration. Hence RQI is practical only if such factorizations can be cheaply obtained at every iteration. See Parlett [192] for more details.

3.3.4 Orthogonal 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 3.4: **Simple Subspace Iteration**

(1) *QR-factorize* $V\ R := Z$ *for the starting matrix* Z
(2) **for** $k = 2, 3, ...$
(3) $Y := A\,V$
(4) $H := V^*\,Y$.
(5) *if* $\|Y - V\,H\|_2 \leq \epsilon$, *stop*
(6) *QR-factorize* $V\ R := Y$.

This algorithm is a straightforward generalization of the power method. Step (6) 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.

3.3.5 Practical Subspace Iteration

Several modifications are needed to make the simple subspace iteration an efficient and practically applicable code. First, it is natural to orthonormalize (6) as infrequently as possible, i.e. to perform several iterations (3) before performing an orthogonalization. Second, we may choose to operate on a subspace whose dimension p is larger than $n_{\rm ev}$, the number of eigenvalues wanted, and use a Rayleigh-Ritz process to get eigenvalue approximations. Third some eigenvalues will converge faster than others, and then 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 in the end of this section.

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

Locking Because of the different rates of convergence of each of the approximate eigenvalues computed by the subspace iteration, it is a common practice to extract them one at a time and perform a form 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 $v_2, \ldots, v_m$. 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 $n_{\rm ev}$ dominant eigenvalues.

ALGORITHM 3.5: **Subspace Iteration with Projection and Deflation**

(1) *QR-factorize* $V\,R := Z$ *for the starting* Z
Set $j = 0$
(2) **while** $j \leq n_{ev}$ **do**
(3) $\widehat{Y} := \left[V_j,\, A^{iter} V_{p-j}\right]$.
(4) *Orthonormalize* $\widehat{Y} = V\,R$, *(*j *first columns will be invariant)*
(5) $H := V_{p-j}^*\, A\, V_{p-j}$
(6) *compute the eigendecomposition* $H = S\,\Theta\,S^*$
(7) *Test the diagonal entries in* Θ *for eigenvalue convergence.*
(8) $V := [V_j\; V_{p-j} S]$.
(9) *Set* $j := j + i_{conv}$, i_{conv} *is the number of newly converged eigenvalues.*
(10) *Choose a new iteration parameter* $iter$.

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.

(2) and (10) The iteration parameter $iter$ should be chosen to minimize orthonormalization cost while maintaining a reasonable amount of numerical accuracy. The amplification factor $(\lambda_1/\lambda_p)^{iter}$, where the eigenvalues λ_i are ordered in decreasing absolute values, gives the loss of accuracy. Rutishauser [**?**], plays it safe and allows an amplification factor of only 10, losing one decimal, while Stewart and Jennings [240] lets it run to $\epsilon^{-1/2}$, half the machine accuracy, but not more than $iter = 10$.

Acceleration If eigenvalues near a *shift* σ are desired, and a factorization $A - \sigma\, I = LU$ can be easily obtained, 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^{iter} 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 gives a translation and scaling of the part of the spectrum one wants to suppress. Ideally one should take $\sigma = (\lambda_{p+1} + \lambda_n)/2$ and $\rho = (\lambda_{p+1} - \lambda_n)/2$ for some reasonable estimates of those eigenvalues. (Now 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 seen 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 [48], Golub and Van Loan [101], and Saad [212]. For further discussion on subspace iteration, the reader is recommended to refer to Chatelin [26], Lehoucq and Scott [157], Stewart [238] and Wilkinson [255].

3.3.6 Software Availability

There are several pieces of software available for subspace iteration. EA12 is a routine by Duff and Scott on symmetric subspace iteration [60]. 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 in [256] where most of the refinements discussed here have been included.

There are also two subspace iteration based software packages for the non-symmetric eigenvalue problem, SRRIT by Z. Bai and G. W. Stewart [16] and LOPSI by W. J. Stewart and A. Jennings [241]. These packages can be used to solve the symmetric eigenvalue problem, however, with some loss in efficiency and accuracy. See Section ?? for more details.

3.4 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 to all those previously tried.

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

$$Ax = \lambda x \ , \tag{3.5}$$

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,

$$K^j(A, v) = \text{span}\{v, Av, A^2v, \ldots, A^{j-1}v\} \ , \tag{3.6}$$

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

$$y = Ax \tag{3.7}$$

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

$$T_j = \begin{bmatrix} \alpha_1 & \beta_1 & & \\ \beta_1 & \alpha_2 & \ddots & \\ & \ddots & \ddots & \beta_{j-1} \\ & & \beta_{j-1} & \alpha_j \end{bmatrix} \ , \tag{3.8}$$

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

$$AV_j = V_jT_j + re_j^* \quad \text{with} \quad V_j^*r = 0. \tag{3.9}$$

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

$$T_j s_i^{(j)} = s_i^{(j)} \theta_i^{(j)} \ , \tag{3.10}$$

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

$$x_i^{(j)} = V_j s_i^{(j)} \ , \tag{3.11}$$

will be a good approximation to an eigenpair of A, if the residual has small norm, see for example [239].

Let us compute the residual for this Ritz pair,

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

We see that its norm satisfies

$$\| r_i^{(j)} \|_2 = | \beta_j s_{i,j}^{(j)} | = \beta_{j,i} \ , \tag{3.12}$$

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 this can be done without actually performing the matrix vector multiplication (3.11). We can save this time consuming operation until the step j when the estimate (3.12) indicates convergence.

3.4.1 Algorithm

We get the following algorithm:

ALGORITHM 3.6: **Hermitian single vector Lanczos**

(1) **Start** *with* $r = v$, *starting vector. Compute* $\beta_0 = \| r \|_2$
(2) **for** $j = 1, 2, \ldots,$ **until** *Convergence,*
(3) $v_j = r / \beta_{j-1}$
(4) $r = A v_j$
(5) $r = r - v_{j-1} \beta_{j-1}$
(6) $\alpha_j = v_j^* r$
(7) $r = r - v_j \alpha_j$
(8) *Reorthogonalize if necessary*
(9) $\beta_j = \| r \|_2$
(10) *Compute approximate eigenvalues* $T_j = S \Theta^{(j)} S^*$
(11) *Test bounds for Convergence*
(12) **end** j
(13) *Compute approximate eigenvectors* $X = V_j \, S$

Let us comment on some of steps of Algorithm 3.6:

(1) If a good guess for the wanted eigenvector is available, use it. For instance, if it is known for a discretized PDE that the wanted eigenvector is smooth with respect to the grid, then 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.

(4) This is most often the CPU-dominating work. Any routine that performs a matrix vector multiplication can be used. In the shifted and inverted case, do solvers with the factored matrix. There is, as of now, little solid experience of using iterative equation solvers in the inverted case.

(8) Were the arithmetic exact, this had not been needed, but in finite precision orthogonality is lost as soon as one eigenvalue converges. The choices are:

1. *Full:* Simple to implement see subsection 3.4.4 below. Iterations will need successively more work as j increases. This is the preferred choice in shifted and inverted 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 subsection 3.4.4.
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 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.

(10) 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 (3.8).

If fast convergence is expected, as for shifted and inverted problems, j will be much smaller than n, then this computation is very fast and standard software from e. g. LAPACK [2] is recommended.

If Lanczos 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 [193, 194, 55] and section 3.2.

(11) 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 (3.8) give good approximations to all the eigenvalues of A sought.

The estimate $\beta_{j,i}$ (3.12) for the residual may be too optimistic if the basis V_j is not fully orthogonal. Then the Ritz vector $x_i^{(j)}$ (3.11), may have its norm smaller than 1, and we have to replace the estimate (3.12) by

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

(13) The eigenvectors of the original matrix A are computed only when the test in step (11) has indicated that they have converged. Then the basis V_j is used in a matrix vector multiplication to get the eigenvector (3.11),

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

for each i that is flagged as converged.

The great beauty of this type of algorithms is that the matrix A is accessed only in a matrix vector operation in step (4) in Algorithm 3.6. Any type of sparsity and any storage scheme can be taken advantage of.

Except for the matrix vector multiplication, only scalar products and additions of a multiple of a vector to another are needed. We recommend to use BLAS1 routines.

3.4.2 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 [192].

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 then it is good that those are among the first to converge. 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.

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 introduce also 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 tot he converged eigenvector(s), where all the converged eigendirections have been projected away. This corresponds to orthogonalizing the vector r in step (3) of Algorithm 3.6 to all converged eigenvectors. This procedure is repeated as long as new vectors converge, see e. g. [117]. There is a second more radical way to deal with possibly multiple eigenvalues, *Block Lanczos*. It starts with several, say p starting directions, forming a block V_1, 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 section 3.6.

In block Lanczos method, the convergence is governed by the separation of the eigenvalue sought λ_i from other eigenvalues, 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.

Block Lanczos is anyhow of advantage for efficiency reasons, whenever computing $Y = AX$ for an $n \times p$ matrix X is much cheaper than that 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 on secondary storage.

3.4.3 Spectral transformation

If the extreme eigenvalues are not well separated and when we want interior eigenvalues, it is of great advantage to replace A by a shifted and inverted operator

$$C = (A - \sigma I)^{-1} , \tag{3.13}$$

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 shifted and inverted operator C has the eigenvalues

$$\theta_i = \frac{1}{\lambda_i - \sigma} , \tag{3.14}$$

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 also get very well separated from the rest. See [75]!

If we run a shifted and inverted operator, we factor

$$LU = (A - \sigma I) , \tag{3.15}$$

once in the beginning, using some convenient sparse Gaussian elimination scheme. If there are eigenvalues λ at both sides of the shift σ, this matrix is indefinite, and we cannot use simple Cholesky decomposition, but we have to make a symmetric indefinite factorization, as for example in MA 47 of Duff and Reid [59]. As a by product, we get a count of the number of eigenvalues in each interval by recording the inertia of $(A - \sigma I)$ for two values σ at the ends of the interval. This is used to make sure that no multiple eigenvalues are missed.

During the actual iteration, we use the factors L and U, computing,

$$r = U^{-1}(L^{-1}v_j) , \tag{3.16}$$

in step (2) of Algorithm 3.6.

3.4.4 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 , the rest of the orthogonality follows from the symmetry of A and the recursion (3.9). As soon as one eigenvalue converges, i. e. gets a small residual (3.12), all the basis vectors v_j get perturbations in the direction of the eigenspace of the converged eigenvalue. This leads to that a duplicate copy of that eigenvalue will soon show up in T. Paige [185] was the one who discovered this, the reader is referred to the monograph [192] for a detailed discussion.

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 it is motivated to look for a more economic scheme. It can be proved that most of the good properties of the Lanczos algorithm are still there, as long as the basis is *semiorthogonal*, i. e. orthogonal to half the machine precision,

$$W_j = V_j^* V_j = I_j + E\,, \quad \|E\| < \sqrt{\epsilon}\,. \tag{3.17}$$

The reader is referred to the investigation by Simon [223] 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 (3.17), then T is a fully accurate projection of A on the subspace spanned by the computed basis V_j,

$$T_j = N_j^* A N_j + G\,, \quad \|G\| = \mathcal{O}(\epsilon\|A\|)\,.$$

Here N_j is some orthogonal basis of the subspace spanned by V_j.

One good way to make 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 do a reorthogonalization when this recurrence indicates that a treshold of $\sqrt{\epsilon}$ is exceeded. The recurrence is between successive elements $\omega_{j,k}$ of the product matrix W_j (3.17) and is derived from the basic recursion (3.9) in the following way.

First we note that the computed vector v_{j+1} satisfies (3.9)

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

where f_j is a vector of rounding errors. To get elements $\omega_{j,k}$ of the product matrix W_j (3.17) we premultiply this with v_k^* and get

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

We do the same thing with the indices j and k exchanged

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

subtract and get

$$\beta_j \omega_{j+1,k} = \beta_k \omega_{j,k+1} + (\alpha_k - \alpha_j)\omega_{j,k} + \beta_{k-1}\omega_{j,k-1} - \beta_{j-1}\omega_{j-1,k} - v_j^* f_k + v_k^* f_j \tag{3.18}$$

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

We follow this recursion while the computation goes along and 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 $\omega_{k-1,k} = \omega_{k,k-1} = \epsilon_k$ at the rounding error level, due to symmetry and the explicit orthogonalization. This gives starting values for the recursion (3.18). We estimate the absolute value of the sum of the two last terms in (3.18) by $2\epsilon\|A\|$, 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,

$$\widetilde{\omega} = \beta_k \omega_{j,k+1} + (\alpha_k - \alpha_j)\omega_{j,k} + \beta_{k-1}\omega_{j,k-1} - \beta_{j-1}\omega_{j-1,k}$$

$$\omega_{j+1,k} = (\widetilde{\omega} + \mathrm{sign}(\widetilde{\omega})2\epsilon\|A\|)/\beta_j$$

As soon as one element $\omega_{j+1,k}$ exceeds the $\sqrt{\epsilon}$ sized treshold, we orthogonalize the two current vectors v_j and v_{j+1} to all the previous vectors v_k, $k = 1, \ldots, j$, 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 we have a three term recurrence in Lanczos, and as a consequence in the ω recursion (3.18). It is not necessary to store more than the two last rows of the matrix W of ω estimates.

This is just one of the techniques described in [223], there is an alternative when one does explicit orthogonalization only to those vectors v_k for which $\omega_{j+1,k}$ exceeds the treshold. This is what Simon calls *partial* reorthogonalization, but the variant described here, *periodic* reorthogonalization, which was first described by Grcar [103], is simpler to implement and can make use of BLAS 2 operations. It is also the variant used in the program [?]. **Axel, what to do with this software?**

3.4.5 Software Availability

There are a few software packages available on `netlib`. One, LANCZOS, is published by Cullum and Willoughby [35, 36]. It uses Lanczos with no reorthogonalization, but contains a mechanism to detect and discard spurious eigenvalue approximations. David Scott developed LASO, a block Lanczos code with selective orthogonalization, as described in the pioneering paper [196]. A more recent package, LANZ, is written by Jones and Patrick [124] and includes shift invert spectral transformation (3.13) and uses selective (partial) reorthogonalization, as described in section 3.4.4 here.

Finally there is a code, LANSO, a FORTRAN package that implements selective orthogonalization as developed by the group around B. N. Parlett in Berkeley. It is available from the EE department at UCB. (**Axel, does this mean this will not be in public-domain?**) It follows the description given in this section very closely. It makes periodic reorthogonalization as recommended above in subsection 3.4.4. It calls a standard tridiagonal QL algorithm to compute eigenvalue approximations (3.10) and estimate residuals (3.12), at appropriate intervals.

3.4.6 Numerical Examples

Let us show how the Lanczos algorithm behaves on two simple but typical examples. We use LANSO software.

The first example is the computation of some eigenmodes of an L shaped membrane. The standard 5 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 5 nonzeros elements at most in each row. Its eigenvalues will be in the interval $0 < \lambda_i < 8$, symmetrically distributed around $\lambda = 4$, and more densily distributed towards the ends of the spectrum.

The second test example is taken from an information retrieval application, and involves a a term document matrix, where each column stands for one document and each row for one term. The element $x_{i,j} = 1$ if term i occurs in document j, zero otherwise. The space of a set of leading singular values ca be used to find connections between some of the documents. The specific matrix MEDLINE is rectangular of size 7014×1033 and moderately sparse with 53287 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 910755 or 85% of all the elements nonzero. Implement $y = Ax$ by first computing $z = Xx$ and then $y = X^T z$.

Results for L membrane: Direct application of a Lanczos algorithm gives converge first towards the extremal eigenvalues, and in this case the convergence will be very similar in either end of the spectrum.

We have plotted the estimated residuals (3.12) for the 6 largest eigenvalues as a function of the number j of Lanczos steps in figure 3.1. The curves show the residual estimates (3.12) for each step. We computed those 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 (3.12), only in those steps we marked with dashdotted vertical lines in the plot. The selective orthogonalization triggered reorthogonalization at the steps we marked with dashed lines, altogether only 3 times during all these 300 steps. This is typical for situations with a slow convergence, the orthogonality stays good just until the first Ritz value converges.

Note that quite many, about 150 steps 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. Look also at the Ritz values $\theta_i^{(j)}$ (3.10) plotted versus j in figure 3.2! The largest Ritz value grows with j until around step $j = 60$ when it stabilizes. Rather often a Ritz value starts to approach one eigenvalue, only to later on move to another, not yet found, eigenvalue. This happens with the third Ritz value at steps 60 to 80, and more pronounced with the sixth between steps 150 and 210. This phenomenon shows up in a less evident fashion in figure 3.1, where the third and sixth curve from the bottom have plateaus at the steps when a Ritz value shifts allegiance. A user of direct Lanczos 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 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 3.3. The first eigenvalue reaches full accuracy already at step $j = 14$, and after $j = 50$ steps all the first 6 eigenvalues are converged. After $j = 300$ steps we had 100 eigenvalues.

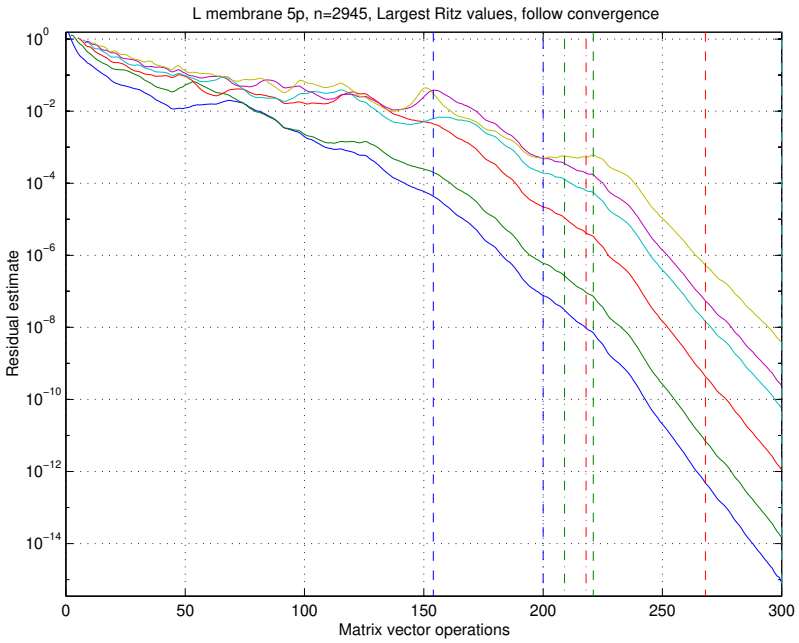


FIGURE 3.1: **Residual estimates, L-membrane matrix,** $n = 2945$.

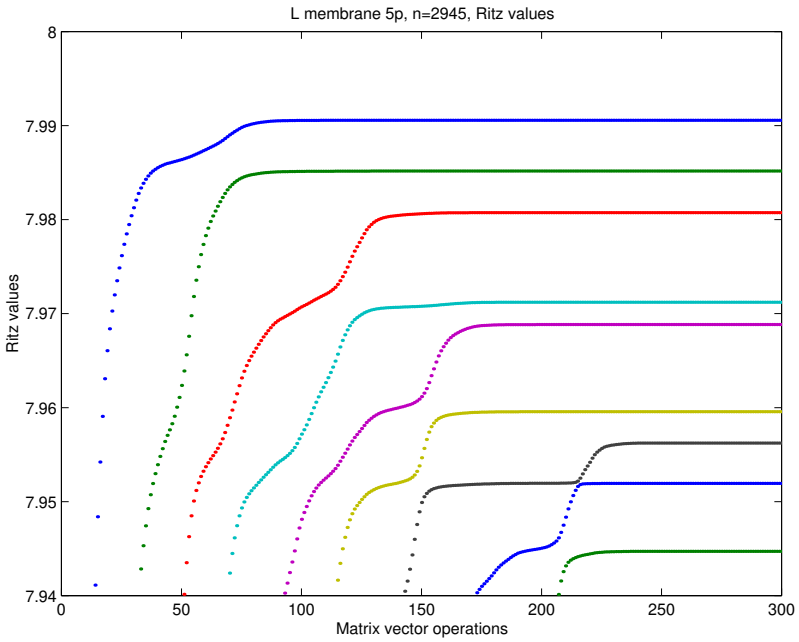


FIGURE 3.2: **Ritz values, L-membrane matrix,** $n = 2945$.

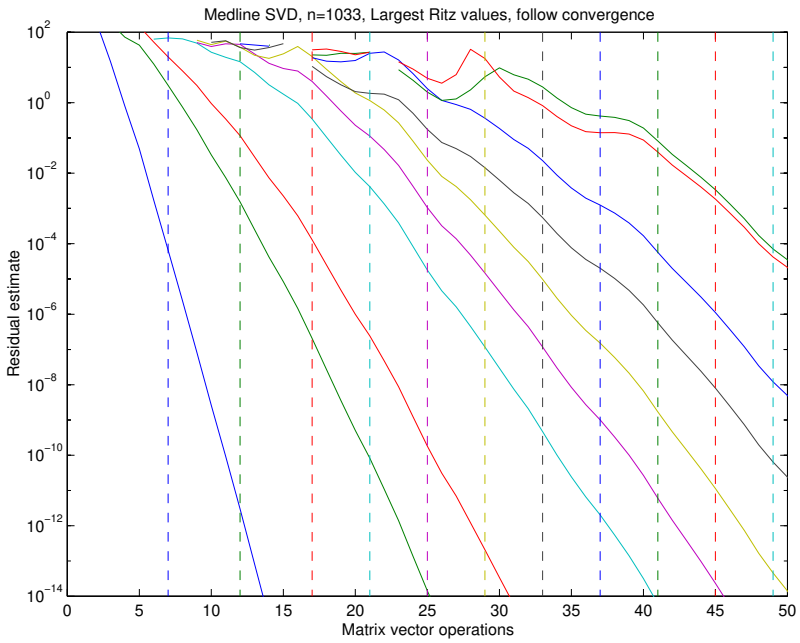


FIGURE 3.3: **Residual estimates, Medline SVD example**

There is another interesting difference between this well separated problem and the L membrane with its more clustered eigenvalues. Now reorthogonalization is triggered much more often, we mark the first reorthogonalization with a dashed vertical line at step $j = 7$, then one at $j = 12$, and later on every 4 steps. Since each reorthogonalization involves 2 vectors, selective reorthogonalization demands about half as much work as a full reorthogonalization. In such cases the user is adviced to use full reorthogonalization, it gives full orthogonality of the basis vectors at a moderate extra cost in arithmetic work.

Results for L membrane, shifted and inverted: To illustrate the effect of applying Lanczos to a shifted and inverted operator (3.13), we follow the convergence for the L membrane matrix with a shift applied at the origin $\sigma = 0$. The picture in figure 3.4 is now very similar to the Medline SVD example, and we get full accuracy in the 6 smallest eigenvalues after $j = 36$ steps. In this run we used full reorthogonalization as advised. Even if the number of steps j is reduced by more than a factor of 10, we have to take into consideration that the LU factorization (3.15) takes some time and that the factors L and U are denser than the original A, so applying their inverses in each step (3.16) needs more work than the application of the original A (3.7) in direct Lanczos.

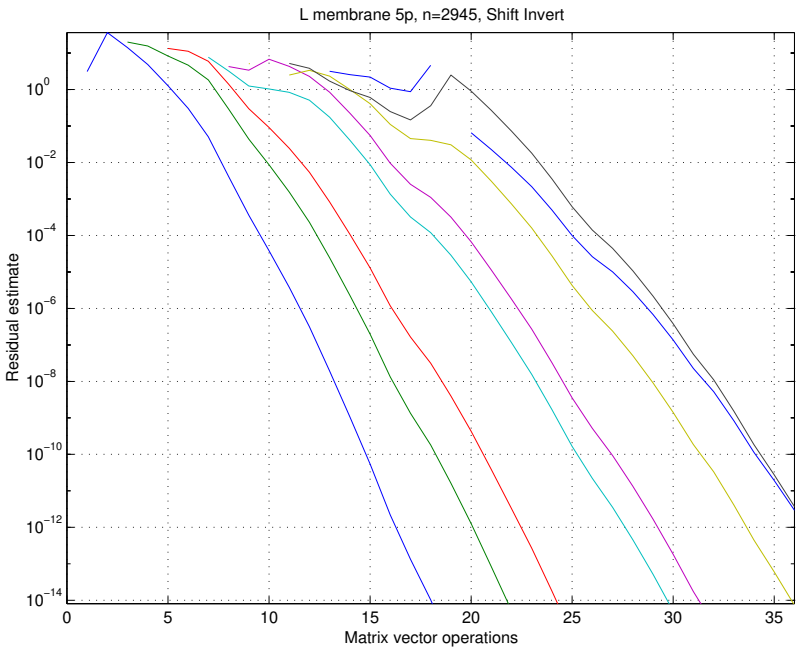


FIGURE 3.4: *Residual estimates, Shifted and inverted L-membrane matrix, $n = 2945$.*

3.5 Implicitly Restarted Lanczos Method

R. Lehoucq and D. Sorensen

The Lanczos process for a Hermitian matrix $A = A^*$ has been derived previously in section 3.4. Here, we discuss it in the framework of the implicit restart. Our starting point is a k-step Lanczos factorization (3.9)

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

where $V_k \in \mathbf{C}^{n\times k}$ has orthonormal columns, $V_k^* r_k = 0$ and $T_k \in \mathbf{R}^{k\times k}$ is real, symmetric and tridiagonal with non-negative subdiagonal elements. We shall call this a *k-step Lanczos factorization* of A. The columns of V_k are referred to as the *Lanczos vectors* . For implicit restart it is important that the columns of V_k are made orthogonal to full accuracy.

3.5.1 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 $\mathcal{O}(n \cdot k^2)$ flops.

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 then compute 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 subpace.

The need for restart was recognized early on by Karush [131] soon after the appearance of the original algorithm of Lanczos [152]. Subsequently, there were developments by Paige [185], Cullum and Donath [34], and Golub and Underwood [100]. All of these schemes are *explicit* in the sense that a new starting vector is produced by some process and then 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 $2nk + \mathcal{O}(k^2)$ storage.**Do you assume that $m = 2k$ here? This choice is not recommended later here, why not say sth like $2nm + m^2$ and explain?** 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 $m = k + p$

$$AV_m = V_m T_m + r_m e_m^*, \tag{3.19}$$

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

$$AV_m^+ = V_m^+ T_m^+ + r_m e_m^* Q, \tag{3.20}$$

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 (3.20) remain in a Lanczos relation. Equating the first k columns on both sides of (3.20) provides an updated $k-$step Lanczos factorization

$$AV_k^+ = V_k^+ T_k^+ + r_k^+ e_k^*, \tag{3.21}$$

with an updated residual of the form $r_k^+ = V_m^+ e_{k+1} \beta_k + r_m \sigma$. 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 implicitly restarted Lanczos method is given in Algorithm 3.7[2]

[2] In the algorithm, where does convergence test?

ALGORITHM 3.7: **Implicitly Restarted Lanczos Method (IRLM)**

(1) *Start with* $v_1 = v/\|v\|$;
(2) *Compute an* $m = k + p$ *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* $\mu_1, \mu_2, \ldots, \mu_p$
$Q = I_m$;
(5) *for* $j = 1, 2, ..., p$,
Factor $[Q_j, R_j] = qr(T_m - \mu_j I)$;
$T_m \leftarrow Q_j^* T_m Q_j$;
$Q \leftarrow QQ_j$;
end
(6) $r_k \leftarrow v_{k+1}\beta_k + r_m\sigma_k$, *with* $\beta_k = T_m(k+1, k)$, $\sigma_k = Q(m, k)$;
(7) $V_k \leftarrow V_m Q(:, 1:k)$; $T_k \leftarrow T_m(1:k, 1:k)$;
(8) *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*
$AV_m = V_m T_m + r_m e_m^*$.
(9) *end*

We will now describe some implementation details, referring to the respective phases in Algorithm 3.7.

(1) and (2) Choose the initial starting vector v, normalize and compute an initial $m = k + p$ step Lanczos factorization[3]. 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 of interest from the previous problem. In the absence of any other considerations, a random vector is a reasonable choice.

(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. A successful strategy has been to sort the eigenvalues of T_m into a "wanted set" $\theta_1, \theta_2, \ldots, \theta_k$ and an "unwanted set" $\mu_1, \mu_2, \ldots, \mu_p$ 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 algebraically smallest eigenvalues.

The k algebraically largest eigenvalues.

The k eigenvalues with largest magnitude.

The k eigenvalues with smallest magnitude.

For Hermitian matrices there are more precise ways, as e. g. all ev above or below a certain number or all in an interval. Somewhere warn against trying to get interior eigenvalues withou shift invert!

[3] **What to use? Algorithm 3.8 below?**

(5) Apply p steps of the shifted QR iteration to T_m using the $\mu_1, \mu_2, \ldots, \mu_p$ 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 $\theta_1, \theta_2, \ldots, \theta_k$ as it's eigenvalues.

(6) When exact shifts are used, then the properties mentioned in step (5) 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, 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 have been produced so far.

(8) This step requires a slight modification of the Lanczos process discussed previously. It simply begins with step k of the usual scheme and assumes that all k Lanczos vectors have been retained in a matrix V_k. The detail is given in Algorithm 3.8 below.

3.5.2 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 orthogonal eigenvectors of A. **This is essentially the same as remark (4) above. Write about wanted sets in one place either here on in remark (4)!**

Other interesting strategies include the roots of Chebyshev polynomials [209], Harmonic Ritz values [175, 180, 187, 227], the roots of Leja polynomials [9], the roots of least squares polynomials [210] and refined shifts [121]. 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.

$$v_1 \leftarrow \psi(A)v_1 \quad \text{with} \quad \psi(\lambda) = \prod_{j=1}^{p}(\lambda - \mu_j). \tag{3.22}$$

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 [234].

3.5.3 Lanczos in GEMV form

Starting with the k-step Lanczos factorization

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

or, for $k = 0$ with a starting vector r_0. A template for applying p additional steps of the Lanczos process to extend this to an m-step Lanczos factorization

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

is given in Algorithm 3.8.

Dan: I have changed the numbering here, should give same result with less text. Couldn't we remove the index from the residual vector r inside the algorithm, this would give further simplification and be more similar to a typical implementation.

ALGORITHM 3.8: **m-Step Lanczos Factorization**

(1) *for* $j = k+1 : m$,
  $\beta_{j-1} = \|r_{j-1}\|$;
(2) $v_j \leftarrow r_{j-1}/\beta_{j-1}$;
  $V_j \leftarrow [V_{j-1}, v_j]$;
  $w \leftarrow Av_j - v_{j-1}\beta_{j-1}$;
(3) $\alpha_j \leftarrow v_j^* w$;
  $r_j \leftarrow w - v_j \alpha_j$;
(4) *if* $(\|r_j\| < \eta\sqrt{\alpha_j^2 + \beta_{j-1}^2}\,)$,
    $s \leftarrow V_j^* r_j$;
    $r_j \leftarrow r_j - V_j s$;
    $\alpha_j \leftarrow \alpha_j + s(j)$;
    $\beta j \leftarrow \beta j + s(j-1)$;
  end_if
  $T_j \leftarrow \begin{bmatrix} T_{j-1} & e_{j-1}\beta_{j-1} \\ \beta_{j-1} e_{j-1}^* & \alpha_j \end{bmatrix}$;
end

We will now describe some implementation details, referring to the respective phases in Algorithm 3.8.

(1) If started from scratch ($k = 0$) take $r_0 = v$ 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.

(2) Normalize r_{j-1} to get the new basis vector v_j. The norm calculation $\beta_{j-1} = \|r_{j-1}\|$ need not be recomputed here. It has already been computed at Step (4) and can be saved for reuse here.

(3) This is the Lanczos three term recurrence step to orthogonalize w with respect to the two most recent columns of V_j. It is organized in modified Gram-Schmidt form.

(4) The sequence of statements in this *if* clause assure that the new direction r_j 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 $w = Av_j$ nearly in the space that already has been constructed?" The ratio $\|r_j\|/\sqrt{\alpha_j^2 + \beta_{j-1}^2} = \tan\theta$, where θ is the angle that the vector w makes with the range space of V_j, that is the angle θ between w and its projection $V_j h$. A larger value of ρ is a more stringent orthogonality requirement. With $\rho = 0$ the algorithm reverts to the standard Lanczos process with no re-orthogonalization. One step of this correction may not be sufficient. This should be implemented as an iteration with the test for subsequent corrections being $\|r_j\| < \rho\|s\|$. If a suitable r_j has not been generated after a fixed number of attempts (say 5), 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_j to continue the factorization. Further discussion of this scheme as well as some alternatives are given below. We suggest setting $\rho = 1$. **replaced η by ρ since η stays for elements of eigenvector later on.**

If it is necessary to re-orthogonalize at Step 4, the dense matrix-vector products $V_j^* w$ and $V_j h$ may be expressed with the Level 2 BLAS operation _GEMV. This provides a significant performance advantage on virtually every platform from workstation to super-computer. Moreover, considerable effort has been made within the ScaLapack project [20] and also by several high-performance computer vendors to optimize these kernels for a variety of parallel machines. **We should give refernce to discussion on Gram Schmidt in one place, there are 3 similar discussions just in Chap 3 now!**

The mechanism used to orthogonalize the new information Av_k against the existing basis V_k is the classical three term Lanczos recurrence. It has been organized in Modified Gram Schmidt form (MGS) as discussed in Algorithm 3.6. It is well known that this scheme will fail to produce orthogonal vectors precisely when a Ritz value (approximate eigenvalue) converges to an eigenvalue of A. This method will certainly fail to produce orthogonal vectors in the restart situation we are about to discuss. To remedy this situation, at step (4) we have included an iterative refinement technique proposed by Daniel, Gragg, Kaufman and Stewart (DGKS) in [38]. This scheme provides an excellent way to construct a vector r_j that is numerically orthogonal to V_j. The simple test specified at step (4) is used to avoid this DGKS correction if it is not needed. For the iterative refinement, see also Algorithm 3.16.

This mechanism maintains orthogonality to full working precision at 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 this numerical orthogonality is not maintained.

3.5.4 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

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

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 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. The basic iteration is listed in Figure 3.7 .

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 tridiagonal submatrix $T_m(1:k, 1:k)$ are mathematically equivalent to the matrices that would appear in the full Implicitly Shifted QR iteration using the same shifts μ_j. In this sense, the Implicitly Restarted Lanczos method may be viewed as a truncation of the Implicitly Shifted QR iteration. bf I agree this far! 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. **Disagree! I think this one tries to split the matrix at column k, it is QR so the zeros are floating from top to bottom. This is different from TRQ where a shifted and inverted matrix is used and zeros float down from the top, I think.**

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 [234] for information on the convergence of IRLM and [8, 237] for other possible shift strategies for Hermitian A. The reader is referred to [158, 172] for studies comparing implicit restart with other schemes.

3.5.5 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 Lanczos 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. **Do you choose k ? I thought it was the number of**

wanted eigenvalues or do you choose k at a higher value, like the subspace iteration people do? Explain! 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 γn. For a sparse matrix, one could think of γ as twice the average number of non-zero entries per row of A. For a dense matrix $\gamma = 2n$, and for an FFT matrix $\gamma \approx log(n)$. For a fixed value of k and p, the cost (in flops) for one iteration of IRLM is

1. Matrix vector products $w \leftarrow Av$: γpn.
2. Basic Lanczos Steps: $9pn$.
3. 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).
4. Implicit Restart: $2n(k^2+kp) + \mathcal{O}((k+p)^3)$.

Total Cost: $\gamma pn + (6k+9)pn + 4p^2n + 2k^2n + \mathcal{O}((k+p)^3)$.

Note that the term $4p^2n$ is the result of a worst case orthogonalization correction scenario. This term is more likely to be linear in p in practice. **I guess one reorth will be needed nearly always, this happened for Arnoldi at least. I think that this one is worse when the start vector gets richer in wanted eigenvectors, causing convergence at an early step j.**

If the matrix vector product Av is cheap (i.e. γ is quite small) then 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 $w \leftarrow \psi(A)$ would, of course, be applied as a succession of matrix vector products involving A. Another alternative would be to use another method such as the one designed by Cullum and Willoughby [32]. **They use L without reorth. I think it will give mighty many steps?**

3.5.6 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 **agree?** 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 round off 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 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 thought to be a member of the wanted set of eigenvalues, then 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 then we wish to decouple and remove the eigenpair (x, θ) from the current Krylov subspaced spanned by the Lanczos vectors. This process is called purging.

Techniques for this deflation were first developed in [154, 159]. The scheme we present here is based an improvement developed in [235]. 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 shift invert spectral transformation 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 [154], the forward instability of the QR bulge chase process discovered by Parlett and Le [195] will prevent implicit restart to be used for purging converged unwanted Ritz values.

3.5.7 An 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, the algorithm shown in Figure 5.27 computes an orthogonal matrix Q such that $Qe_1 = y$ (hence $y = Q^* e_1$). This orthogonal matrix has a very special form. The orthogonal matrix Q constructed as prescribed in Figure 5.27 may be written as

$$Q = R + ye_1^*, \quad \text{with} \quad Re_1 = 0, \; R^* y = 0, \tag{3.23}$$

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

$$Q = L + yg^*, \quad \text{with} \quad Le_1 = 0, \; L^* y = e_1 - g, \tag{3.24}$$

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

Now, consider the matrix Q^*TQ. The substitutions $Q^* = (L+yg^*)^*$, $Q = (R+yfe_1^*)$ from the equations (5.16) and (5.15) 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 5.27, Q will have componentwise relative errors on the order of machine precision ϵ_M with no element growth. Moreover, extension to complex arithmetic is completely straightforward (unlike Givens or Householder transformations).

ALGORITHM 3.9: **Orthogonal Deflating Transformation**

function $[Q] = \mathbf{orthQ}(y)$;
Input: *y of dimension k with* $\|y\| = 1$ *and* $\eta = e_k^* y$.
Output: *Q such that* $Qe_1 = y$, $e_k^* Q = (\eta, \tau e_{k-1}^*)$, *with* $Q^*Q = I$.

$Q = 0$; $Q(:,1) = y$;
$\sigma = y(1)^2$; $\tau_o = |y(1)|$;
for $j = 2 : n$,
  $\sigma \leftarrow \sigma + y(j)^2$;
  $\tau = \sqrt{\sigma}$;
  if $(\ \tau_o \neq 0)$,
    $\gamma = (y(j)/\tau)/\tau_o$;
    $Q(1 : j-1, j) = -y(1 : j-1)\gamma$;
    $Q(j,j) = \tau_o/\tau$;
  else
    $Q(j-1, j) = 1$;
  end_if
  $\tau_o = \tau$;
end_for;

3.5.8 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 θ: 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 ϵ_M and 1.

If θ is "wanted" then it is desirable to lock θ. 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 θ. This may be accomplished by constructing a $k \times k$ orthogonal matrix $Q = Q(y)$ using Algorithm 5.27

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

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

The end result of these transformations is

$$\begin{aligned} Av_1 &= v_1\theta + f\eta, \ \text{ where } \ v_1^* f = 0 \\ AV_2 &= V_2 T_2 + f\tau e_{k-1}^*, \end{aligned}$$

where $[v_1, V_2] = VQ$.

Is this f the same as your previous f_k, that I denoted by r_k? Make clear with a reference!

This means that subsequent implicit restart takes place as if

$$AV_2 = V_2 T_2 + f\tau e_{k-1}^*$$

with all the subsequent orthogonal transformations associated with implicit restart applied to T_2 and never disturbing the relation $Av_1 = v_1\theta + f\eta$. In subsequent Lanczos steps, v_1 participates in the orthogonalization so that the selective orthogonalization recommended by Parlett and Scott [196, 192] is accomplished automatically.

Purging θ: If θ is "unwanted" then we may wish to remove θ 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 [159].

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

$$AV_2 = V_2 T_2 + f\tau e_{k-1}^*,$$

where $V_2 = VQ(:, 2:k)$. 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 η 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 θ. Finally, this procedure is valid in complex arithmetic with minor notational modifications.

3.5.9 Stability of Q^*TQ

The deflating orthogonal transformation construction shown in Algorithm 5.27 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^*TQ numerically preserves the eigenvalues of T. However, there is a serious question about how well these transformations perform numerically in preserving tridiagonal form 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. that 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 η_1 is the first component of y. Therefore,

$$\|g\| = \frac{1}{|\eta_1|}, \tag{3.25}$$

so there may be numerical difficulty when the first component of y is small. To be specific, $y^*T = \theta y^*$ and thus $y^*TR = 0$ in exact arithmetic. However, in finite precision, the computed $fl(y^*T) = \theta y^* + z^*$. The error z will be on the order of ϵ_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 $\mathcal{O}(1)$ if $\eta_1 = \mathcal{O}(\epsilon_M)$. 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

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

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

$$y^*Tq_j = 0, \quad \text{for} \quad j > 1,$$

numerically. As shown in [235], this modification is sufficient to establish $q_i^*Tq_j = 0$ numerically, relative to $\|T\|$ for $i > j + 1$.

The basic idea is as follows: If, at the j-th step, the computed quantity $fl(y^*Tq_j)$ is not sufficiently small, then it is adjusted to be small enough by scaling the vector y_j with a number ϕ and the component η_{j+1} with a number ψ. With this re-scaling just prior to the computation of q_{j+1}, we then have $y_j \leftarrow y_j\phi$ and $\widehat{y}_j \leftarrow \widehat{y}_j\psi$ where $y^* = [y_j^*, \widehat{y}_j^*]$. Certainly, $\|y\|$ should not be altered with this scaling and this is therefore required. This gives the following system of equations to determine ϕ and ψ: 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 ρ_j is on the order of $\sqrt{\epsilon_M}$, then the scaling may be absorbed into ρ_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 $q_i, 2 \le i < j$, need to be altered. After step j, the vector y_j is simply rescaled in subsequent steps, and the formulas defining $q_i, 2 \le i \le j$, are invariant with respect to scaling of y_j. For complete detail, one should consult [235].

Algorithm 5.28 implements this scheme to compute an acceptable Q. In practice, this transformation may be computed and applied in place to obtain $T \leftarrow Q^*TQ$ and $V \leftarrow VQ$ 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 $2 - n$ have been determined.

ALGORITHM 3.10: **Stable Orthogonal Deflating Transformation**

function $[Q] = \mathbf{orthQ}(y, T)$;
Input: *y of dimension m with* $\|y\| = 1$ *and* $\eta = e_m^* y$.
T symmetric tridiagonal of order m with $Ty = y\theta$.
Output: *Q such that* $Qe_1 = y$, $e_m^* Q = (\eta, \tau e_{m-1}^*)$, *with* $Q^*Q = I$.
y may be modified. Q^*TQ *is tridiagonal.*

$Q = 0$;
(1) *while* $(y(i) = 0)$, $y(i) = \epsilon_M / n$; $i = i + 1$; *end_while*
$\sigma = y(1)^2$; $\tau_o = |y(1)|$;

for $j = 2 : n$,
$\sigma = \sigma + y(j)^2$; $\tau = \sqrt{\sigma}$;
$\gamma = -(y(j)/\tau)/\tau_o$;
$\rho = \tau_o/\tau$;
$Q(1 : j-1, j) = y(1 : j-1)\gamma$; $Q(j, j) = \rho$;
(2) *if* $(j < n \;\; \& \;\; \tau < .05)$,
$\tau_p = \sqrt{\sigma + y(j+1)^2}$;
$\alpha = y(1 : j)^* T(1 : j, 1 : j) Q(1 : j, j)$;
$\delta = y(j+1) T(j, j+1) \rho$;
$\sigma_o = -1$; *if(* $sign(\alpha) \cdot sign(\delta) < 0$*)*, $\sigma_o = 1$; *end_if;*
(3) *if*$(|\delta + \alpha| > \epsilon_M \tau_p)$,
if $(\rho < \sqrt{\epsilon_M}/100)$,
$Q(j, j) = Q(j, j)\sigma_o(\epsilon_M \tau_p + |\alpha|)/|\delta|$;
else
if$(|\alpha| > |\delta|)$,
$\phi = \sigma_o(\epsilon_M \tau_p + |\delta|)/|\alpha|$;
$y(1 : j) = y(1 : j)\phi$;
$\sigma = \sigma\phi^2$; $\tau = \tau|\phi|$;
else
$\psi = \sigma_o(\epsilon_M \tau_p + |\alpha|)/|\delta|$;
$y(j+1 : n) = y(j+1 : n)\psi$;
end_if
end_if
end_if
$\tau_0 = \tau$;
end_for
$Q(:, 1) = y/\|y\|$;

There are several implementation issues.

(1) 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 when a diagonal entry of T is small but not zero. **This happens if θ is weakly represented in the starting vector. The product of the β_j s may be small without one of them being small!** There is a

cleaner implementation possible that does not modify zero entries. This is the simplest (and crudest) correction.

(2) 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 5.27.

(3) 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 ρ instead of y is designed so that $\mathrm{fl}(1 + (\rho\psi)^2) = \mathrm{fl}(1 + \rho^2) = 1$.

3.5.10 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 [235], 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 3.11: **Deflation for IRLM**

(1) Lock a single Ritz value θ 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 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 ($p \leftarrow p - 1$). This allows the polynomial filter to have a larger relative magnitude on the Ritz value that is most likely to converge next (See Section 3.5.4). 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 then the IRLM iteration takes place in columns $k_o + 1 : m$ 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 Section 3.5.4 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.

3.5.11 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 purpose. 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.

Avoid details, give reference to publication instead! ARPACK is a collection of Fortran77 subroutines designed to solve large scale eigenvalue problems. ARPACK stands for ARnoldi PACKage. 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. **Ref?**

The ARPACK software is based upon the Implicitly Restarted Arnoldi Method (IRAM) that is presented here in Section 5.5. When the matrix A is symmetric (Hermitian) this reduces to the the Implicitly Restarted Lanczos Method (IRLM) just described in this section.

It provides various drivers in template form for shift-invert spectral transformations. There are special drivers for banded matrices. It also provides a driver for computing a partial singular value decomposition (SVD) of a general rectangular matrix. For complete information, one should consult the ARPACK Users' Guide [160].

3.6 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 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 Section 5.9.4 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., [85]. 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 to compute 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 matrix-vector products Av.

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

$$Ax = \lambda x, \tag{3.26}$$

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

$$b_1, b_2, \ldots, b_p. \tag{3.27}$$

The matrix A and the starting vectors (3.27) induce the block-Krylov sequence

$$b_1, b_2, \ldots, b_p, Ab_1, Ab_2, \ldots, Ab_p, \ldots, A^{k-1}b_1, A^{k-1}b_2, \ldots, A^{k-1}b_p, \ldots . \tag{3.28}$$

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

$$v_1, v_2, \ldots, v_j, \tag{3.29}$$

that build a basis for the subspace spanned by the first j linearly independent vectors of the block-Krylov sequence (3.28).

3.6.1 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., $A^j b \in K^j(A,b)$. In this case, the Lanczos vectors build a basis of the A-invariant subspace $K^j(A,b)$ and all eigenvalues of the Lanczos tridiagonal matrix are also eigenvalues of A. The subspace $K^j(A,b)$ 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 (3.28) 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 (3.28). This process of detecting and deleting linearly dependent vectors is called *exact deflation*. For example, suppose the starting vectors (3.27) are such that the vector Ab_1 is a linear combination of $b_1, b_2, \ldots, b_p$. Then Ab_1 is certainly linearly dependent on the Krylov vectors to the left of Ab_1 in (3.28); in fact, each vector $A^i b_1$ with $i \geq 1$ is linearly dependent on vectors to the left of Ab_1 in (3.28). In this case, all vectors $A^i b_1$, $i \geq 1$, need to be deleted from (3.28), resulting in the deflated block-Krylov sequence

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

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 matrix 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 the sequel, we will refer to the process of detecting and deleting linearly dependent and almost linearly dependent vectors as *deflation*.

3.6.2 Basic Properties

After the first j iterations, the band Lanczos algorithm has generated the first j Lanczos vectors (3.29). These vectors are constructed to be orthonormal:

$$V_j^* V_j = I_j, \quad \text{where} \quad V_j = \left[\begin{array}{cccc} v_1 & v_2 & \cdots & v_j \end{array} \right]. \tag{3.30}$$

Here, I_j denotes the $j \times j$ identity matrix. In addition to (3.29), the algorithm has constructed the vectors

$$\widehat{v}_{j+1}, \widehat{v}_{j+2}, \ldots, \widehat{v}_{j+p_c}, \tag{3.31}$$

which are the candidates for the next p_c Lanczos vectors, $v_{j+1}, v_{j+2}, \ldots, v_{j+p_c}$. 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 (3.31) are constructed so that they satisfy the orthogonality relations

$$V_j^* \widehat{v}_k = 0 \quad \text{for all} \quad k = j+1, j+2, \ldots, j+p_c. \tag{3.32}$$

The band Lanczos algorithm has a very simple built-in deflation procedure based on the vectors (3.31). In fact, an exact deflation at iteration $j+1$ is equivalent to $\widehat{v}_{j+1} = 0$.

Therefore, in the algorithm, one checks if $\|\widehat{v}_{j+1}\|_2$ is smaller than some suitable deflation tolerance. If yes, the vector $\widehat{v}_{j+1}$ is deflated and p_c is reduced by one. Otherwise, $\widehat{v}_{j+1}$ is normalized to become the next Lanczos vector v_{j+1}.

The recurrences that are used in the algorithm to generate the vectors (3.29) and (3.31) can be summarized compactly as follows:

$$AV_j = V_j T_j + \left[\begin{array}{ccccccc} 0 & \cdots & 0 & \widehat{v}_{j+1} & \widehat{v}_{j+2} & \cdots & \widehat{v}_{j+p_c} \end{array}\right] + \widehat{V}_j^{(\mathrm{dl})}. \tag{3.33}$$

Here, T_j is a $j \times j$ matrix whose entries are chosen so that the orthogonality conditions (3.30) and (3.32) are satisfied. The matrix $\widehat{V}_j^{(\mathrm{dl})}$ in (3.33) 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

$$T_j = T_j^{(\mathrm{b})} + T_j^{(\mathrm{d})}, \tag{3.34}$$

where $T_j^{(\mathrm{b})}$ is a banded matrix and $T_j^{(\mathrm{d})}$ contains the horizontal "spikes" outside the band of T_j due to deflation. In particular, if no inexact deflation has occurred, then $T_j^{(\mathrm{d})}$ is the zero matrix. Finally, we note that the banded part $T_j^{(\mathrm{b})}$ 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 (3.28). In this case, the matrix $T_{15} = T_{15}^{(\mathrm{b})} + T_{15}^{(\mathrm{d})}$ has the following sparsity structure:

$$T_{15} = \left[\begin{array}{cccccccccccccc}
* & * & * & * & * & * & & & & & & & & \\
* & * & * & * & * & * & * & & & & & & & \\
* & * & * & * & * & * & * & \mathtt{d} & \mathtt{d} & \mathtt{d} & \mathtt{d} & \mathtt{d} & \mathtt{d} & \mathtt{d} & \mathtt{d} \\
* & * & * & * & * & * & * & * & & & & & & & \\
* & * & * & * & * & * & * & * & * & & & & & & \\
* & * & * & * & * & * & * & * & * & * & & & & & \\
 & * & * & * & * & * & * & * & * & * & \mathtt{d} & \mathtt{d} & \mathtt{d} & \mathtt{d} & \mathtt{d} \\
 & & & * & * & * & * & * & * & * & * & & & & \\
 & & & & * & * & * & * & * & * & * & * & & & \\
 & & & & & * & * & * & * & * & * & * & \mathtt{d} & \mathtt{d} & \mathtt{d} \\
 & & & & & & & * & * & * & * & * & * & & \\
 & & & & & & & & * & * & * & * & * & * & \\
 & & & & & & & & & & * & * & * & * & * \\
 & & & & & & & & & & & * & * & * & * \\
 & & & & & & & & & & & & * & * & *
\end{array}\right]. \tag{3.35}$$

Here, the $*$'s denote potentially nonzero entries within the banded part, $T_{15}^{(\mathrm{b})}$, and the $\mathtt{d}$'s denote potentially nonzero entries of $T_{15}^{(\mathrm{d})}$ 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 (3.26) are obtained by computing eigensolutions of $T_j^{(\mathrm{pr})}$,

$$T_j^{(\mathrm{pr})} z_i^{(j)} = \theta_i^{(j)} z_i^{(j)}, \quad i = 1, 2, \ldots, j.$$

Here, $T_j^{(\mathrm{pr})}$ is the projection of A onto the space spanned by the Lanczos basis matrix V_j, i.e.,

$$T_j^{(\mathrm{pr})} = V_j^* A V_j. \tag{3.36}$$

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 $T_j^{(\mathrm{pr})}$ is not computed via its definition (3.36). Instead, we use the formula

$$T_j^{(\mathrm{pr})} = T_j + \left(T_j^{(\mathrm{d})}\right)^*. \tag{3.37}$$

By (3.37), we simply have to conjugate and transpose the part of T_j outside its banded part and add it to T_j in order to obtain $T_j^{(\mathrm{pr})}$. To show that (3.37) indeed holds true, note that by multiplying (3.33) from the left by V_j^* and by using the orthogonality relations (3.30) and (3.32), we get

$$T_j^{(\mathrm{pr})} = V_j^* A V_j = T_j + S_j, \quad \text{where} \quad S_j = V_j^* \widehat{V}_j^{(\mathrm{dl})}. \tag{3.38}$$

Since the matrices $T_j^{(\mathrm{pr})}$ and $T_j^{(\mathrm{b})}$ are both Hermitian, it follows from (3.34) that $S_j = \left(T_j^{(\mathrm{d})}\right)^*$ in (3.38). Thus (3.38) reduces to (3.37).

For example, for T_{15} in (3.35) the associated matrix $T_{15}^{(\mathrm{pr})} = T_{15} + \left(T_{15}^{(\mathrm{d})}\right)^*$ is of the form

$$T_{15}^{(\mathrm{pr})} = \begin{bmatrix}
* & * & * & * & * & * & & & & & & & & & \\
* & * & * & * & * & * & * & & & & & & & & \\
* & * & * & * & * & * & * & \mathtt{d} & \mathtt{d} & \mathtt{d} & \mathtt{d} & \mathtt{d} & \mathtt{d} & \mathtt{d} & \mathtt{d} \\
* & * & * & * & * & * & * & * & & & & & & & \\
* & * & * & * & * & * & * & * & * & & & & & & \\
* & * & * & * & * & * & * & * & * & * & & & & & \\
 & * & * & * & * & * & * & * & * & * & \mathtt{d} & \mathtt{d} & \mathtt{d} & \mathtt{d} & \mathtt{d} \\
 & & \overline{\mathtt{d}} & * & * & * & * & * & * & * & * & & & & \\
 & & \overline{\mathtt{d}} & & * & * & * & * & * & * & * & * & & & \\
 & & \overline{\mathtt{d}} & & & * & * & * & * & * & * & * & \mathtt{d} & \mathtt{d} & \mathtt{d} \\
 & & \overline{\mathtt{d}} & & & & \overline{\mathtt{d}} & * & * & * & * & * & * & & \\
 & & \overline{\mathtt{d}} & & & & \overline{\mathtt{d}} & & * & * & * & * & * & * & \\
 & & \overline{\mathtt{d}} & & & & \overline{\mathtt{d}} & & & \overline{\mathtt{d}} & * & * & * & * & * \\
 & & \overline{\mathtt{d}} & & & & \overline{\mathtt{d}} & & & \overline{\mathtt{d}} & & * & * & * & * \\
 & & \overline{\mathtt{d}} & & & & \overline{\mathtt{d}} & & & \overline{\mathtt{d}} & & & * & * & *
\end{bmatrix}. \tag{3.39}$$

Here, the $\overline{\mathtt{d}}$'s below the banded part were obtained by conjugating and transposing the corresponding entries above the banded part in (3.35). We remark that, in (3.39), the

entries $\mathtt{d}$ and $\overline{\mathtt{d}}$ outside the banded part are usually small. More precisely, if the deflation criterion (3.43) below is used, then the absolute values of all $\mathtt{d}$'s and $\overline{\mathtt{d}}$'s is bounded by the deflation tolerance $\mathtt{dtol}$. Even though these entries are small, simply setting them to zero would introduce unnecessary errors. Indeed, the projection property (3.36) for $T_j^{(\mathrm{pr})}$ holds true only if all entries $\mathtt{d}$ and $\overline{\mathtt{d}}$ outside the banded part are included in $T_j^{(\mathrm{pr})}$.

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 (3.33) for the Lanczos vectors reduces to

$$AV_j = V_j T_j + \widehat{V}_j^{(\mathrm{dl})}. \tag{3.40}$$

Using (3.38), we can rewrite (3.40) as follows:

$$AV_j - V_j T_j^{(\mathrm{pr})} = \left(I - V_j V_j^*\right) \widehat{V}_j^{(\mathrm{dl})}. \tag{3.41}$$

Now let $\theta_i^{(j)}$ and $z_i^{(j)}$ be any of the eigenpairs of $T_j^{(\mathrm{pr})}$, and assume that $z_i^{(j)}$ is normalized so that $\left\| z_i^{(j)} \right\|_2 = 1$. 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 (3.41) 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:

$$\left\| A x_i^{(j)} - \theta_i^{(j)} x_i^{(j)} \right\|_2 = \left\| \left(I - V_j V_j^*\right) \widehat{V}_j^{(\mathrm{dl})} z_i^{(j)} \right\|_2 \leq \left\| \widehat{V}_j^{(\mathrm{dl})} \right\|_2. \tag{3.42}$$

In particular, if only exact deflation is performed, then $\widehat{V}_j^{(\mathrm{dl})} = 0$ and (3.42) shows that each eigenvalue $\theta_i^{(j)}$ of $T_j^{(\mathrm{pr})}$ is indeed an eigenvalue of A. For general deflation, $\widehat{V}_j^{(\mathrm{dl})} \neq 0$, however $\left\| \widehat{V}_j^{(\mathrm{dl})} \right\|_2$ is of the order of the deflation tolerance. More precisely, if the deflation check (3.43) below is used, then

$$\left\| \widehat{V}_j^{(\mathrm{dl})} \right\|_2 \leq \sqrt{p}\, \mathtt{dtol},$$

where $\mathtt{dtol}$ denotes the deflation tolerance.

3.6.3 The Algorithm

A complete statement of the band Lanczos algorithm for Hermitian matrices A is as follows.

ALGORITHM 3.12: **Hermitian band Lanczos**

Input: *Starting vectors* $b_1, b_2, \ldots, b_p$.

Output: *The* $j \times j$ *Lanczos matrix* $T_j^{(\mathrm{pr})}$.

(1) *Set* $\widehat{v}_k = b_k$ *for* $k = 1, 2, \ldots, p$. *Set* $p_c = p$ *and* $\mathcal{I} = \emptyset$.
(2) *For* $j = 1, 2, \ldots,$ *until convergence or* $p_c = 0$ *do*:
(3) *Compute* $\|\widehat{v}_j\|_2$.
(4) *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* $k = j, j+1, \ldots, j + p_c - 1$, *set* $\widehat{v}_k = \widehat{v}_{k+1}$.
d) *Return to Step* (3).
(5) *Set* $t_{j,j-p_c} = \|\widehat{v}_j\|_2$ *and normalize* $v_j = \widehat{v}_j / t_{j,j-p_c}$.
(6) *For* $k = j+1, j+2, \ldots, j+p_c-1$, *set*
$t_{j,k-p_c} = v_j^* \widehat{v}_k$ *and* $\widehat{v}_k = \widehat{v}_k - v_j t_{j,k-p_c}$.
(7) *Compute* $\widehat{v}_{j+p_c} = A v_j$.
(8) *Set* $k_0 = \max\{1, j - p_c\}$. *For* $k = k_0, k_0+1, \ldots, j-1$, *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* $T_j^{(\mathrm{pr})} = T_j + S_j = [\,t_{ik}\,]_{i,k=1,2,\ldots,j} + [\,s_{ik}\,]_{i,k=1,2,\ldots,j}$.
(12) *Test for convergence.*

Next, we discuss some of the steps of Algorithm 3.12 in more detail.

(4) Generally, the decision if $\widehat{v}_j$ needs to be deflated should be based on checking if

$$\|\widehat{v}_j\|_2 \leq \texttt{dtol}, \tag{3.43}$$

where `dtol` is a suitably chosen deflation tolerance. The vector $\widehat{v}_j$ is then deflated if (3.43) 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 $T_j^{(\mathrm{d})}$ of T_j, and the current block size is updated to $p_c = p_c - 1$. If $p_c = 0$, then the block-Krylov sequence (3.28) is exhausted, and the algorithm terminates naturally. If $p_c > 0$, the vector $\widehat{v}_j$ is deleted, the index $k+1$ of each of the remaining candidate vectors $\widehat{v}_{k+1}$ is reset to k, and finally, the algorithm returns to Step (3). If (3.43) is not satisfied, then no deflation is necessary and the algorithm proceeds with Step (5).

Usually, in (3.43), one will choose a small tolerance `dtol`, such as the square root of machine precision. However, `dtol` does not have to be small; Algorithm 3.12 and its properties remain correct for *any* choice of `dtol`. Note that the algorithm performs only exact deflation if one sets `dtol` $= 0$ in (3.43).

(5) The vector $\widehat{v}_j$ has passed the deflation check and is now normalized to become the next Lanczos vector v_j. The normalization is such that

$$\|v_j\|_2 = 1 \quad \text{for all} \quad j.$$

(6) The remaining candidate vectors, $\widehat{v}_{j+1}, \widehat{v}_{j+1}, \ldots, \widehat{v}_{j+p_c-1}$, are explicitly orthogonalized against the latest Lanczos vector v_j.

(7) The block-Krylov sequence is advanced by computing a new candidate vector, $\widehat{v}_{j+p_c}$, as the A-multiple of the latest Lanczos vector v_j.

(8) The vector $\widehat{v}_{j+p_c}$ is orthogonalized against the previous Lanczos vectors $v_{k_0}, v_{k_0+1}, \ldots, v_{j-1}$, where $k_0 = \max\{1, j - p_c\}$. Here, we exploit the fact that the matrix $T_j^{(\mathrm{pr})}$ 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 $\widehat{v}_{j+p_c}$ 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 $T_{j-1}^{(\mathrm{pr})}$ of the previous iteration $j-1$, to yield the current matrix $T_j^{(\mathrm{pr})}$. Here, we use the convention that entries t_{ik} and s_{ik} that are not explicitly defined in Algorithm 3.12 are set to be zero.

(12) To test for convergence, the eigenvalues $\theta_i^{(j)}$, $i = 1, 2, \ldots, j$, of the Hermitian $j \times j$ matrix $T_j^{(\mathrm{pr})}$ 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.

3.6.4 Variants

A band Lanczos method for Hermitian matrices was first proposed in [205]. The Hermitian band Lanczos Algorithm 3.12 described here, however, is somewhat different in that it uses both the first j Lanczos vectors (3.29) and the candidate vectors (3.31) for the following p_c Lanczos vectors. This allows to only build the $j \times j$ matrix $T_j^{(\mathrm{pr})}$. The algorithm in [205] 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 $T_j^{(\mathrm{pr})}$. The version of the band Lanczos method described here was first derived in [90] as a special case of the general nonsymmetric band Lanczos method proposed in [77].

The version of the band Lanczos method stated as Algorithm 3.12 performs all multiplications with A as matrix-vector products with single vectors. However, at any stage of the algorithm, it is also possible to pre-compute 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 3.12, we compute the matrix-matrix product

$$A \left[\begin{array}{ccccc} v_j & \widehat{v}_{j+1} & \widehat{v}_{j+2} & \cdots & \widehat{v}_{j+p_c-1} \end{array} \right].$$

This gives us the vector $\widehat{v}_{j+p_c} = Av_j$, which is used in the remaining steps of iteration j, as well as the vectors

$$A\widehat{v}_{j+1}, A\widehat{v}_{j+2}, \ldots, A\widehat{v}_{j+p_c-1}. \tag{3.44}$$

In the following $p_c - 1$ iterations, we will need the vectors

$$Av_{j+1}, Av_{j+2}, \ldots, Av_{j+p_c'-1}. \tag{3.45}$$

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 (3.45) we need are not quite the ones (3.44) we computed. However, the vectors (3.44) and (3.45) are connected by the relation

$$\begin{aligned} &\left[\begin{array}{cccc} A\widehat{v}_{j+1} & A\widehat{v}_{j+2} & \cdots & A\widehat{v}_{j+p_c-1} \end{array}\right] \\ &\qquad = \left[\begin{array}{cccc} Av_{j+1} & Av_{j+2} & \cdots & Av_{j+p_c'-1} \end{array}\right] T_{j+1:j+p_c'-1,j-p_c+1:j-1}. \end{aligned} \tag{3.46}$$

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 $j+1,\ldots,j+p_c'-1$ and columns $j-p_c+1,\ldots,j-1$ of $T_{j+p_c'-1}$. If $p_c' < p_c$, then some of the $\widehat{v}_k$'s in (3.46) will lead to deflated vectors. Let $\widehat{v}_{j_1}, \widehat{v}_{j_2}, \ldots, \widehat{v}_{j_{p_c'-1}}$, be the vectors that are not deflated. Furthermore, let $\widetilde{T}$ 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 $\widehat{v}_k$ vectors in (3.46). With these notations, by (3.46), we have

$$\left[\begin{array}{cccc} A\widehat{v}_{j_1} & A\widehat{v}_{j_2} & \cdots & A\widehat{v}_{j_{p_c'-1}} \end{array}\right] = \left[\begin{array}{cccc} Av_{j+1} & Av_{j+2} & \cdots & Av_{j+p_c'-1} \end{array}\right] \widetilde{T}. \tag{3.47}$$

The matrix $\widetilde{T}$ is nonsingular and upper triangular. Therefore, using (3.47), we can obtain each of the vectors Av_{j+k}, $k = 1, 2, \ldots, p_c' - 1$, in (3.45) as a suitable linear combination of k of the pre-computed vectors (3.44).

For Hermitian positive definite matrices A, yet another variant of Algorithm 3.12, based on coupled recurrences, was proposed in [15]. 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 $T_j^{(\mathrm{pr})}$ may become negative; see [14, 15] for examples. In exact arithmetic, this would be impossible, since the positive definiteness of A implies that $T_j^{(\mathrm{pr})} = V_j^* A V_j$ is also positive definite. However, in finite-precision arithmetic, round-off may cause the matrix $T_j^{(\mathrm{pr})}$ generated by Algorithm 3.12 to be slightly indefinite. This can be avoided easily by using coupled recurrences to generate a Cholesky factor of $T_j^{(\mathrm{pr})}$, instead of $T_j^{(\mathrm{pr})}$ itself. More precisely, the recurrences summarized in (3.33) are replaced by coupled recurrences of the form

$$\begin{aligned} AP_j &= V_j L_j D_j + \left[\begin{array}{ccccccc} 0 & \cdots & 0 & \widehat{v}_{j+1} & \widehat{v}_{j+2} & \cdots & \widehat{v}_{j+p_c} \end{array}\right] + \widehat{V}_j^{(\mathrm{dl})}, \\ V_j &= P_j U_j. \end{aligned} \tag{3.48}$$

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 (3.48), 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 [34]), then the resulting algorithm is mathematically equivalent to the band Lanczos method described here. However, the band Lanczos method has two distinguished 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 $T_j^{(\mathrm{pr})}$ that is slightly sparser than the one produced by the block Lanczos method.

3.7 Jacobi-Davidson methods

G. Sleijpen and H. van der Vorst

The Lanczos method (see Section 3.4) is quite effective to compute eigenvalues in the ends of the spectrum of A if these eigenvalue are well separated from the remaining spectrum, or when it can be applied to $(A-\sigma I)^{-1}$, for some reasonable guess σ for an eigenvalue.

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 [227] offer an attractive alternative.

3.7.1 Basic Theory

The Jacobi-Davidson method is based on a combination of two basic principles. The first one is to apply a Ritz-Galerkin approach for the eigenproblem $Ax = \lambda x$, with respect to some given subspace spanned by an orthonormal basis v_1, ..., v_m. The usage of other than Krylov subspaces was suggested by Davidson [39] who also suggested specific choices, different from the ones that we will take, for the construction of orthonormal basis vectors v_j. The Ritz-Galerkin condition is

$$AV_m s - \theta V_m s \perp \{v_1, \ldots, v_m\},$$

which leads to the reduced system

$$V_m^* A V_m s - \theta s = 0, \tag{3.49}$$

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 $(\theta_j^{(m)}, z_j^{(m)} \equiv V_m s_j^{(m)})$ 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 the play. The idea goes back to Jacobi [119]. Suppose that we have an eigenvector approximation $z_j^{(m)}$ for an eigenvector x corresponding to a given eigenvalue λ. Then

Jacobi suggested (in the original paper for strongly diagonally dominant matrices) to compute the orthogonal correction t for $z_j^{(m)}$ so that

$$A(z_j^{(m)} + t) = \lambda(z_j^{(m)} + t).$$

Since $t \perp z_j^{(m)}$, we can restrict ourselves to the subspace orthogonal to $z_j^{(m)}$. The operator A restricted to that subspace is given by

$$(I - z_j^{(m)} z_j^{(m)^*}) A (I - z_j^{(m)} z_j^{(m)^*}),$$

and we find that t satisfies the equation

$$(I - z_j^{(m)} z_j^{(m)^*})(A - \lambda I)(I - z_j^{(m)} z_j^{(m)^*})t = -(A - \theta_j^{(m)} I) z_j^{(m)}.$$

In practical situations we do not know λ 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 $t^{(m)} \perp z_j^{(m)}$:

$$(I - z_j^{(m)} z_j^{(m)^*})(A - \theta_j^{(m)} I)(I - z_j^{(m)} z_j^{(m)^*})t^{(m)} = -r_m \equiv -(A - \theta_j^{(m)} I) z_j^{(m)}. \quad (3.50)$$

This correction equation is solved only approximately and its approximate solution $\widetilde{t}^{(m)}$ 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 (3.49) we conclude that $r_m \perp \{v_1, \ldots, v_m\}$, and in particular that $r_m \perp z_j^{(m)}$, so that the Jacobi-Davidson correction equation represents a consistent linear system.

It can be shown that the exact solution of (3.50) 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 [227] it is suggested to solve equation (3.50) only approximately, for instance by some steps of MINRES [188], with an appropriate preconditioner K for $A - \theta_j^{(m)} I$, if available, but in fact any approximation technique for $t^{(m)}$ is formally allowed, provided that the projectors $(I - z_j^{(m)} z_j^{(m)^*})$ are taken into account.
In our templates we will present ways to approximate $t^{(m)}$ with Krylov subspace methods.

We will now discuss how to use preconditioning for an iterative solver like MINRES, GMRES, or CGS, applied with equation (3.50). 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 θ. Of course, the preconditioner K has to be restricted to the subspace orthogonal to $z_j^{(m)}$ as well, which means that we have to work effectively with

$$\widetilde{K} = (I - z_j^{(m)} z_j^{(m)^*}) K (I - z_j^{(m)} z_j^{(m)^*}).$$

This may seem unnecessarily complicated, and at first sight it seems also to involve a lot of overhead because of the projections $(I - z_j^{(m)} z_j^{(m)^*})$ that encircle the matrix vector operations. But it all amounts to a handful of rather simple operations, as we will show now.

We will from now on 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 (3.50). Since the starting vector is in the subspace orthogonal to $z_j^{(m)}$, all iteration vectors for the Krylov solver will be in that space. In that subspace we have to compute the vector $u = \widetilde{K}^{-1}\widetilde{A}v$, for a vector v supplied by the Krylov solver. We will do this in two steps and first we have to compute

$$\begin{aligned} \widetilde{A}v &= (I - z_j^{(m)} z_j^{(m)^*})(A - \theta_j^{(m)} I)(I - z_j^{(m)} z_j^{(m)^*})v \\ &= (I - z_j^{(m)} z_j^{(m)^*})y \\ \text{with } \ y &= (A - \theta_j^{(m)} I)v \ \text{ since } \ z_j^{(m)^*} v = 0 \ . \end{aligned}$$

Then, with left-preconditioning we have to solve $u \perp z_j^{(m)}$ from

$$\widetilde{K}u = (I - z_j^{(m)} z_j^{(m)^*})y.$$

Since $u \perp z_j^{(m)}$, it follows that u satisfies $Ku = y - \alpha z_j^{(m)}$ or $u = K^{-1}y - \alpha K^{-1} z_j^{(m)}$. The condition $u \perp z_j^{(m)}$, leads to

$$\alpha = \frac{z_j^{(m)^*} K^{-1} y}{z_j^{(m)^*} K^{-1} z_j^{(m)}}.$$

The vector $\widehat{y} = K^{-1}y$ is solved from $K\widehat{y} = y$ and, likewise, $\widehat{z} = K^{-1} z_j^{(m)}$ is solved from $K\widehat{z} = z_j^{(m)}$. Note that the last equation has to be solved only once in an iteration process for equation (3.50), so that effectively $i_S + 1$ operations with the preconditioner are required for i_S iterations of the linear solver. Note also that one 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 $(I - z_j^{(m)} z_j^{(m)^*})$. This has been worked out in the solution template, given in Alg. 3.14. 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 Alg. 3.15.
For more details on preconditioning of the correction equation, see [228].

If we form an approximation for $t^{(m)}$ in (3.50) as $\widetilde{t}^{(m)} = K^{-1}r - \alpha K^{-1} z_j^{(m)}$, with α such that $\widetilde{t}^{(m)} \perp z_j^{(m)}$, and without acceleration by an iterative solver, then we obtain a process which was suggested by Olsen et al [183].

3.7.2 The Basic Algorithm

The basic form of the Jacobi-Davidson algorithm is given in Alg. 3.13. Later on we will describe more sophisticated variants with restart and other strategies.

In each iteration of this algorithm an approximated eigenpair (θ, z) 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 $Az - \theta z$ is below a given threshold ϵ.

ALGORITHM 3.13: **Jacobi-Davidson Method for** $\lambda_{\max}(A)$

(1) *Start with* $t = v_0$, *starting guess*
for $m = 1, \ldots$
(2) **for** $j = 1, .., m-1$
$t = t - (t^* v_j) v_j$
$v_m = t/\|t\|_2$, $w_m = A v_m$
(3) **for** $j = 1, .., m$
$M_{j,m} = v_j^* w_m$
(4) *Compute the largest eigenpair* $Ms = \theta s$
of the m *by* m *Hermitian matrix* M, $(\|s\|_2 = 1)$
$z = V_m s$
$\widehat{z} = W_m s$
$r = \widehat{z} - \theta\, z$
(5) **Stop if** $\|r\|_2 \leq \varepsilon$
(8) *Solve (approximately) a* $t \perp z$ *from*
$(I - zz^*)(A - \theta I)(I - zz^*)t = -r$

We will now describe some implementation details, referring to the respective phases in Alg. 3.13.

(1) This is the initialization phase of the process. The search subspace is each time expanded by a vector t, and we start this process by a given vector. 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 as a starting vector. This gives some confidence that the wanted eigenvector has a component in the starting vector, which is necessary for detection of the eigenvector.

(2) The first two lines represent the modified Gram-Schmidt process for the orthogonalization of the new vector t with respect to the set $v_1, \ldots, v_{m-1}$. If $m = 1$ then this is an empty loop. Let t_{in} represent the vector before the start of modified Gram-Schmidt, and t_{out} the vector that results at completion of phase (2). It is advisable (see [38]) to repeat the Gram-Schmidt process one time if $\|t_{out}\|_2 / \|t_{in}\|_2 \leq \kappa$, where κ 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 Alg. 3.16.

(3) Computation of the m-th column of the upper triangular part of the matrix $M \equiv V_m^* A V_m$.

(4) 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 Section 3.2).

The vector $\widehat{z}$ may be alternatively computed as $\widehat{z} = Az$, depending on which way is cheapest. The choice is between a m-fold update and another multiplication with A; if A has less than m nonzero elements on average per row, then the computation via Az is preferable. If $\widehat{z}$ is computed as Az then it is not necessary to store the vectors w_j.

(5) The algorithm is terminated if $\|Az - \theta z\|_2 \leq \epsilon$. In that case A has an eigenvalue λ for which $|\lambda - \theta| \leq \epsilon$. For the corresponding normalized eigenvector there is a similar bound on the angle, provided that λ is well separated from the other eigenvalues of A, see (3.4).

(8) The approximate solution for the expansion vector t can be computed with a Krylov solver, for instance, MINRES, SYMMLQ, or CGS. 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 Alg. 3.14. The right-preconditioned case, which is slightly more expensive, is covered by the template in Alg. 3.15. For iterative Krylov subspace solvers see [18]. The approximate solution has to be orthogonal to z, but that is automatically the case with Krylov solvers if one starts with an initial guess orthogonal to z, 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 [39] suggested to take $t = (\mathrm{diag}(A) - \theta I)^{-1} r$, but in this case t is not orthogonal with respect to z. 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 for v the latest value of z. We will describe a variant of the Jacobi-Davidson algorithm with a more sophisticated restart strategy in Section 3.7.3.

Note that most of the computationally intensive operations, i.e., those operations of which the cost is proportional to n, can be done easily in parallel computation mode. Also, the multiple vector updates can be done by the appropriate BLAS2 routines (see Section ??).

ALGORITHM 3.14: **Approximate solution of the Jacobi-Davidson correction equation with left-preconditioning**

"Solve" with left preconditioner $\widetilde{K} = (I - zz^*)K(I - zz^*)$*:*

(8a) *Solve* w *from* $Kw = z$, $\mu = z^*w$
Compute $\widetilde{r} \equiv \widetilde{K}^{-1}r$ *as:*
(b') *solve* x *from* $Kx = r$
(c') $\widetilde{r} = x - \frac{z^*x}{\mu} w$
Apply Krylov subspace method with start $t_0 = 0$,
with operator $\widetilde{K}^{-1}\widetilde{A}$, *and right-hand side* $-\widetilde{r}$,
$u = \widetilde{K}^{-1}\widetilde{A}v$ *for arbitrary* v *is computed as:*
(a) $y = (A - \theta I)v$
(b) *solve* x *from* $Kx = y$
(c) $u = x - \frac{z^*x}{\mu} w$

ALGORITHM 3.15: **Approximate solution of the Jacobi-Davidson correction equation with right-preconditioning**

"Solve" with right preconditioner $\widetilde{K} = (I - zz^*)K(I - zz^*)$*:*

(8a) *Solve* w *from* $Kw = z$, $\mu = z^*w$
Apply Krylov subspace method with start $t_0 = 0$,
with operator $\widetilde{A}\widetilde{K}^{-1}$, *and right-hand side* $-r$,
$u = \widetilde{A}\widetilde{K}^{-1}v$ *for arbitrary* v *is computed as:*
(a) *solve* x *from* $Kx = v$
(b) $y = x - \frac{z^*x}{\mu} w$
(c) $u = (I - zz^*)(A - \theta I)y$
The approximate solution t *should be back-transformed as:*
(a'') *solve* x *from* $Kx = t$
(b'') $t = x - \frac{z^*x}{\mu} w$

ALGORITHM 3.16: **Modified Gram-Schmidt orthogonalization with refinement**

(2a) *Select a value for* κ *less than* 1, *say* $\kappa = .25$
$\tau_{in} = \|t\|_2$
for $j = 1, .., m - 1$
$t = t - (t^*v_j)v_j$
if $\|t\|_2/\tau_{in} \leq \kappa$
for $j = 1, .., m - 1$
$t = t - (t^*v_j)v_j$

In the coming subsections we will describe more sophisticated variants of the Jacobi-Davidson algorithm. In Section 3.7.3 we will introduce a variant that allows for restarts, which is very convenient if one wants to keep the dimensions of the involved subspaces limited. The variant is also suitable for a restart after an eigenpair has been discovered,

in order to locate a 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 Section 3.7.4 we will describe a variant of Jacobi-Davidson that is suitable for the computation of interior eigenvalues of A.

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 k 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	$m+1$ dots, m updates in iteration m
projected system	m dots
eigensystem projected system	$\mathcal{O}(m^3)$
residual	1 Matrix vector product, 1 update or $m-$fold update
approx. eigenvector	$m-$fold update
correction equation	depends on choice solver

3.7.3 Jacobi-Davidson with restart and deflation

3.7.3.1 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.

3.7.3.2 Deflation

After convergence of a Ritz value to an eigenvalue, the remaining part of the current subspace will allready have rich components in nearby eigenpairs, since we have selected

in all steps the Ritz vectors 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 that the old eigenvector will re-enter the computational process, we make the new search vectors in the Jacobi-Davidson algorithm explicitly orthogonal to the converged eigenvectors. This technique is called explicit *deflation*. We will discuss this in slightly more detail.

Let $q_1, \ldots, q_{k-1}$ denote the converged eigenvectors and let us assume that these vectors are orthonormal. The matrix Q_{k-1} has the vectors q_j as its columns. In order to find the next eigenvector q_k, we apply the Jacobi-Davidson algorithm to the deflated matrix

$$(I - Q_{k-1}Q_{k-1}^*)\, A\, (I - Q_{k-1}Q_{k-1}^*),$$

and this leads to a correction equation of the form

$$P_m(I - Q_{k-1}Q_{k-1}^*)\,(A - \theta_j^{(m)} I)\,(I - Q_{k-1}Q_{k-1}^*)P_m t^{(m)} = -r_m, \tag{3.51}$$

with $P_m \equiv (I - z_j^{(m)} z_j^{(m)^*})$, that has to be solved for the correction $t^{(m)}$ to each new eigenvector approximation $z_j^{(m)}$, with corresponding Ritz value $\theta_j^{(m)}$. In [82] it is shown, by numerical evidence, that the explicit deflation against the vectors represented by Q_{k-1} is highly recommended for the correction equation, but is 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).

3.7.3.3 Preconditioning

Preconditioning for an iterative solver like MINRES, GMRES, or CGS, applied with equation (3.51) is only slightly more complicated than in the single vector case (cf. Sec. 3.7.1). Suppose that we have a left preconditioner K available for the operator $A - \theta_j^{(m)} I$. Let $\widetilde{Q}$ denote the matrix Q_{k-1} expanded with $z_j^{(m)}$ as its k-th column. In this case, the preconditioner K has to be restricted to the subspace orthogonal to $\widetilde{Q}$ as well, which means that we have to work effectively with

$$\widetilde{K} \equiv (I - \widetilde{Q}\widetilde{Q}^*)K(I - \widetilde{Q}\widetilde{Q}^*).$$

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 (3.51). Since the starting vector is in the subspace orthogonal to $\widetilde{Q}$, all iteration vectors for the Krylov solver will be in that space. In that subspace we have to compute the vector $z = \widetilde{K}^{-1}\widetilde{A}v$, for a vector v supplied by the Krylov solver. This is done in two steps and first we compute

$$\begin{aligned} \widetilde{A}v &= (I - \widetilde{Q}\widetilde{Q}^*)(A - \theta_j^{(m)} I)(I - \widetilde{Q}\widetilde{Q}^*)v \\ &= (I - \widetilde{Q}\widetilde{Q}^*)y \\ \text{with } \; y &= (A - \theta_j^{(m)} I)v \;\; \text{ since } \;\; \widetilde{Q}^* v = 0\ . \end{aligned}$$

Then, with left-preconditioning we have to solve $z \perp \widetilde{Q}$ from

$$\widetilde{K}z = (I - \widetilde{Q}\widetilde{Q}^*)y.$$

Since $\widetilde{Q}^*z = 0$, it follows that z satisfies $Kz = y - \widetilde{Q}\vec{\alpha}$ or $z = K^{-1}y - K^{-1}\widetilde{Q}\vec{\alpha}$. The condition $\widetilde{Q}^*z = 0$ leads to

$$\vec{\alpha} = (\widetilde{Q}^*K^{-1}\widetilde{Q})^{-1}\widetilde{Q}^*K^{-1}y.$$

The vector $\widehat{y} = K^{-1}y$ is solved from $K\widehat{y} = y$ and, likewise, $W = K^{-1}\widetilde{Q}$ is solved from $KW = \widetilde{Q}$. Note that the last set of equations has to be solved only once in an iteration process for equation (3.51), so that effectively $i_S + k$ operations with the preconditioner are required for m iterations of the linear solver. Note also that one matrix vector multiplication with the left-preconditioned operator, in an iteration of the Krylov solver, requires only one operation with $\widetilde{Q}^*$ and $K^{-1}\widetilde{Q}$, instead of the four actions of the projector operator $(I - \widetilde{Q}\widetilde{Q}^*)$. This has been worked out in the solution template, given in Alg. 3.18. 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 [?]).

3.7.3.4 Jacobi-Davidson template with restart and deflation for exterior eigenvalues

The complete algorithm for the Jacobi-Davidson method, that includes restart with a number of Ritz values, and deflation for the computation of a number of eigenpairs is called JDQR [82], since it can be interpreted as an iterative approach for the QR algorithm. The template for this algorithm is given in Alg. 3.17.

ALGORITHM 3.17: **Jacobi-Davidson for k_{max} exterior eigenvalues**

Start with v_0 starting vector and τ target value.
(1) $t = v_0$, $k = 0$, $m = 0$; $Q = [\,]$,
while $k < k_{max}$
(2) **for** $i = 1, .., m$
$t = t - (v_i^* t) v_i$
$m = m + 1$; $v_m = t/\|t\|_2$, $w_m = A v_m$
(3) **for** $i = 1, .., m$
$M_{i,m} = v_i^* w_m$
(4) *Make eigendecomposition $MS = S\Theta$*
of the m by m Hermitian matrix M.
Sort the pairs: $|\theta_i - \tau| \geq |\theta_{i-1} - \tau|$
$z = V s_1$, $\widetilde{w} = W s_1$, $r = \widetilde{w} - \theta_1 z$
(5) **while** $\|r\|_2 \leq \epsilon$
$\widetilde{\theta}_{k+1} = \theta_1$, $Q = [Q, z]$, $k = k + 1$
if $k = k_{max}$ **then** *STOP*
(6) $m = m - 1$, $M = 0$
for $i = 1, .., m$
$v_i = V s_{i+1}$, $w_i = W s_{i+1}$,
$M_{i,i} = \theta_{i+1}$, $s_i = e_i$, $\theta_i = \theta_{i+1}$
$z = v_1$, $r = w_1 - \theta_1 z$
(7) **if** $m \geq m_{max}$ **then**
$M = 0$
for $i = 2, .., m_{min}$
$v_i = V s_i$, $w_i = W s_i$, $M_{i,i} = \theta_i$
$v_1 = z$, $w_1 = \widetilde{w}$, $M_{1,1} = \theta_1$, $m = m_{min}$
$\theta = \theta_1$, $\widetilde{Q} = [Q, z]$
(8) *Solve t $(\perp \widetilde{Q})$ (approximately) from:*
$(I - \widetilde{Q}\widetilde{Q}^*)(A - \theta I)(I - \widetilde{Q}\widetilde{Q}^*)t = -r$

In principle, this algorithm computes the k_{max} eigenvalues closest to a specified target value τ. 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 Section 3.7.4. We will comment on some parts of the algorithm in view of our discussions in previous subsections.

(1) Initialization phase. Search subspace is initialized with t.

(2) 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 Alg. 3.16. If $m = 1$ then this is an empty loop.

(3) We compute only the upper triangular part of the Hermitian matrix M (of order m).

(4) The eigenproblem for the m by m matrix M can be solved by a standard eigensolver for dense Hermitian eigenproblems. 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 then the usage of harmonic Ritz values is advocated, see Section 3.7.4.

(5) The stopping criterion is to accept an eigenvector approximation as soon as the norm of the residual (for the normalized eigenvector 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, provided that the associated eigenvalue is well separated from the other eigenvalues.

(6) After convergence of an eigenpair, we continue the search for a next eigenpair, with the unconverged eigenvectors as a basis for the initial search space.

(7) 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 τ.

(8) We have collected the locked (converged) eigenvectors in Q, and the matrix $\widetilde{Q}$ is Q expanded with the current eigenvector approximation z. This is done in order to obtain a more compact formulation; the correction equation is equivalent to the one in (3.51). The new correction v has to be orthogonal to the columns of Q as well as to z.
Of course, the correction equation can be solved by any suitable process, for instance a preconditioned Krylov subspace method. Because of the occurrence of $\widetilde{Q}$ one has to be careful with the usage 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 Alg. 3.18 for a Template. Make sure that the starting vector for an iterative solver satisfies the orthogonality constraints. Note that significant savings can be made in Alg. 3.18 if K is kept the same for a (few) Jacobi-Davidson iterations. In that case columns of E can be saved from previous steps. Also the matrix $\mathcal{M}$ in Alg. 3.18 can be updated from previous steps, as well as its $\mathcal{LU}$ decomposition.

ALGORITHM 3.18: **Approximate solution of the deflated Jacobi-Davidson correction equation**

"Solve" with left preconditioner $\widetilde{K} = (I - \widetilde{Q}\widetilde{Q}^*)K(I - \widetilde{Q}\widetilde{Q}^*)$*:*

(8a) *Solve* E *from* $KE = \widetilde{Q}$
Compute $\mathcal{M} = \widetilde{Q}^*E$
Decompose $\mathcal{M} = \mathcal{L}\mathcal{U}$
Compute $\widetilde{r} \equiv \widetilde{K}^{-1}r$ *as:*
(b′) *solve* x *from* $Kx = r$
(c′) $\vec{\gamma} = \widetilde{Q}^*r$
solve $\vec{\beta}$ *from* $\mathcal{L}\vec{\beta} = \vec{\gamma}$
solve $\vec{\alpha}$ *from* $\mathcal{U}\vec{\alpha} = \vec{\beta}$
(d′) $\widetilde{r} = x - E\vec{\alpha}$
Apply Krylov subspace method with start $t_0 = 0$*,*
with operator $\widetilde{K}^{-1}\widetilde{A}$*, and right-hand side* $-\widetilde{r}$*,*
$u = \widetilde{K}^{-1}\widetilde{A}v$ *for arbitrary* v *is computed as:*
(a) $y = (A - \theta I)v$
(b) *solve* x *from* $Kx = y$
(c) $\vec{\gamma} = \widetilde{Q}^*y$
solve $\vec{\beta}$ *from* $\mathcal{L}\vec{\beta} = \vec{\gamma}$
solve $\vec{\alpha}$ *from* $\mathcal{U}\vec{\alpha} = \vec{\beta}$
(d) $u = x - E\vec{\alpha}$

3.7.4 Jacobi-Davidson for interior Eigenvalues

3.7.4.1 The usage of harmonic Ritz values for interior eigenvalues

If one is heading 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 be even 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 the optimal vector for restart? The answer is given by the so-called *harmonic Ritz* vectors [175, 187, 227].

As we have seen, the Jacobi-Davidson methods generate basis vectors $v_1, \ldots, v_m$, for a subspace $\mathcal{V}_m$. For the projection of A onto this subspace we have to compute the vectors $w_j = Av_j$. 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 $(\widetilde{\theta}^{(m)}, \widetilde{z}^{(m)})$ satisfies

$$A\widetilde{z}^{(m)} - \widetilde{\theta}^{(m)}\widetilde{z}^{(m)} \perp \mathcal{W}_m \equiv \text{span}\{w_1, \ldots, w_m\}, \tag{3.52}$$

for $\widetilde{z}^{(m)} \in V_m$ and $\widetilde{z}^{(m)} \neq 0$. This implies that the harmonic Ritz values are the

eigenvalues of the pencil $(W_m^* A V_m, W_m^* V_m)$, or since $W_m = A V_m$:

$$W_m^* W_m s - \widetilde{\theta}^{(m)} W_m^* V_m s = 0 \, .$$

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 [187] it is shown that for Hermitian A the harmonic Ritz values converge monotonically towards the smallest non-zero 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 τ. Fortunately, the search subspace $\mathcal{V}_m$ for the shifted matrix and the unshifted matrix coincide, which facilitates the computation of harmonic Ritz pairs. The harmonic Ritz vector, for the shifted matrix, corresponding to the harmonic Ritz value closest to τ can be interpreted as to maximize a Rayleigh quotient for $(A - \tau I)^{-1}$. It represents the best information that is available for the wanted eigenvalue, and hence it represents the best candidate as a starting vector after restart.

3.7.4.2 A template for Jacobi-Davidson for interior eigenvalues

An algorithm for Jacobi-Davidson based on harmonic Ritz values and vectors, combined with restart and deflation, is given in Alg. 3.19. The algorithm can be used for the computation of a number of successive eigenvalues immediately to the right of the target value τ.

ALGORITHM 3.19: **Jacobi-Davidson for k_{max} interior eigenvalues at the right side nearest to τ**

(1) $t = v_0$, $k = 0$; $m = 0$; $Q = [\,]$;
while $k < k_{max}$
(2) $w = (A - \tau I)t$
for $i = 1, .., m$
$\gamma = w_i^* w$, $w = w - \gamma w_i$, $t = t - \gamma v_i$
$m = m + 1$,$w_m = w/\|w\|_2$, $v_m = t/\|w\|_2$
(3) **for** $i = 1, .., m$
$M_{i,m} = w_i^* v_m$
(4) *Compute eigendecomposition* $MS = S\Theta$
of the m *by* m *Hermitian matrix* M.
Sort the pairs so that: $\theta_1 \geq \theta_2 \geq \ldots$
$\widetilde{z} = Vs_1$, $\mu = \|\widetilde{z}\|_2$, $z = \widetilde{z}/\mu$, $\vartheta = \theta_1/\mu^2$
$\widetilde{w} = Ws_1$, $r = \widetilde{w}/\mu - \vartheta z$
(5) **while** $\|r\|_2 \leq \epsilon$
$k = k + 1$, $Q = [Q, z]$, $\widetilde{\theta}_k = \vartheta + \tau$
if $k = k_{max}$ **then** *STOP*
(6) $m = m - 1$, $M = 0$
for $i = 1, .., m$
$v_i = Vs_{i+1}$, $w_i = Ws_{i+1}$
$M_{i,i} = \theta_{i+1}$, $s_i = e_i$, $\theta_{i+1} = \theta_i$
$\mu = \|v_1\|_2$, $\vartheta = \theta_2/\mu^2$, $z = v_1/\mu$, $r = w_1/\mu - \theta_r z$
(7) **if** $m \geq m_{max}$ **then**
$M = 0$
for $i = 2, .., m_{min}$
$v_i = Vs_i$, $w_i = Ws_i$, $M_{i,i} = \theta_i$
$w_1 = \widetilde{w}$, $v_1 = \widetilde{z}$, $M_{1,1} = \theta_1$, $m = m_{min}$
(8) $\theta = \vartheta + \tau$, $\widetilde{Q} = [Q, z]$
Solve $t\,(\perp \widetilde{Q})$ *(approximately) from:*
$(I - \widetilde{Q}\widetilde{Q}^*)(A - \theta I)(I - \widetilde{Q}\widetilde{Q}^*)t = -r$

For exterior eigenvalues a simpler algorithm has been described in Section 3.7.3. We will comment on some parts of the algorithm in view of our discussions in previous subsections.

(1) Initialization phase.

(2) The vector $(A-\tau I)t$ 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 the template given in Alg. 3.16.

(3) The values $M_{i,j}$ represent elements of the square m by m matrix $M \equiv W_m^* V_m$. Because M is Hermitian, only the upper triangular part of this matrix is computed.

(4) At this point the eigenpairs for the problem $Ms = \theta s$ should be computed. This can be done with a suitable routine for Hermitian dense matrices from LAPACK.

Note that the harmonic Ritz values are just the inverses of the eigenvalues of M. Then we have to compute the Rayleigh quotient ϑ for Vs_1, and we have to normalize Vs_1, in order to compute a proper residual $r \perp Vs_1$.
We have used that $s_1^* V^*(A - \tau I)Vs_1 = s_1^* M^* s_1 = \theta_1$.

(5) The stopping criterion is to accept an eigenvector approximation as soon as the norm of the residual (for the normalized eigenvector approximation) is below ϵ. This means that we accept inaccuracies in the order of ϵ in the computed eigenvalues, and relative inaccuracies (in norm) in the eigenvectors of $\mathcal{O}(\sqrt{\epsilon})$, provided that the associated eigenvalue is well separated from the others.

(6) This is a restart after convergence of an eigenpair. The restart is slightly more complicated since two subspaces are involved.

(7) 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 τ.

(8) The deflation with converged 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 Alg. 3.18.

3.7.4.3 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 [31], and is discussed, in the context of the Jacobi-Davidson algorithm in [227, page 410].

We use Alg. 3.17, for the computation of the $k_{max} = 10$ largest eigenvalues. The input parameters have been chosen as follows. The starting vector v has entries $v_i = 0.01$, $i = 1, \ldots, 999$, $v_{1000} = 1$. 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 Fig. 3.5. Every time when the norm is less than ϵ, then 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:

1. Top left: this shows what happens when t, in line (8) of Alg. 3.17, is simply taken as $t = -r$. In exact arithmetic, this should deliver the same Ritz values as Arnoldi's algorithm (assuming a similar restart strategy as in Alg. 3.17).

2. Top right: here we have applied a simple preconditioner $K = \mathrm{diag}(A)$, as in Alg. 3.18, without further subspace acceleration, that is we stopped after step (d′). This is equivalent to the method published in [183].

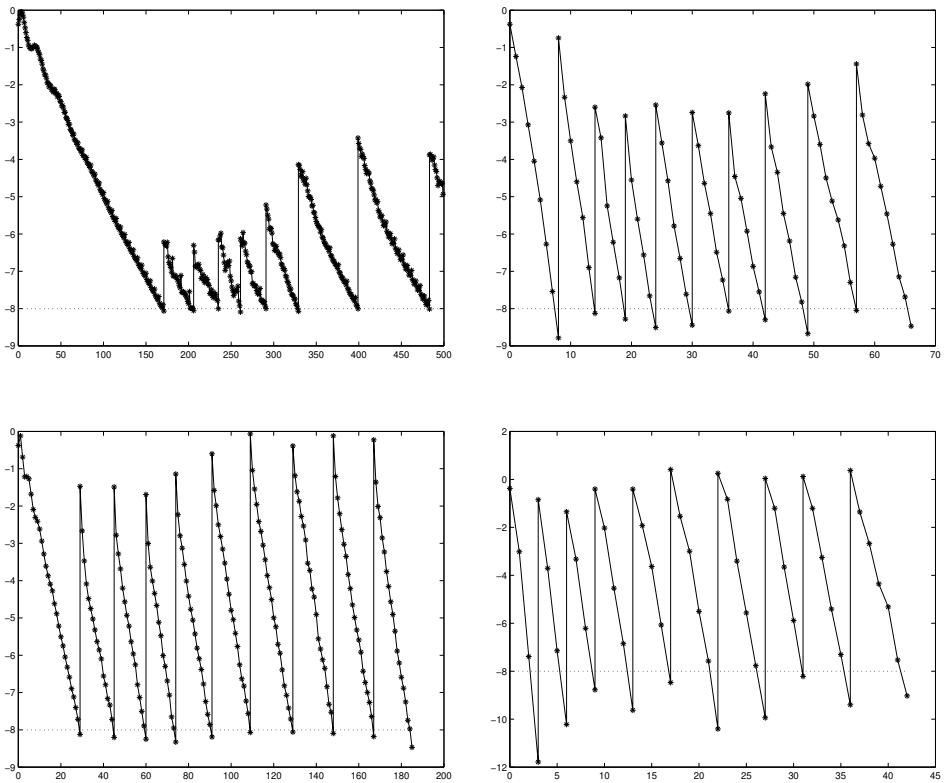


FIGURE 3.5: *Jacobi-Davidson for exterior eigenvalues with several strategies for solving the correction equation.*

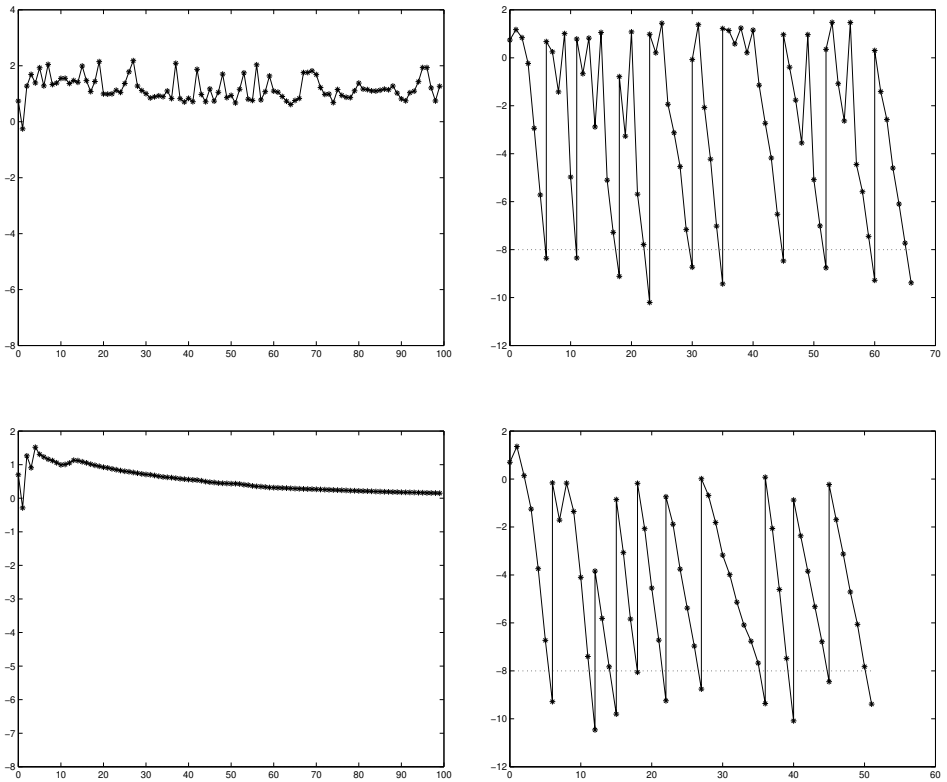


FIGURE 3.6: *Jacobi-Davidson for exterior eigenvalues (top figures) and interior eigenvalues (bottom figures). The correction equations have been solved with 5 steps of plain GMRES (left figures) and with 5 steps of preconditioned GMRES (right figures).*

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 (Alg. 3.17 with $K = I$).

4. Bottom right: here we have used preconditioning as in Alg. 3.17, with $K = \mathrm{diag}(A)$ 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).

In Fig. 3.6, we give the convergence history for interior eigenvalues, as obtained with Alg. 3.17 (top figures) and with Alg. 3.19 (bottom figures), with the following input specifications:
$v_i = 0.01$, $i = 1, \ldots, 999$, $v_{1000} = 1$, $\tau = 900.5$, $\epsilon = 10^{-8}$, $k_{max} = 10$, $m_{min} = 5$, and $m_{max} = 10$. Again, every time when the curve gets below ϵ, then this indicates convergence of an approximated eigenvalue within that tolerance. For all figures, we used 5 steps of GMRES to solve the correction equation in (8). For the left figures, we did not use preconditioning. For the right figures, we preconditioned GMRES with $K = \mathrm{diag}(A) - \tau I$, as in Alg. 3.18

3.7.5 Software Availability

Matlab versions of the algorithms are available from either
`http://www.math.uu.nl/people/sleijpen`
or
`http://www.math.uu.nl/people/vorst`

Fortran implementations of the Jacobi-Davidson method are available via
`http://www.math.uu.nl/people/bomhof`
The Fortran software is of experimental type and does not reflect all insights and possibilities described in this Section.

CHAPTER 4

Generalized Hermitian Eigenproblem

4.1 Introduction

4.2 Transformation Methods

4.3 Single and Multiple Vector Iterations Methods

4.4 Lanczos Method

4.5 Implicity Restarted Lanczos Method

4.6 Jacobi-Davidson Method

CHAPTER 5

Non-Hermitian Eigenvalue Problem

5.1 Introduction

In this chapter we discuss the problem of computing eigenvalues λ (and eigenvectors $x \neq 0$) of a non-Hermitian square matrix A:

$$Ax = \lambda x.$$

More precisely, x is called right eigenvector, and a vector $y \neq 0$ satisfying

$$y^* A = \lambda y^*$$

is called a left eigenvector of A.

In contrast to a Hermitian matrix a non-Hermitian matrix does not have an orthogonal set of eigenvectors, or said in other words, a non-Hermitian matrix A can in general not be transformed by an orthogonal matrix Q to diagonal form $D = Q^* AQ$. Most non-Hermitian matrices can be transformed by a nonorthogonal X to diagonal form $D = X^{-1} AX$, 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 does 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 much as possible. It is well-known that for any non-Hermitian matrix A there exists a unitary matrix Q that transforms it to upper-triangular form

$$R = Q^* AQ. \tag{5.1}$$

The matrix R is called the *Schur form* of A. The eigenvalues of A appear along the diagonal of R. Furthermore, the matrix Q can be permuted in order to have the eigenvalues of A in a prescribed order along the diagonal of R. If z is an eigenvector

of R corresponding to the eigenvalue λ, $Rz = \lambda z$, it follows that

$$A(Qz) = \lambda(Qz),$$

so that Qz is an eigenvector of A corresponding to the eigenvector λ.

Reduction to Schur form can be done but it has its price. Let n be the order of a non-Hermitian matrix A, then for $n \geq 3$, 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 §5.2 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, §5.3.

The largest eigenvalue in absolute value and 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 σ 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 the situation that one can not afford storage for more than two iteration vectors.

Arnoldi methods, §5.4, §5.5, and §5.6.

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 Methods** one starts with an initial guess v and generates an orthonormal basis for a *Krylov subspace* of dimension m:

$$\mathcal{K}^m(A; v) \equiv \text{span}(v, Av, A^2v, \ldots, A^{m-1}v)$$

(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 Arnoldi's method is given in §5.4. 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* and these are discussed in §5.5. Corresponding software is available in the package ARPACK, also discussed in that section. 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 computation block variants have been suggested. These *block Arnoldi variants* are discussed in §5.6.

Lanczos methods, §5.7, §5.8, §5.9 and §5.10.

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 (see Chapter **??**) 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, has been described in §5.7, as well as appropriate software.

A *block version* of the Lanczos method, and an implementation of it called *ABLE*, is presented in §5.8. 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 biortogonality, and clusters of nearby eigenvalues). Deflation within Krylov subspaces is nicely incorporated in yet another variant, the *band Lanczos method*, presented in §5.9. This is in particular very convenient in reduced order modelling, for instance of linear dynamical models.

A variant of the two-sided Lanczos method for *complex symmetric* systems is discussed in §5.10. With respect to the unsymmetric case, this variant reduces CPU-time and computer storage with a factor two.

Jacobi-Davidson method, §5.11.

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 $x = (A - \sigma I)^{-1}$ 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.

We still need to state a few sentences about the selection of an algorithm, and refer to the discussion in §2.6

5.2 Transformation Methods

The main direct transformation method[1] used in practice is the QR algorithm. It computes the Schur cannonical form (or simply call Schur decomposition) of a non-Hermitian matrix A:

$$A = QTQ^*. \tag{5.2}$$

where Q is a unitary matrix, $Q^*Q = I$, and T is an upper tridiagonal matrix. The eigenvalues of A are the diagonal entries of T (see also Chapter 2). When A is real, the QR algorithm computes the real Schur decomposition to avoid complex numbers for saving in flops and memory.

The QR algorithm is orignated from the simple QR iteration:

Let $A_0 = A$,
For $k = 1, 2, \ldots,$
    QR-Factorize $A_{k-1} = Q_k R_k$
    Compute $A_k = R_k Q_k$
End for

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 have been developed, including Hessenberg reduction, implicitly shifting, deflation, and matrix balacing. We refer the interested reader to [101, 48] and references therein for the details of the related theory and implementations.

The QR algorithm costs $O(n^3)$ flops 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 $25n^3$ flops if both Q and T are computed. If only eigenvalues are desired, then about $10n^3$ flops are necessary.

The QR algorithm is backward stable[2], i.e., for the computed unitary matrix $\widehat{Q}$ (within machine precision) and the computed upper triangular matrix $\widehat{T}$, we have

$$A + E = \widehat{Q}\widehat{T}\widehat{Q}^*,$$

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. In LAPACK [2], the following driver routines are available for performing a variant of tasks for computing Schur decompositions, eigenvalues, eigenvectors, and estimates for the accruacy of computed results:

[1] Note that "direct transformation 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

[2] for the definition of backward stability, see Chapter 9.

`xGEES`	compute Schur decomposition with eigenvalue ordering,
`xGEESX`	`xGEES` plus condition estimates,
`xGEEV`	Eigenvalues and eigenvectors,
`xGEEVX`	`xGEEVX` 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.

Parallelism? (ScaLAPACK)

5.3 Single and Multiple Vector Iterations

M. Gu

The single and multiple vector iteration methods for the Hermitian eigenproblem in §3.3 can also be used to solve the non-Hermitian eigenproblem, 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. One exception seems to be the Rayleigh Quotient Iteration, which is expensive and converges much more slowly than its Hermitian counterpart.

5.3.1 Power Method

The power method, described in Algorithm ??, can be used to solve the non-Hermitian eigenproblem 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 λ_2 is the second largest eigenvalue of A in magnitude. For detailed discussions on the power method, see Wilkinson [255], Golub and Van Loan [101] and Demmel [48].

5.3.2 Inverse Iteration

Inverse iteration, described in Algorithm ??, can also be used to solve the non-Hermitian eigenproblem without any apparent change. As in the Hermitian case, assume that λ_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. 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 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.

In general, inverse iteration tends to have much more rapid convergence than the power method if σ 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.

5.3.3 Subspace Iteration with Projection

The subspace iteration methods can also be generalized for the non-Hermitian eigenproblem. In particular, we can generalize Algorithm ?? of §?? to get

ALGORITHM 5.20: **Subspace Iteration with Projection**

(1) *QR-factorize* $Z\,R := V$
for $k = 2, 3, ...$
(2) $Y := A^{iter}\,Z$
(3) *QR-factorize* $Z\,R := Y$.
(4) $\Theta := Z^*\,A\,Z$ *and the Schur Decomposition* $\Theta = Q\,\Lambda\,Q^*$
(5) $Z := Z\,Q$
(6) *if* $||AZ - Z\,\Lambda||_2 \leq \epsilon$, *stop*
(7) *choose a new iteration parameter* $iter$.

In §5.3.4, we will discuss more implementation details about this procedure to make it more practical. In Algorithm **??**, the iteration parameter *iter* is introduced to reduce the costly orthonormalization computation in step (3) as much as possible. However, *iter* must not be too large lest one lose numerical accuracy in the computation of the matrix Y in step (2), leading to inaccurate computation of some of the eigenvalues. **what's the typical value of iter?**

Eigendecomposition can also be used instead of Schur decomposition in Algorithm **??**, with some potential loss of numerical accuracy in the computed eigenvalues. The resulting algorithm will converge under conditions similar to its Hermitian counterpart in step (4). However, these two versions are equivalent when A is Hermitian.

5.3.4 Practical Implementations

As in the Hermitian case, there are a number of implementation details that enhance the performance of the non-Hermitian subspace iteration methods. The following algorithm is a modification of the subspace iteration method of §3.3.5 for the Hermitian eigenproblem. It includes deflation (locking) for computing the *nev* dominant eigenvalues. Since eigenvalues for non-Hermitian eigenproblems can be very ill-conditioned, we impose a more stringent convergence criterion.

ALGORITHM 5.21: **Subspace Iteration with Projection and Deflation**

(1) *QR-factorize* $Z\,R := V$ *for the starting vectors* V
(2) **while** $j \leq nev$ **do**
(3) *Compute* $\widehat{Y} := \left[q_1, q_2, \ldots, q_{j-1}, A^{iter}\,Z\right]$.
(4) *Orthonormalize the columns of* $\widehat{Y}$ *(starting at* j*) into* Z.
(5) $\Theta := Z^*\,A\,Z$ *and compute the Schur decomposition*
form $\Theta = Q\,\Lambda\,Q^*$ *with nearly equimodular eigenvalues*
grouped in diagonal blocks in Λ.
(6) **for** *diagonal blocks* Λ_k, $k = 1, 2, \ldots$ *in* Λ **do**
(7) *Let* Q_k *be the corresponding columns in* Q
(8) **if** $\|A\,(Z\,Q_k) - (Z\,Q_k)\,\Lambda_k\| \leq \epsilon$ **then**
(9) *append* $Z\,Q_k$ *to* $[q_1, q_2, \ldots, q_{j-1}]$
(10) *set* $j = j +$ *number of columns in* $Z\,Q_k$
(11) **else**
(12) *break*
(13) **end if**
(14) *Choose a new iteration parameter* $iter$.

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.

(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 Λ. In step (12), convergence testing is stopped as soon as the first block of eigenvalues in Λ fail to converge.

(14) The iteration parameter $iter$ should be chosen to minimize orthonormalization cost while maintaining a reasonable amount of numerical accuracy.

If eigenvalues near a shift σ 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 σ 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 ρ is an estimate of the spectral radius of A.

Much of the material in this section is based on Bai and Stewart [16] and Saad [212]. For further discussion on subspace iteration, The reader is refered to Wilkinson [255] and Stewart [238].

5.3.5 Software Availability

There are several pieces of software available for subspace iteration. EB12 is a routine by Duff and Scott for subspace iteration [60]. 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 Z. Bai and G. W. Stewart to calculate a few eigenvalues with largest modulus for a real nonsymmetric matrix [16]. It also computes an approximate orthonormal basis for the associated invariant subspace. SRRIT is non-proprietary. The template in §5.3.4 is closely related to SRRIT.

LOPSI is a package by W. J. Stewart and A. 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 [241]. 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.

5.4 Arnoldi's Method

Y. Saad

Editor's comments: Note that there were quite few typos and references to non-existence places and variables. Although we have tried our best to correct those errors, Yousef, please check, and double check it carefully. Thanks!

Arnoldi's method was introduced in 1951 as a *direct* algorithm for reducing a general matrix into upper Hessenberg form [7]. It was later discovered that this algorithm leads to a good *iterative* technique for approximating eigenvalues of large sparse matrices.

The algorithm works for nonsymmetric (real) or non-Hermitian (complex) 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.

5.4.1 The Basic Algorithm

Arnoldi's method is an orthogonal projection method onto a Krylov subspace. It starts with the Arnoldi procedure as described in Algorithm 5.22. The procedure can be essentially viewed as a modified Gram-Schmidt process for building an orthogonal basis of the Krylov subspace $\mathcal{K}_m$.

ALGORITHM 5.22: **Arnoldi Procedure**

(1) $v_1 = v/\|v\|_2$ *for the starting vector* v
(2) *For* $j = 1, 2, \ldots, m$ *do*
(3) $\quad w := Av_j$
(4) $\quad$ *For* $i = 1, 2, \ldots, j$ *do*
(5) $\quad\quad h_{ij} = w^* v_i$
(6) $\quad\quad w := w - h_{ij} v_i$
(7) $\quad$ *End for*
(8) $\quad h_{j+1,j} = \|w\|_2$
(9) $\quad$ *If* $h_{j+1,j} = 0$ *Stop*
(10) $\quad v_{j+1} = w/h_{j+1,j}$
(11) *End for*

The algorithm will stop if the vector w computed in line (8) vanishes. The vectors $v_1, v_2, \ldots, v_m$ form an orthonormal system by construction. 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

$$Av_j = \sum_{i=1}^{j+1} h_{ij} v_i, \quad j = 1, 2, \ldots, m \ . \tag{5.3}$$

If we denote by V_m the $n \times m$ matrix with column vectors $v_1, \ldots, v_m$, 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:

$$AV_m = V_m H_m + h_{m+1,m} v_{m+1} e_m^*, \tag{5.4}$$

$$V_m^* A V_m = H_m \ . \tag{5.5}$$

Relation (5.4) follows from (5.5) by multiplying both sides of (5.4) by V_m^* and making use of the orthonormality of $\{v_1, \ldots, v_m\}$.

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 [212].

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 λ_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 process 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 $u_i^{(m)}$ 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,

$$\|(A - \lambda_i^{(m)} I) u_i^{(m)}\|_2 = h_{m+1,m} |e_m^* y_i^{(m)}| \ .$$

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.

5.4.2 Orthogonality

The description of the Arnoldi process given earlier was based on the modified Gram-Schmidt process. Other orthogonalization algorithms could be used as well. One possible improvement to the classical Gram-Schmidt process is to re-orthogonalize 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 cancellation 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 [192]).

One of the most reliable orthogonalization techniques, from the numerical point of view, is the Householder algorithm [101]. This has been implemented for the Arnoldi process by Walker [253]. 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 of 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. The alternative of the Modified Gram-Schmidt with a good reorthogonalization strategy based on a measure of the level of cancellation, is more economical and seems to be adequate for most cases, see for example, Algorithm 3.16.

5.4.3 Explicit Restarts

As was mentioned earlier, the standard implementations of Arnoldi's 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 Arnoldi's 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 §5.5.

ALGORITHM 5.23: **Explicitly Restarted Arnoldi Algorithm**

(1) **Iterate:** *Perform m steps of Algorithm 5.22*
(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)*

5.4.4 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

$$\widehat{v}_1 = \sum_{i=1}^{p} \rho_i \widetilde{u}_i,$$

where the eigenvalues are numbered in decreasing order of their real parts. The vector v_1 is then obtained from normalizing $\widehat{v}_1$. The simplest choice for the coefficients ρ_i is to take $\rho_i = 1, i = 1, \ldots, p$. There are several drawbacks to this approach, the most important of which being that there is no easy way of choosing the coefficients ρ_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 $U_{k-1} = [u_1, \ldots, u_{k-1}]$ of the invariant subspace has already been computed, then, to compute the eigenvalue λ_k, we can work with the matrix

$$\widetilde{A} = A - U_{k-1} \Sigma U_{k-1}^*,$$

in which $\Sigma = \text{diag}\,(\sigma_i)$ is a diagonal matrix of shifts. The eigenvalues of $\widetilde{A}$ consist of two groups. Those eigenvalues associated with the Schur vectors $u_1, \ldots, u_{k-1}$ will be shifted to $\widetilde{\lambda}_i = \lambda_i - \sigma_i$ and the others remain unchanged. If the eigenvalues with largest real parts are sought, then the shifts are selected so that λ_k becomes the next eigenvalue with largest real part of $\widetilde{A}$. 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:

$$\widetilde{A} = A(I - U_{k-1} U_{k-1}^*)$$

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 $\widetilde{A} = A - U_{k-1} R_{k-1} U_{k-1}^*$. Those eigenvalues associated with the Schur vectors $u_1, \ldots, u_{k-1}$ will now all be moved to zero.

A better implementation of deflation, which fits in well with the Arnoldi process is to work with a single basis $v_1, v_2, ..., v_m$ whose first vectors are the Schur vectors that have already converged. Suppose that $k-1$ such vectors have converged and call them $v_1, v_2, ..., v_{k-1}$. Then we start by choosing a vector v_k which is orthogonal to $v_1, ..., v_{k-1}$ and of norm 1. Next we perform $m-k$ steps of an Arnoldi process in which orthogonality of the vector v_j against all previous $v_i's$, including $v_1, ..., v_{k-1}$ is enforced. This generates an orthogonal basis of the subspace

$$\text{Span } \{v_1, \ldots, v_{k-1}, v_k, Av_k, \ldots, A^{m-k}v_k\} \ . \tag{5.6}$$

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 Arnoldi's method is the following.

ALGORITHM 5.24: **Explicitly Restarted Arnoldi with Deflation**

(1) *Set* $k := 1$.
(2) **Loop:**
(3) *Arnoldi Iteration. For* $j = k, k+1, ..., m$ *do:*
$w := Av_j$.
Compute a set of j *coefficients* h_{ij} *so that*
$w := w - \sum_{i=1}^{j} h_{ij}v_i$ *is orthogonal to*
to all previous v_i*'s,* $i = 1, 2, ..., j$.
$h_{j+1,j} = \|w\|_2$
$v_{j+1} = w/h_{j+1,j}$.
(3) *Compute approximate eigenvector of* A *associated with the eigenvalue* $\widetilde{\lambda}_k$ *and its associated residual norm estimate* ρ_k.
(4) *Orthonormalize this eigenvector against all previous* v_j*'s to get the approximate Schur vector* $\widetilde{u}_k$ *and define* $v_k := \widetilde{u}_k$.
(5) *If* ρ_k *is small enough then (accept eigenvalue):*
$h_{i,k} = v^* Av_k$, $i = 1, .., k$,
Set $k := k + 1$,
If $k \geq nev$ *then stop else goto (2)*
(6) *Else*
go to (3)
End if

Note that in the Loop, the Schur vectors associated with the eigenvalues $\lambda_1, ..., \lambda_{k-1}$ 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.

$$\underbrace{[v_1, v_2, \ldots, v_{k-1}}_{Locked}, \underbrace{v_k, v_{k+1}, \ldots v_m]}_{Active}$$

When a new Schur vector converges, step (5) 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

$$H_m = \begin{pmatrix} * & * & * & * & * & * \\ & * & * & * & * & * \\ & & * & * & * & * \\ & & * & * & * & * \\ & & & * & * & * \\ & & & & * & * \end{pmatrix} \tag{5.7}$$

In the subsequent steps, only the eigenvalues not associated with the 2×2 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 (2×2) block is the same matrix that would be obtained from an Arnoldi run applied to the matrix

$$\widetilde{A} = (I - U_{k-1}U_{k-1}^*)A$$

Thus, we are implicitly projecting out the invariant subspace already computed from the range of A.

5.5 The Implicitly Restarted Arnoldi Method

R. Lehoucq and D. Sorensen

5.5.1 Introduction

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.

$$AV = VR, \tag{5.8}$$

This well known decomposition asserts that every square matrix A is *unitarily similar* to an upper triangular matrix R.

The QR algorithm produces a sequence of unitary similarity transformations that iteratively reduce A to upper triangular form. 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 $AV = VH$ where H is upper Hessenberg (Almost upper triangular) and V is unitary. Then the iteration shown below

SHIFTED QR METHOD
Factor $AV = VH$
for $j = 1, 2, 3..$, until convergence
select a shift μ
QR-Factorize $QR = H - \mu I$;
$H \leftarrow Q^*HQ$; $V \leftarrow VQ$;
end

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 V_k represents the leading k columns of V and R_k the leading principal $k \times k$ submatrix of R in (5.8), then

$$AV_k = V_k R_k$$

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 R_k consisting of any specified subset of k eigenvalues of A. Moreover, $Range\{V_k\}$ 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 R 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 this method the implicitly restarted Arnoldi method.

5.5.2 The Arnoldi Process in GEMV Form

To begin our discussion, we shall look at the Arnoldi process from a slightly different perspective. The Arnoldi process was derived previously in §5.4 and has been presented in Modified Gram-Schmidt algorithmic form, Algorithm 5.22. In this section, we describe it in matrix-vector (or GEMV) form using the classical Gram-Schmidt process together with the orthogonalization refinement [38]. 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 $V_k \in \mathbf{C}^{n \times k}$ has orthonormal columns, $V_k^* f_k = 0$ and $H_k \in \mathbf{C}^{k \times k}$ is upper Hessenberg with non-negative 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* or *Lanczos vectors* , respectively.

A template for computing a k-step Arnoldi factorization are given in Algorithm 5.25.

ALGORITHM 5.25: **k-step Arnoldi Factorization**

(1) $v_1 = v/\|v\|$ *of the starting vector* v,
(2) $w = Av_1$;
$h = v_1^* w$; $f_1 \leftarrow w - v_1 h$;
$V_1 \leftarrow [v_1]$; $H_1 = [h]$;
for $j = 1, 2, 3, \ldots k\text{-}1$,
$\beta_j = \|f_j\|$;
(3) $v_{j+1} \leftarrow f_j/\beta_j$;
$V_{j+1} \leftarrow [V_j, v_{j+1}]$; $\widehat{H}_j \leftarrow \begin{bmatrix} H_j \\ \beta_j e_j^* \end{bmatrix}$;
$w \leftarrow Av_{j+1}$;
(4) $h \leftarrow V_{j+1}^* w$;
$f_{j+1} \leftarrow w - V_{j+1} h$;
(5) *if* $(\|f_{j+1}\| < \eta \|h\|)$,
$s \leftarrow V_{j+1}^* f_{j+1}$;
$f_{j+1} \leftarrow f_{j+1} - V_{j+1} s$;
$h \leftarrow h + s$;
end_if
(6) $H_{j+1} \leftarrow [\widehat{H}_j, h]$;
end

We will now describe some implementation details, referring to the respective phases in Algorithm 5.25.

(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 process. In the subsequent step, f_1 will become the new Arnoldi basis vector. This step should include a re-orthogonalization just as all of the others do.

(3) 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 re-computed here. It has already been computed at Step (5) and can be saved for re-use here.

(4) 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.

(5) 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 η 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 θ is the angle that the vector w makes with $Range(V_{j+1})$, (i.e., the angle θ between w and its projection $V_{j+1}h$).

A larger value of η is a more stringent orthogonality requirement. With $\eta = 0$ the algorithm reverts to classical Gram-Schmidt. 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 $w \in Range(V_{j+1})$. In this case, the Arnoldi process 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 β_{j+1} should be set to zero to continue the factorization. The value of η used in the ARPACK implentation (see §5.5.12 is $1/\sqrt{2}$. Additional discussion of this orthogonalization device is given below.

(6) 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 super-computer. Moreover, considerable effort has been made within the ScaLapack project [?] 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 classical Gram Schmidt process (CGS). It is notoriously unstable and will fail miserably in this setting without modification. One remedy is to use the Modified Gram Schmidt process (MGS) as used in Algorithm 5.22. 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 [38]. 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/Arnoldi process. In the Hermitian case, Paige [185] 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 re-orthogonalization have been proposed. These are discussed in §3.4.

5.5.3 The Restarting Idea

An unfortunate aspect of the Lanczos/Arnoldi 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. The eigen-information 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, eigen-information 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 $\mathcal{O}(k^3)$ flops.

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 non-trivial 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 [131] soon after the appearance of the original algorithm of Lanczos [152]. Subsequently, there were developments by Paige [185], Cullum and Donath [34], and Golub and Underwood [100]. More recently, a restarting scheme for eigenvalue computation was proposed by Saad based upon the polynomial acceleration scheme originally introduced by Manteuffel [166] 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.

5.5.4 Implicit Restarting

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 processes 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 $2nk + \mathcal{O}(k^2)$storage. The computed Schur basis vectors for the desired k-dimensional eigen-space 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 $m = k + p$

$$AV_m = V_m H_m + f_m e_m^*, \tag{5.9}$$

is compressed to a factorization of length k that retains the eigen-information of interest. This is accomplished using QR steps to apply p shifts implicitly. The first stage

of this shift process results in

$$AV_m^+ = V_m^+ H_m^+ + f_m e_m^* Q, \tag{5.10}$$

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 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 (5.10) remain in an Arnoldi relation. Equating the first k columns on both sides of (5.10) provides an updated $k-$step Arnoldi factorization

$$AV_k^+ = V_k^+ H_k^+ + f_k^+ e_k^*, \tag{5.11}$$

with an updated residual of the form $f_k^+ = V_m^+ e_{k+1} \widehat{\beta}_k + f_m \sigma$. Using this as a starting point it is possible to apply p additional steps of the Arnoldi process to return to the original m-step form.

A template for computing a k-step Arnoldi factorization with implicit restart is given in Algorithm 5.26

ALGORITHM 5.26: **Implicitly Restarted Arnoldi Method (IRAM)**

(1) *Starting with* $v_1 = v/\|v\|$;
Compute an $m = k + p$ *step Arnoldi factorization*
$AV_m = V_m H_m + f_m e_m^*$.
(2) *repeat until "convergence",*
(3) *Compute* $\sigma(H_m)$ *and select* p *shifts* $\mu_1, \mu_2, \ldots \mu_p$
$Q = I_m$;
(4) *for* $j = 1, 2, \ldots, p,$
Factor $[Q_j, R_j] = qr(H_m - \mu_j I)$;
$H_m \leftarrow Q_j^* H_m Q_j$;
$Q \leftarrow Q Q_j$;
end
$\beta_k = H_m(k+1, k)$; $\sigma_k = Q(m, k)$;
(5) $f_k \leftarrow v_{k+1}\beta_k + f_m \sigma_k$;
$V_k \leftarrow V_m Q(:, 1:k)$; $H_k \leftarrow H_m(1:k, 1:k)$;
(6) *Beginning with the k-step Arnoldi factorization*
$AV_k = V_k H_k + f_k e_k^*$,
apply p *additional steps of the Arnoldi process*
to obtain a new m-step Arnoldi factorization
$AV_m = V_m H_m + f_m e_m^*$.
end

We will now describe some implementation details, referring to the respective phases in Algorithm 5.26.

(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.

(2) A Ritz pair (x, θ) is "converged" if $\|f_k\||e_k^* y| < \|H_k\|\epsilon_D$, where $x = V_k y$ and θ is "wanted". Upon convergence, this pair should be deflated. See the discussion of deflation in §5.5.7.

(3) Shift Selection: 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 H_m. A successful strategy has been to sort the eigenvalues of H_m into a "wanted set" $\theta_1, \theta_2, \ldots, \theta_k$ and an "unwanted set" $\mu_1, \mu_2, \ldots, \mu_p$ 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.

(4) Apply p steps of the shifted QR iteration to H_m using the $\mu_1, \mu_2, \ldots, \mu_p$ 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 $\theta_1, \theta_2, \ldots, \theta_k$ as it's eigenvalues.

(5) When exact shifts are used, then the properties mentioned in item(4) 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.

(6) This step requires a slight modification of the Arnoldi factorization discussed previously. It simply begins with step k of the usual scheme.

5.5.4.1 Shift Selection

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 [209], Harmonic Ritz values [175, 180, 187, 227], the roots of Leja polynomials [9], the roots of

least squares polynomials [210] and refined shifts [121]. In particular, the Leja and Harmonic Ritz values have been used to estimate the interior eigenvalues of A.

5.5.4.2 Convergence Behavior

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.

$$v_1 \leftarrow \psi(A)v_1 \quad \text{with} \quad \psi(\lambda) = \prod_{j=1}^{p}(\lambda - \mu_j). \tag{5.12}$$

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 [234].

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 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 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 μ_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 [234] for information on the convergence of IRAM and [8, 237] for other possible shift strategies for Hermitian A. The reader is referred to [157, 177] for studies comparing implicit restarting with other schemes.

5.5.5 Numerical Stability of IRAM

Robust implementations of a practical QR algorithm, such as those found in the software packages EISPACK [231] and LAPACK [2], compute an approximate Schur form for a matrix A that satisfies

$$(A + E)\widehat{V} = \widehat{V}\widehat{R}, \tag{5.13}$$

with $\widehat{R}$ is upper triangular, $\|\widehat{V}^*\widehat{V} - I\| \approx \epsilon_M$, and $\|E\| \approx \epsilon_M\|A\|$, where ϵ_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 §(??) 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 process, if extended to n steps, is simply another way to obtain this reduction. The introduction of the orthogonal correction [38] allows the Arnoldi process 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

$$(A + E_o)V_k = V_kH_k + f_ke_k^*,$$

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 $E = E_o + f_k(V_ke_k)^*$. If $H_kZ_k = Z_kR_k$ is a Schur decomposition of H_k then

$$(A + E)\widehat{V}_k = \widehat{V}_kR_k \quad \text{with} \quad \widehat{V}_k = V_kZ_k$$

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

$$(A + E)x = x\theta, \tag{5.14}$$

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, θ) are exact for a nearby matrix $A + E$.

Through the use of the deflation techniques that we shall present here in §5.5.7, 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

$$(A + \widehat{E})\widehat{V}_k = \widehat{V}_k R_k, \quad \text{with} \quad \widehat{V}_k = V_m Q,$$

where $\widehat{E}$ is small relative to $\|A\|$. A discussion of this is given in Sec 5.5.7 and more detail is available in [235].

5.5.6 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 γn. For a sparse matrix, one could think of γ as twice the average number of non-zero entries per row of A. For a dense matrix $\gamma = 2n$, and for an FFT matrix $\gamma \approx \log(n)$. For a fixed value of k and p, the cost (in flops) for one iteration of IRAM is

1. Matrix-vector products $w \leftarrow Av$: $\gamma p n$.
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: $2n(k^2 + kp) + \mathcal{O}((k + p)^3)$.
5. Total Cost: $\gamma p n + 2[(5k - 2)p + 2p^2]n + 2k^2 n + \mathcal{O}((k + p)^3)$.

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. γ 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.

5.5.7 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 ϵ_D with $1 > \epsilon_D > \epsilon_M$ where ϵ_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 round-off 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 process, 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 θ 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, 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 then we wish to decouple and remove the eigenpair (x, θ) from the current Krylov subspaced spanned by the Arnoldi vectors. This process is called purging.

Techniques for this deflation were first developed in [154, 159]. The scheme we present here is based an improvement developed in [235]. 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 shift-invert spectral transformation 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 [154], the forward instability of the QR bulge-chase process discovered by Parlett and Le [195] will prevent implicit restarting to be used for purging converged unwanted Ritz values.

5.5.8 An 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, the algorithm shown in Figure 5.27 computes an orthogonal matrix Q such that $Qe_1 = y$ (hence $y = Q^* e_1$). This orthogonal matrix has a very special form. The orthogonal matrix Q constructed as prescribed in Algorithm 5.27 may be written as

$$Q = R + ye_1^*, \quad \text{with} \quad Re_1 = 0, \; R^* y = 0, \tag{5.15}$$

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

$$Q = L + yg^*, \quad \text{with} \quad Le_1 = 0, \; L^* y = e_1 - g, \tag{5.16}$$

where L is lower triangular, and $g^* \equiv e_1^* + \frac{1}{\eta_1} e_1^* R$.

Now, consider the matrix $Q^* H Q$. The substitutions $Q^* = (L + yg^*)^*$, $Q = (R + ye_1^*)$ from Equations 5.16 and 5.15 will give

$$\begin{aligned} Q^* H Q &= Q^* H (R + ye_1^*) \\ &= (L^* + gy^*) HR + Q^* H y e_1^* \\ &= L^* HR + gy^* HR + Q^* H y e_1^*. \end{aligned}$$

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 $H_+ \equiv Q^* H Q$ is of the form

$$H_+ = \begin{bmatrix} \theta & h^* \\ 0 & H_2 \end{bmatrix}. \tag{5.17}$$

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

$$H_+ = \begin{bmatrix} \theta & 0 \\ h & H_2 \end{bmatrix}, \tag{5.18}$$

where H_2 is upper Hessenberg.

We shall use (5.17) for locking a converged Ritz value, and we shall use (5.18) to purge an unwanted but converged Ritz value.

It should be noted that, as computed by the algorithm shown in 5.27, Q will have componentwise relative errors on the order of machine precision ϵ_M with no element growth. Moreover, extension to complex arithmetic is completely straightforward (unlike Givens or Householder transformations).

ALGORITHM 5.27: **Orthogonal Deflating Transformation**

function $[Q] = \mathbf{orthQ}(y)$;
Input: *y of dimension k with* $\|y\| = 1$ *and* $\eta = e_k^* y$.
Output: Q *such that* $Qe_1 = y$, $e_k^* Q = (\eta, \tau e_{k-1}^*)$, *with* $Q^*Q = I$.

```
Q = 0;  Q(:,1) = y;
σ = y(1)^2;  τ_o = |y(1)|;
for j = 2 : n,
    σ ← σ + y(j)^2;
    τ = √σ;
    if ( τ_o ≠ 0),
        γ = (y(j)/τ)/τ_o;
        Q(1 : j-1, j) = -y(1 : j-1)γ;
        Q(j, j) = τ_o/τ;
    else
        Q(j-1, j) = 1;
    end_if
    τ_o = τ;
end_for
```

5.5.9 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 (blocksize = 2) of the single vector case we are about to present. The purging in this case will be a direct analogue. Unfortunately, there maybe numerical complications

with locking a complex conjugate pair. A complete discussion would involve considerable detail and analysis. We feel these details are beyond the scope of a template presentation. They may be found in [235].

Locking θ: 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 ϵ_M and 1.

If θ is "wanted" then it is desirable to lock θ. 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 θ. This may be accomplish by constructing a $k \times k$ orthogonal matrix $Q = Q(y)$ using Algorithm 5.27

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

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 $\widehat{H}_2 = U^* H_2 U$ 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 5.27 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 $Q \leftarrow Q \begin{bmatrix} 1 & 0 \\ 0 & U \end{bmatrix}$, and the end result of these transformations will be

$$\begin{aligned} Av_1 &= v_1\theta + f\eta, \quad \text{where} \quad v_1^* f = 0 \\ AV_2 &= [v_1, V_2] \begin{bmatrix} h^* \\ H_2 \end{bmatrix} + f\tau e_{k-1}^*. \end{aligned}$$

where $[v_1, V_2] = VQ$ and H_2 is upper Hessenberg. After this, all subsequent implicit restarting takes place as if

$$AV_2 = V_2 H_2 + f\tau e_{k-1}^*.$$

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 [196, 192] is accomplished automatically.

Purging θ: If θ is "unwanted" then we may wish to remove θ 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 [159].

We shall use (5.18) 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 η be small. The desired puging amounts to simply discarding the first column on both sides of this equation. We are then left with

$$AV_2 = V_2H_2 + f\tau e_{k-1}^*.$$

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 $y^*H - \theta y^*$ is small.

5.5.10 Stability of Q^*HQ

The deflating orthogonal transformation construction shown in 5.27 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

$$Q^*HQ = L^*HR + gy^*HR + \theta e_1e_1^*.$$

However, on closer examination, we see that

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

where η_1 is the first component of y. Therefore,

$$\|g\| = \frac{1}{|\eta_1|}, \tag{5.19}$$

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 ϵ_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 $\mathcal{O}(1)$ if $\eta_1 = \mathcal{O}(\epsilon_M)$. 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

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

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

$$y^*Hq_j = 0, \quad \text{for} \quad j > 1$$

numerically. As shown in [235], 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 adjusted to be small enough by scaling the vector y_j with a number ϕ and the component η_{j+1} with a number ψ. With this re-scaling just prior to the computation of q_{j+1}, we then have $y_j \leftarrow y_j\phi$ and $\widehat{y}_j \leftarrow \widehat{y}_j\psi$ where $y^* = [y_j^*, \widehat{y}_j^*]$. Certainly, $\|y\|$ should not be altered with this scaling and this is therefore required. This gives the following system of equations to determine ϕ and ψ: If $|y_j^*H_j\widehat{q}_j + \rho_j\beta_j\eta_{j+1}| > \epsilon_M\tau_{j+1}$,

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

ALGORITHM 5.28: **Stable Orthogonal Deflating Transformation**

function $[Q] = \mathbf{orthQ}(y, H)$;
Input: *y of dimension m with* $\|y\| = 1$ *and* $\eta = e_m^* y$.
H upper Hessenberg of order m with $y^* H = \theta y*$.
Output: *Q such that* $Qe_1 = y$, $e_m^* Q = (\eta, \tau e_{m-1}^*)$, *with* $Q^*Q = I$.
y may be modified. Q^*HQ *is Hessenberg.*

```
        Q = 0;
(1)     while(y(i) = 0), y(i) = ε_M/n;  i = i + 1;  end_while
        σ = y(1)^2;  τ_o = |y(1)|;

        for j = 2 : n,
           σ = σ + y(j)^2;  τ = √σ;
           γ = −(y(j)/τ)/τ_o;
           ρ = τ_o/τ;
           Q(1 : j − 1, j) = y(1 : j − 1)γ;  Q(j, j) = ρ;

(2)        if (j < n  &  τ < .05),
              τ_p = √(σ + y(j + 1)^2);
              α = y(1 : j)* H(1 : j, 1 : j)Q(1 : j, j);
              δ = y(j + 1)H(j, j + 1)ρ;
              σ_o = −1; if( sign(α) · sign(δ) < 0), σ_o = 1; end_if;

(3)           if(|δ + α| > ε_M τ_p),
                 if (ρ < √ε_M/100),
                    Q(j, j) = Q(j, j)σ_o(ε_M τ_p + |α|)/|δ|;
                 else
                    if(|α| > |δ|),
                       φ = σ_o(ε_M τ_p + |δ|)/|α|;
                       y(1 : j) = y(1 : j)φ;
                       σ = σφ^2;  τ = τ|φ|;
                    else
                       ψ = σ_o(ε_M τ_p + |α|)/|δ|;
                       y(j + 1 : n) = y(j + 1 : n)ψ;
                    end_if
                 end_if
              end_if
           end_if
           τ_0 = τ;
        end_for
        Q(:, 1) = y/‖y‖;
```

If ρ_j is on the order of $\sqrt{\epsilon_M}$, then the scaling may be absorbed into ρ_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 [235].

The code shown in Figure 5.28 implements this scheme to compute an acceptable

Q. In practice, this transformation may be computed and applied in place to obtain $H \leftarrow Q^*HQ$ and $V \leftarrow VQ$ 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 $2-n$ have been determined.

There are several implementation issues.

(1) 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.

(2) 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 5.27.

(3) 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 ρ instead of y is designed so that $f\ell(1 + (\rho\psi)^2) = f\ell(1 + \rho^2) = 1$.

5.5.11 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 [235], 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 displying code. Instead, we shall indicate the main ideas of the strategy with a fairly high level verbal description.

ALGORITHM 5.29: **Deflation for IRAM**

(1) Lock a single Ritz value θ 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 §5.5.4.2). 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 §5.5.4.2 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).

5.5.11.1 Eigenvector Computation with a Spectral Transformation

The shift-invert spectral transformation was introduced in §2.8. Specifically, we use the matrix

$$C \equiv (A - \sigma I)^{-1}$$

in place of A and rely on the fact that

$$Cx = x\theta \iff Ax = x(\sigma + \frac{1}{\theta})$$

to recover eigenvalues of the original matrix. The Arnoldi method is extremely effective when used in conjunction with shift-invert spectral transformation. Convergence to eigenvectors corresponding to the eigenvalues of A that are nearest to σ 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 criterion 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 post processing. In order to compute eigenvalues of A near to σ we will compute the eigenvalues θ of C that are of largest magnitude. It is not really necessary for σ 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 σ 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

$$(A - \sigma I)^{-1} x - x\theta = CV_m y - V_m H_m y = f_m \, e_m^* y, \tag{5.20}$$

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 ϵ_U is a user-specified tolerance, then according to 5.14, x and θ 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 (5.20) gives

$$Ax - x(\sigma + 1/\theta) = -(A - \sigma I) f_m \frac{e_m^* y}{\theta} \tag{5.21}$$

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 (5.21) and rearranging terms, will result in

$$Az - z(\sigma + 1/\theta) = -f_m \frac{e_m^* y}{\theta^2}, \tag{5.22}$$

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 (5.20) it follows that

$$(A - \sigma I)^{-1} x = x\theta + f_m e_m^* y \tag{5.23}$$

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, $\|z\|^2 = 1 + \left(\frac{|e_m^* y| \|f_m\|}{|\theta|}\right)^2$, so the error bound will decrease after z is normalized. Moreover, if $\frac{|e_m^* y| \|f_m\|}{|\theta|} < \sqrt{\epsilon_M}$, then the floating point computation of the norm will already result in $\|z\| = 1$ without any rescaling.

¿From (5.20), 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 (5.22)

$$|\frac{z^* A z}{z^* z} - (\sigma + 1/\theta)| = \frac{1}{|\theta|} \left(\frac{\|f_m\| |e_m^* y|}{|\theta| \|z\|}\right)^2 \tag{5.24}$$

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 round-off level of being a Rayleigh quotient when $\frac{|e_m^* y| \|f_m\|}{|\theta|} < \sqrt{\epsilon_M}$.

On the other hand, from (5.21) we deduce that

$$|x^* A x - (\sigma + 1/\theta)| = \frac{|x^* A f_m|}{\|f_m\|} \left(\frac{\|f_m\| |e_m^* y|}{|\theta|}\right) \tag{5.25}$$

is the error in using $\sigma + 1/\theta$ as a Rayleigh quotient for A when x is used.

Equations (5.22) and (5.24) 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 one. 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 σ 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 [75] for the symmetric positive-definite generalized eigenvalue problem.

5.5.12 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 Fortran77 subroutines designed to solve large scale eigenvalue problems. ARPACK stands for ARnoldi PACKage. 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 Implicitly Restarted Arnoldi Method (IRAM) that is presented here in §5.5.4. When the matrix A is symmetric (Hermitian)

this reduces to the the Implicitly Restarted Arnoldi Method (IRAM) just described in this section.

It provides various drivers in template form for shift-invert spectral transformations. There are special drivers for banded matrices. It also provides a driver for computing a partial singular value decomposition (SVD) of a general rectangular matrix. For complete information, one should consult the ARPACK Users' Guide [160].

5.6 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 computational 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 §??) 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 §5.3. 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.

5.6.1 Block Arnoldi Reductions

Let A be a matrix of order n and $b > 0$ be the block size. We say that

$$AV_{[m]} = V_{[m]} H_{[m]} + F_m E_m^* \tag{5.26}$$

is a block Arnoldi reduction of length m when $V_{[m]}^* A V_{[m]} = H_{[m]}$ 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 sub-diagonals. The columns of $V_{[m]} = [V_1, V_2, \ldots, V_m]$ are an orthogonal basis for the block Krylov subspace

$$\mathcal{K}_m(A, V_1) \equiv \{V_1, AV_1, \cdots, A^{m-1} V_1\}.$$

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 [101, 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 5.30 lists an algorithm to compute a block Arnoldi reduction.

ALGORITHM 5.30: **Extending a Block Arnoldi Reduction**

(1) *Compute the QR factorization* $V_{m+1}H_{m+1,m} = F_m$ *using iterated classical Gram-Schmidt.*

(2) $V_{[m+1]} = \left[\begin{array}{cc} V_{[m]} & V_{m+1} \end{array} \right]$.

(3) $W = AV_{m+1}$

(4) $H_{m+1,m+1} = V_{m+1}^* W$.

(5) $H_{[m+1]} = \left[\begin{array}{cc} H_{[m]} & V_{[m]}^* W \\ H_{m+1,m} E_m^* & H_{m+1,m+1} \end{array} \right]$.

(6) $F_{m+1} = W - V_{[m+1]} \left[\begin{array}{c} V_{[m]}^* W \\ H_{m+1,m+1} \end{array} \right]$.

We now comment on some steps of Algorithm 5.30.

(1) Here the QR factorization is computed via an iterated classical Gram-Schmidt (CGS) algorithm using a possible correction step. See [38] 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 [57] matrix-vector multiplication subroutine `_GEMV`. 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 jth 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 [56] matrix-matrix multiplication subroutine `_GEMM` for computing $V_m^* W$.

(6) This is one step of block classical Gram-Schmidt (bCGR). The Level 3 BLAS [56] matrix-matrix multiplication subroutine `_GEMM` 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 [38] is used to determine whether a second orthogonalization step is needed. See [160] for details.

The scheme given for computing $V_{[m+1]}$ is equivalent to the one proposed by Ruhe [205] (for symmetric matrices) and is the one used by the implicitly restarted block Lanczos code [10]. Although the approach in [10] 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 [205], 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 step (3)–(5) reduce the cost of I/O by a factor of the blocksize and increase the amount of floating-point operations per memory reference.

5.6.2 Practical Algorithm

Algorithm 5.31 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 5.31: **Implicitly Restarted Block Arnoldi Method (BIRAM)**

For $i = 1, 2 \cdots$ *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 5.30

(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 hence retain a length r block Arnoldi reduction.*

5.6.2.1 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 blocksize 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 a polynomials of larger degree to be applied. The downside 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 $\mathcal{O}(\epsilon_M)$ 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.

5.6.2.2 Stopping Criterion

Suppose that (s, θ) is an eigenpair of $H_{[m]}$. It follows easily from equation (5.26) that

$$AV_{[m]} s - \theta V_{[m]} s = F_m E_m^* s, \tag{5.27}$$

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 (5.27) 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 (5.27). The eigenvalues are partitioned as in (5.28) so that the wanted ones correspond to the eigenvalues of A desired.

5.6.2.3 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 §??. 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 $m \cdot b$.

5.6.2.4 Restarting a Block Arnoldi Reduction

§?? motivated and explained how to implicitly restarted Arnoldi method. The experimental results in [?, 155, 10] show that an implicitly restarted scheme is superior to other block methods that have appeared in the literature [216, 222]. 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 [216, 111, 122, 222] for information on explicitly restarted Arnoldi methods.

The extension of the scheme in §?? 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 [101, 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

$$\{\underbrace{\theta_1, \ldots, \theta_r}_{\text{wanted}}\} \cup \{\underbrace{\theta_{r+1}, \ldots, \theta_{m \cdot b}}_{\text{unwanted}}\}. \tag{5.28}$$

For an unblocked reduction, the p shifts are selected from the unwanted eigenvalues of $H_{[m]}$ where $r = k$. Sorensen [234] proposed this as a *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 [209], Harmonic Ritz values [175, 180, 187, 227], the roots of Leja polynomials [9], the roots of least squares polynomials [210] and refined shifts [121]. In particular, the Leja and Harmonic Ritz values have been used to estimate the interior eigenvalues of A.

In Algorithm 5.31, 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 [212, p. 181] and used in the implementations in [10, 222].) 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 [237] discusses an adaptive strategy for symmetric eigenvalue problems.

5.7 Lanczos Method

Z. Bai and D. Day

The non-Hermitian Lanczos method is an oblique projection method (see §2.7) for solving the non-Hermitian eigenvalue problem,

$$Ax = \lambda x \quad \text{and} \quad y^* A = \lambda y^*, \tag{5.29}$$

where A is a non-Hermitian matrix. For complex symmetric matrices ($A = A^T$ but $A \neq A^*$), a special Lanczos method is presented §5.10.

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 Arnoldi's method discussed in §5.4. 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 matrices. This section defines the basic Lanczos method and its main properties, and presents algorithmic techniques that enhance the method's numerical stability and accuracy.

5.7.1 Algorithm

The non-Hermitian Lanczos method as presented in Algorithm 5.32 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 $\{q_1, Aq_1, A^2q_1, \ldots\}$ and $\{p_1, A^*p_1, (A^*)^2p_1, \ldots, \}$. The two sequences of vectors $\{q_i\}$ and $\{p_i\}$ are generated using the three-term recurrences:

$$\beta_{j+1} q_{j+1} = Aq_j - \alpha_j q_j - \gamma_j q_{j-1} \tag{5.30}$$

$$\bar{\gamma}_{j+1} p_{j+1} = A^* p_j - \bar{\alpha}_j p_j - \bar{\beta}_j p_{j-1}. \tag{5.31}$$

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 jth step, the Lanczos method generates two $n \times j$ matrices Q_j and P_j:

$$Q_j = [q_1, q_2, \ldots, q_j], \qquad P_j = [p_1, p_2, \ldots, p_j]$$

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:

$$AQ_j = Q_j T_j + \beta_{j+1} q_{j+1} e_j^* , \tag{5.32}$$

$$A^* P_j = P_j T_j^* + \bar{\gamma}_{j+1} p_{j+1} e_j^* , \tag{5.33}$$

$$P_j^* Q_j = I_j . \tag{5.34}$$

In addition, $P_j^* q_{j+1} = 0$ and $p_{j+1}^* Q_j = 0$. The relation (5.34) shows that the Lanczos vectors (bases) are bi-orthonormal. 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,

$$P_j^* A Q_j = T_j . \tag{5.35}$$

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,

$$x_i^{(j)} = Q_j z_i^{(j)} \quad \text{and} \quad y_i^{(j)} = P_j w_i^{(j)} . \tag{5.36}$$

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

$$r_i^{(j)} = A x_i^{(j)} - \theta_i^{(j)} x_i^{(j)} \tag{5.37}$$

$$(s_i^{(j)})^* = (y_i^{(j)})^* A - \theta_i^{(j)} (y_i^{(j)})^* \tag{5.38}$$

to the norms of $x_i^{(j)}$ and $y_i^{(j)}$ respectively. Moreover, by equation (5.32), the right residual vector becomes

$$r_i^{(j)} = \beta_{j+1} q_{j+1} (e_j^* z_i^{(j)}) . \tag{5.39}$$

and by equation (5.33), the left residual vector becomes

$$(s_i^{(j)})^* = \gamma_{j+1} p_{j+1}^* ((w_i^{(j)})^* e_j) . \tag{5.40}$$

Therefore, as in the Hermitian case (see §3.4), 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 §5.7.2 for a more detailed discussion of this topic.

ALGORITHM 5.32: **Non-Hermitian Lanczos Method**

(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** $j = 1, 2, \ldots,$ **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) $p_{j+1} = s / \overline{\gamma}_{j+1}$
(15) *compute eigentriplets* $(\theta_i^{(j)}, z_i^{(j)}, w_i^{(j)})$ *of* T_j
(16) *test for convergence*
(17) *rebiorthogonalize if necessary*
(18) $r = Aq_{j+1}$
(19) $s = A^* p_{j+1}$
(20) $r = r - q_j \overline{\gamma}_{j+1}$
(21) $s = s - p_j \overline{\beta}_{j+1}$
(22) **end for**
(23) *compute approximate eigenvectors* $x_i^{(j)}$ *and* $y_i^{(j)}$

We will now comment on certain steps of Algorithm 5.32:

(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-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 $\{q_1, q_2, \ldots, q_j\}$ 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 breaks down essentially. 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 [86] and implemented in QMRpack (see §5.7.4 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 [190] for a further discussion. A different treatment of the breakdown is discussed in §5.8.

In practice, exact breakdown is rare. Near breakdowns occur more often, i.e., ω_j is non-zero but extremely small in absolute value. Near breakdowns cause stagnation and instability. Any criterion for detecting a near breakdown either forces to 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 eigen-decomposition of the tridiagonal matrix T_j by the QR algorithm (see §5.2) is approximately $30j^3$ flops per iteration, and the cumulative cost for steps one to j is $15j^4$ flops. Solving the eigen-problem periodically, say at every 10 steps, reduces this cost.

No stable algorithm has been found for eigenvalue problem that approximates the eigenvalues of a general tridiagonal in j^2 flops, though recently conditionally stable algorithms such as [37, 97] 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 (5.35) approximate all the desired eigenvalues of A with sufficiently small residuals, which are calculated according to the equations (5.39) and (5.40). Convergence properties are discussed in more detail in §5.7.2.

If there is no re-biorthogonalization (see [43]), 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 ([33]) 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. Under conditions specified in [43], one may use the two-sided modified Gram-Schmidt process [190] to re-biorthogonalize the $j+1$st Lanczos vectors;

$$\begin{aligned}&\text{for } i = 1, 2, \ldots, j\\ &\quad q_{j+1} = q_{j+1} - q_i(p_i^* q_{j+1})\\ &\quad p_{j+1} = p_{j+1} - p_i(q_i^* p_{j+1})\end{aligned}$$

end for

Applying the two-sided modified Gram-Schmidt process to at each step is very costly and becomes a computational bottleneck. In [43], an efficient alternative scheme is proposed. This topic is revisited in §5.8.

5.7.2 Convergence properties

In this section, we discuss convergence properties in detail. First the shift-invert spectral transformation 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

$$C = (A - \mu I)^{-1} \tag{5.41}$$

Eigenvalues in the complex plane nearest the shift μ converge first. Note that the LU factorization,

$$A - \mu I = LU \tag{5.42}$$

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.

An inherent difficulty of a non-Hermitian eigen-problem that the known bounds on the distance from $\theta_i^{(j)}$ to an eigenvalue of A are not practical to compute [239, 130]. 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 eigen-triplet $(\theta_i^{(j)}, x_i^{(j)}, y_i^{(j)})$. Eigenvalue sensitivities (condition numbers) are also available. Recall that residual vectors $r_i^{(j)}$ and $s_i^{(j)}$ as defined in (5.37) and (5.38). Next observe that the biorthogonality condition (5.34) implies that $(s_i^{(j)})^* x_i^{(j)} = (y_i^{(j)})^* r_i^{(j)} = 0$. Together these relations imply that

$$F = \frac{r_i^{(j)}(x_i^{(j)})^*}{\|x_i^{(j)}\|_2^2} + \frac{y_i^{(j)}(s_i^{(j)})^*}{\|y_i^{(j)}\|_2^2} \tag{5.43}$$

is the desired perturbation such that

$$(A - F)x_i^{(j)} = x_i^{(j)}\theta_i^{(j)}, \tag{5.44}$$

$$(y_i^{(j)})^*(A - F) = \theta_i^{(j)}(y_i^{(j)})^*, \tag{5.45}$$

and $\|F\|_{\mathrm{F}}^2 = \|r_i^{(j)}\|_2^2/\|x_i^{(j)}\|_2^2 + \|s_i^{(j)}\|_2^2/\|y_i^{(j)}\|_2^2$. See §9.1 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 [33]: one either re-runs the three term recurrences to generate the requested eigenvectors, or applies inverse iteration (see §5.3). The complication is that the decision whether or not to accept a Ritz vector as an approximate eigenvector cannot be based on these un-normalized 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

$$\frac{\|r_i^{(j)}\|_2}{\|x_i^{(j)}\|_2} \qquad \text{and} \qquad \frac{\|s_i^{(j)})\|_2}{\|y_i^{(j)}\|_2}$$

to user specified tolerance. Normalized residuals can be larger in norm than the un-normalized 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 [33]. This case is addressed either by re-biorthogonalization [43] or by not computing the Ritz vectors that correspond to "spurious" modes [33].

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 [87, 190, 43]. 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 [190]. In [13] it is 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 [87, 43]. And in this case the un-normalized 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.

5.7.3 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 [115]. This means that in exact arithmetic the Lanczos algorithm computes a complete chain of principal vectors for a multiple eigenvalue.

5.7.4 Software Availability

A Fortran implementation of the non-Hermitian Lanczos procedure with look-ahead to cure breakdowns is available in QMRPACK [88]. 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 the identification test as described at the step (16) of Algorithm 5.32.

A set of Matlab routines for implementing the adaptive block Lanczos method described in §5.8 can be used to implement Algorithm 5.32 by defining the blocksize as one at the initial step. Optional re-biorthogonalization schemes are available. An adaptive blocksize scheme is used for the treatment of (near) breakdown and/or multiple or closely clustered eigenvalues of interest.

5.7.5 Notes and References

Lanczos' original 1950 paper studied the eigenvalue problem $Ax = \lambda x$ for a general matrix A [152] [3]. Cullum and Willoughby [33] demonstrated the success of the the nonsymmetric Lanczos method with no reorthogonalization for a number of nonsymmetric eigen-problems 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 [245], and Parlett, Taylor and Liu [197] 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 [86]. Ye also proposed a scheme to cure the breakdown by choosing a new starting vector to generate another Krylov subspace that includes the old one in an appropriate way [260]. The basic idea is incorporated in the ABLE method proposed by Bai, Day and Ye (1995) [13] (see §5.8). Theoretical studies of the breakdown phenomena and connections with the underlying polynomial theory can be found in papers by Gutknecht [107, 108] and Brezinski [23].

Since the Lanczos method is an oblique projection method, the general convergence analysis of projection methods by Saad [208] and Jia [120] is applicable, with significant limitations in practical usage.

[3] Historical note: it appears that Lanczos did not know Krylov's work. Ostrowski informed Lanczos about the early parallel work of Krylov.

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 [11]. Day has published an error analysis of the nonsymmetric Lanczos process in finite precision arithmetic [42, 43]. 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 [224] is presented. A partial reorthogonalization scheme was also published recently by van der Veen and Vuik [246].

5.8 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 §5.7. It uses block versions of the three-term recursions (5.31) and (5.30). 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 blocksize. 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.

In §5.9, 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, an area closely related to matrix eigenvalue problems.

5.8.1 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

$$Q_{[j]} = [\,Q_1, Q_2, \ldots, Q_j\,] \quad \text{and} \quad P_{[j]} = [\,P_1, P_2, \ldots, P_j\,]$$

that satisfy the biorthonormality condition

$$P_{[j]}^* Q_{[j]} = I. \tag{5.46}$$

The block Lanczos method uses three term recurrences

$$\begin{aligned} Q_{j+1} C_{j+1} &= A Q_j - Q_j A_j - Q_{j-1} B_j, & (5.47)\\ P_{j+1} B_{j+1}^* &= A^* P_j - P_j A_j^* - P_{j-1} C_j^*, & (5.48) \end{aligned}$$

for $j = 1, 2, \ldots$, where $Q_0 = P_0 = 0$. In matrix notation, these quantities satisfy the governing relations

$$\begin{aligned} A Q_{[j]} &= Q_{[j]} T_{[j]} + Q_{j+1} C_{j+1} E_j^*, & (5.49)\\ A^* P_{[j]} &= P_{[j]} T_{[j]}^* + P_{j+1} B_{j+1}^* E_j^*, & (5.50) \end{aligned}$$

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 eigen-triplet $(\theta_i^{(j)}, z_i^{(j)}, w_i^{(j)})$ 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 eigen-triplets of A first.

To assess the approximation, let $r_i^{(j)}$ and $s_i^{(j)}$ denote the corresponding right and left residual vectors:

$$\begin{aligned} r_i^{(j)} &= A x_i^{(j)} - \theta_i^{(j)} x_i^{(j)} & (5.51)\\ (s_i^{(j)})^* &= (y_i^{(j)})^* A - \theta_i^{(j)} (y_i^{(j)})^*. & (5.52) \end{aligned}$$

Substitute (5.49) and (5.50) respectively, and there appears

$$\begin{aligned} r_i^{(j)} &= Q_{j+1} C_{j+1} (E_j^* z_i^{(j)}), & (5.53)\\ (s_i^{(j)})^* &= ((w_i^{(j)})^* E_j) B_{j+1} P_{j+1}^*. & (5.54) \end{aligned}$$

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 §5.7).

A simple blocked version of the basic non-Hermitian Lanczos method is presented in Algorithm 5.33.

ALGORITHM 5.33: **Basic Block Lanczos Method**

(1) *Choose $n \times p$ starting blocks P_1 and Q_1 such that $P_1^* Q_1 = I$*
(2) $R = AQ_1$
(3) $S = A^* P_1$
(4) **for** $j = 1, 2, \ldots,$ **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** Σ_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) *Rebiorthogonalize if necessary*
(20) $R = AQ_{j+1}$
(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 5.33.

(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 blocksize 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 sub-

space has been revealed. Continuing the recurrence in the rank deficient case is discussed in §5.8.2.

(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*. A proper action must be taken to continue. See §5.8.2 for possible treatments.

(17), (18) The eigentriplets of $T_{[j]}$ can be computed using the method discussed in §5.2. The convergence of Ritz values can be tested by the norms of residuals (5.53) and (5.54). For more details on accessing the accuracy of the approximation, see § 5.7.2 and 9.1 and [13].

When j becomes large, solving the eigenproblem of $T_{[j]}$ becomes time-comsuming. A simple way to reduce the cost is to do this periodically, say every 5 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 Two-Sided Modified Gram-Schmidt (TSMGS) process can be used to biorthogonalize Q_{j+1} and P_{j+1} in place against the previous Lanczos vectors $Q_{[j]} = [\,Q_1, Q_2, \cdots, Q_j\,]$ and $P_{[j]} = [\,P_1, P_2, \cdots, P_j\,]$.

PROCEDURE TSMGS
for $i = 1, 2, \ldots, j$
$Q_{j+1} := Q_{j+1} - Q_i(P_i^* Q_{j+1})$
$P_{j+1} := P_{j+1} - P_i(Q_i^* P_{j+1})$
end

(25) This is recommended only if the 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 (5.51) and (5.52) 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.

5.8.2 An Adaptively Blocked Lanczos Method

Algorithm 5.33 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. The adaptive block Lanczos method (ABLE) avoids 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, $\|P_{[j]}^* Q_{[j]} - I\| \approx \epsilon$, or *semi-biorthogonality*, i.e, $\|P_{[j]}^* Q_{[j]} - I\| \approx \sqrt{\epsilon}$.

Algorithm 5.34 is a pseudo-code 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 5.33. Essentially, the three term recurrences (5.47) and (5.48) 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 $10n\epsilon$, where ϵ 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. Eigen-triplets 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 $\epsilon^{1/2}$.

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 [33] and discussed in §5.7. Such phenomena are not anticipated in the higher levels of ABLE.

ALGORITHM 5.34: **ABLE**

(1) *Choose $n \times p$ starting blocks P_1 and Q_1 such that $P_1^* Q_1 = I$*
(2) $R = AQ_1$
(3) $S = A^* P_1$
(4) **for** $j = 1, 2, \ldots, j_{\max}$ **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)
end if
(11) *Compute the SVD:* $P_{j+1}^* Q_{j+1} = U_j \Sigma_j V_j^*$
(12) **if** $\omega_j = \min(\mathrm{diag}(\Sigma_j)) <$ `tolbd`,
breakdown := `T`
if ***treatbd*** $=$ `F`, **method fails, stop**
end if
(13) **if** ***group*** $=$ `T` *and (**fullbo** =* `T` *or **semibo** =* `T`*),*
Detect the clusters of converged Ritz values
end if
(14) **if** (breakdown $=$ `T` *and **treatbd** =* `T`*) or **group** =* `T`
Cure breakdown or adapt to clusters
end if
(15) $B_{j+1} := B_{j+1} U_j \Sigma_j^{1/2}$
(16) $C_{j+1} := \Sigma_j^{1/2} V_j^* C_{j+1}$
(17) $Q_{j+1} := Q_{j+1} V_j \Sigma_j^{-1/2}$
(18) $P_{j+1} := P_{j+1} U_j \Sigma_j^{-1/2}$
(19) *Compute eigentriplets* $(\theta_i^{(j)}, z_i^{(j)}, w_i^{(j)})$ *of* $T_{[j]}$
(20) *Test for convergence*
(21) **if** ***fullbo*** $=$ `T`
Maintain full biorthogonality
else if ***semibo*** $=$ `T`,
Monitor biorthogonality loss and correct if necessary
end if
(22) $R = AQ_{j+1}$
(23) $S = A^* S_{j+1}$
(24) $R = R - Q_j B_{j+1}$
(25) $S = S - P_j C_{j+1}^*$
(26) **end for**
(27) *Compute approximate eigenvectors* $x_i^{(j)}$ *and* $y_i^{(j)}$

We now comment on some steps of Algorithm 5.34:

(1) The comment to Algorithm 5.33 step (1) applies here as well.

(2), (3), (22), (23) The comment to Algorithm 5.33 steps (2),(3) and (20),(21) 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):

$$R := R - Q_j(P_j^* R)$$
$$S := S - P_j(Q_j^* S)$$

(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 $Q_{[j]} = [\,Q_1, Q_2, \cdots, Q_j\,]$ and $P_{[j]} = [\,P_1, P_2, \cdots, P_j\,]$, using TSMGS.

(12) 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.

(13) 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 $|\theta_i^{(j)} - \theta_k^{(j)}| \leq \texttt{tolcl} \cdot |\theta_i^{(j)}|$, where `tolcl` is the user prescribed tolerance.

(14) 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:

$$Q_{j+1} := \left[\begin{array}{cc} Q_{j+1} & \widehat{Q} \end{array}\right], \quad P_{j+1} := \left[\begin{array}{cc} P_{j+1} & \widehat{P} \end{array}\right],$$

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 blocksize before argument and $p_{\max}$ is the prescribed maximum block size. The new blocksize is $p_j + \delta$. TSMGS is then invoked to bi-orthogonalize Q_{j+1} and P_{j+1} against previous Lanczos vectors stored in $Q_{[j]}$ and $P_{[j]}$.

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 `tolbd` $= 0$, i.e., the exact breakdown. One may choose $\widehat{Q}$ and $\widehat{P}$ as random vectors.

(19), (20) If semi-biorthogonality is maintained, we solve the eigen-problem of $T_{[j]}$ after each re-biorthogonalization.

The formulas (5.53) and (5.54) can be used to detect convergence. Since we expect to many steps in the Lanczos iteration, at Level 1, we use the following convergence test:

$$\min\{\|r_i^{(j)}\|, \|(s_i^{(j)})^*\|\} \leq \texttt{tolconv} \cdot \|A\|_1,$$

At the higher levels we may use the semi-quadratic convergence test:

$$\min\left\{\|r_i^{(j)}\|, \|(s_i^{(j)})^*\|, \eta_i^{(j)}\right\} \leq \texttt{tolconv} \cdot \|A\|_1,$$

where

$$\eta_i^{(j)} = \frac{\|r_i^{(j)}\| \, \|(s_i^{(j)})^*\|_2}{\mathrm{gap}(\theta_i^{(j)}, T_{[j]})},$$

$\mathrm{gap}(\theta_i^{(j)}, T_{[j]})$ defines the gap between $\theta_i^{(j)}$ and other Ritz values. Specifically, $\mathrm{gap}(\theta_i^{(j)}, T_{[j]}) = \min_{k \neq i} |\theta_i^{(j)} - \theta_k^{(j)}|$. In the blocked case, the gap is defined between the clusters.

In [13] it is shown that under mild conditions, for a Ritz value $\theta_i^{(j)}$, there is an eigenvalues 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 semi-quadratic 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 §9.1).

(21) 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 is developed in [13] to efficiently simulate this quantity at $\mathcal{O}(n)$ cost.

(27) The comment to Algorithm 5.33 step (25) also applies here.

5.8.2.1 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 3-term recurrences	$matrix + 6pn + 3p^2j^2 + 4pj$
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 [105, 217], 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 3-term	2	6	8	2	2
fullbo		$2j$	$2j$		
semibo		2			

where 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 *m*ultiple-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 $2p^2n$ flops, respectively. QR decomposition (QRD) of an $n \times p$ ($p \ll n$) matrix costs $2p^2n$ flops. 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 blocksize 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 two-sided MGS procedure.
- It costs about $30(jp)^3$ for computing all eigentriplets of the block tridiagonal matrix $T_{[j]}$.

5.8.3 Software Availability

A Matlab implementation of ABLE is to be available at the project homepage.

5.8.4 Notes and References

A block version of the Lanczos method in the symmetric case appeared in the work of Cullum and Donath [34] and Golub and Underwood [100]. The so-called band Lanczos algorithm proposed by Ruhe is an implementation variation of the block Lanczos method [205]. The implementation of the block symmetric Lanczos method by Grimes, Lewis and Simon represents the the-state-of-the-art [104], this software is proprietary.

A block version of the nonsymmetric Lanczos method is straightforward, but has many pitfalls awaiting for the unwary implementor. Bai studied a simple implementation of the

block Lanczos method [12]. Aliaga, Boley, Freund and Hernandez generalized Ruhe's band Lanczos implementation to the nonsymmetric case [1]. This is presented in §5.9.

The adaptive block Lanczos method for nonsymmetric eigenvalue problems (in short, ABLE) proposed in this section is presented by Bai, Day and Ye [13]. ABLE also implements a block version of the algorithm presented in [42, 43], 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 Chronopoulus [134] for better data locality on parallel vector computers.

5.9 Band Lanczos Method

R. Freund

The standard non-Hermitian Lanczos algorithm as presented in §5.7 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 non-Hermitian eigenproblem,

$$Ax = \lambda x. \tag{5.55}$$

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 §5.9.4 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 have been discussed for blocks of equal size, in §5.8.

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,

$$b_1, b_2, \ldots, b_m \quad \text{and} \quad c_1, c_2, \ldots, c_p. \tag{5.56}$$

The matrix A and the vectors (5.56) 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 (5.56) induce the right block-Krylov sequence

$$b_1, b_2, \ldots, b_m, Ab_1, Ab_2, \ldots, Ab_m, \ldots, A^{k-1}b_1, \ldots, A^{k-1}b_m, \ldots, \tag{5.57}$$

and the left block-Krylov sequence

$$c_1, c_2, \ldots, c_p, A^Tc_1, A^Tc_2, \ldots, A^Tc_p, \ldots, (A^T)^{k-1}c_1, \ldots, (A^T)^{k-1}c_p, \ldots . \tag{5.58}$$

The goal of the band Lanczos algorithm is to construct suitable right and left Lanczos vectors,

$$v_1, v_2, \ldots, v_j \quad \text{and} \quad w_1, w_2, \ldots, w_j, \tag{5.59}$$

that build bases for the subspaces spanned by the first j linearly independent vectors of the block-Krylov sequence (5.57) and (5.58), 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 §3.6, 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 §5.8. 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 §5.9.5 below.

5.9.1 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 §3.6.1 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 (5.57) means that a vector, say $A^i b_j$, in the sequence (5.57) is linearly dependent on vectors to the left of $A^i b_j$ in (5.57) and that this vector and all its A-multiples are removed from (5.57). Similarly, an *exact deflation* in the left block-Krylov sequence (5.58) means that a vector in the sequence (5.58) is linearly dependent on previous Krylov vectors in (5.58) and that this vector and all its A^T-multiples are removed from (5.58). After m exact deflations in the right block-Krylov sequence (5.57) have occurred, the remaining non-deflated vectors of (5.57) span an A-invariant subspace. Furthermore, the right Lanczos vectors in (5.59) build a suitable basis for this A-invariant subspace and all the eigenvalues of the Lanczos matrix $T_j^{(\mathrm{pr})}$ defined in §5.9.2 below are also eigenvalues of A. Similarly, after p exact deflations in the left block-Krylov sequence (5.58) have occurred, the remaining non-deflated vectors of (5.58) span an A^T-invariant subspace. The left Lanczos vectors in (5.59) build a suitable basis for this A^T-invariant subspace and all the eigenvalues of the Lanczos matrix $T_j^{(\mathrm{pr})}$ 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*.

5.9.2 Basic Properties

After j iterations, the algorithm has generated the vectors (5.59). It will be convenient to introduce the notation

$$V_j = \left[\begin{array}{cccc} v_1 & v_2 & \cdots & v_j \end{array} \right] \quad \text{and} \quad W_j = \left[\begin{array}{cccc} w_1 & w_2 & \cdots & w_j \end{array} \right] \tag{5.60}$$

for the matrices whose columns are just the right and left Lanczos vectors (5.59), respectively. The vectors (5.59) are constructed to be bi-orthogonal. Using the notation (5.60), the bi-orthogonality can be stated compactly as follows:

$$W_j^T V_j = D_j = \mathrm{diag}\,(\delta_1, \delta_2, \ldots, \delta_j)\,. \tag{5.61}$$

In order to enforce bi-orthogonality of the next Lanczos vectors, the algorithm involves division by the diagonal entries, δ_k, in (5.61). As a result, the algorithm has to be stopped as soon as

$$\delta_k = w_k^T v_k = 0, \quad \text{but} \quad v_k,\ w_k \neq 0, \tag{5.62}$$

occurs. The situation (5.62) 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 (5.59), the algorithm has constructed the vectors

$$\widehat{v}_{j+1}, \widehat{v}_{j+2}, \ldots, \widehat{v}_{j+m_c} \quad \text{and} \quad \widehat{w}_{j+1}, \widehat{w}_{j+2}, \ldots, \widehat{w}_{j+p_c}. \tag{5.63}$$

The vectors $\widehat{v}_{j+1}, \widehat{v}_{j+2}, \ldots, \widehat{v}_{j+m_c}$ are the candidates for the next m_c right Lanczos vectors, $v_{j+1}, v_{j+2}, \ldots, v_{j+m_c}$, and the vectors $\widehat{w}_{j+1}, \widehat{w}_{j+2}, \ldots, \widehat{w}_{j+p_c}$ are the candidates for the next p_c left Lanczos vectors, $w_{j+1}, w_{j+2}, \ldots, w_{j+p_c}$. 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 (5.63) are constructed so that they satisfy the bi-orthogonality relations

$$\begin{aligned} W_j^T \widehat{v}_k &= 0 \quad \text{for all} \quad k = j+1, j+2, \ldots, j+m_c, \\ V_j^T \widehat{w}_k &= 0 \quad \text{for all} \quad k = j+1, j+2, \ldots, j+p_c. \end{aligned} \tag{5.64}$$

The algorithm has a very simple built-in deflation procedure based on the vectors (5.63). In fact, an exact deflation in the right block-Krylov sequence at iteration $j+1$ is equivalent to $\widehat{v}_{j+1} = 0$. Therefore, in the algorithm, one checks if $\|\widehat{v}_{j+1}\|_2$ is smaller than some suitable deflation tolerance. If yes, the vector $\widehat{v}_{j+1}$ is deflated and m_c is reduced by one. Otherwise, $\widehat{v}_{j+1}$ 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 $\widehat{w}_{j+1} = 0$. In the algorithm, one checks if $\|\widehat{w}_{j+1}\|_2$ is smaller than the deflation tolerance. If yes, the vector $\widehat{w}_{j+1}$ is deflated and p_c is reduced by one. Otherwise, $\widehat{w}_{j+1}$ is normalized to become the next left Lanczos vector w_{j+1}.

The recurrences that are used in the algorithm to generate the vectors (5.59) and (5.63) can be summarized compactly as follows:

$$\begin{aligned} AV_j &= V_j T_j + \begin{bmatrix} 0 & \cdots & 0 & \widehat{v}_{j+1} & \widehat{v}_{j+2} & \cdots & \widehat{v}_{j+m_c} \end{bmatrix} + \widehat{V}_j^{(\mathrm{dl})}, \\ A^T W_j &= W_j \widetilde{T}_j + \begin{bmatrix} 0 & \cdots & 0 & \widehat{w}_{j+1} & \widehat{w}_{j+2} & \cdots & \widehat{w}_{j+p_c} \end{bmatrix} + \widehat{W}_j^{(\mathrm{dl})}. \end{aligned} \tag{5.65}$$

Here, T_j and $\widetilde{T}_j$ are $j \times j$ matrices whose entries are chosen so that the bi-orthogonality conditions (5.61) and (5.64) are satisfied. The matrix $\widehat{V}_j^{(\mathrm{dl})}$ in (5.65) contains mostly zero columns, together with the $\widehat{v}_k$ vectors that have been deflated during the first j iterations. The matrix $\widehat{W}_j^{(\mathrm{dl})}$ in (5.65) contains mostly zero columns, together with the $\widehat{w}_k$ vectors that have been deflated during the first j iterations. We remark that $m - m_c$ is the number of deflated $\widehat{v}_k$ vectors and that $p - p_c$ is the number of deflated $\widehat{w}_k$ vectors. It turns out that bi-orthogonality 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 $\widetilde{T}_j$ 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 $\widehat{v}_k$ vector is deflated, and the upper bandwidth is reduced by 1 every time a $\widehat{w}_k$ vector is deflated. In addition, each deflation of a $\widehat{w}_k$ 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 $\widehat{w}_k$ 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

$$T_j = T_j^{(\mathrm{b})} + T_j^{(\mathrm{d})}, \tag{5.66}$$

where $T_j^{(\mathrm{b})}$ is a banded matrix and $T_j^{(\mathrm{d})}$ contains horizontal "spikes" above the band of T_j due to deflation of $\widehat{w}_k$ vectors. Similarly,

$$\widetilde{T}_j = \widetilde{T}_j^{(\mathrm{b})} + \widetilde{T}_j^{(\mathrm{d})},$$

where the banded part $\widetilde{T}_j^{(\mathrm{b})}$ has lower bandwidth p_c+1 and upper bandwidth m_c+1 and $\widetilde{T}_j^{(\mathrm{d})}$ contains horizontal "spikes" above the band of $\widetilde{T}_j$ due to deflation of $\widehat{v}_k$ vectors.

The entries of the matrices T_j and $\widetilde{T}_j$ are not independent of each other. More precisely, setting

$$T_j^{(\mathrm{pr})} = T_j + D_j^{-1} \left(\widetilde{T}_j^{(\mathrm{d})}\right)^T D_j \quad \text{and} \quad \widetilde{T}_j^{(\mathrm{pr})} = \widetilde{T}_j + D_j^{-1} \left(T_j^{(\mathrm{d})}\right)^T D_j\,, \tag{5.67}$$

we have

$$T_j^{(\mathrm{pr})} = D_j^{-1} \left(\widetilde{T}_j^{(\mathrm{pr})}\right)^T D_j\,, \tag{5.68}$$

where D_j is the diagonal matrix given by (5.61). Inserting (5.66) into the definition of $T_j^{(\mathrm{pr})}$ in (5.67), we obtain the relation

$$T_j^{(\mathrm{pr})} = T_j^{(\mathrm{b})} + T_j^{(\mathrm{d})} + D_j^{-1} \left(\widetilde{T}_j^{(\mathrm{d})}\right)^T D_j\,,$$

which shows that $T_j^{(\mathrm{pr})}$ consists of the banded part $T_j^{(\mathrm{b})}$, horizontal spikes due to deflation of $\widehat{w}_k$ vectors above the banded part, and vertical spikes due to deflation of $\widehat{v}_k$ 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 $\widehat{w}_k$ vectors have occurred at iterations $k = 8$, $k = 11$, and $k = 13$, and deflations of $\widehat{v}_k$ vectors have occurred at iterations $k = 7$ and $k = 12$. In this case, the matrix $T_{15}^{(\mathrm{pr})}$ has the following sparsity

structure:

$$T_{15}^{(\mathrm{pr})} = \begin{bmatrix}
* & * & * & * & * & * & & & & & & & & & \\
* & * & * & * & * & * & * & & & & & & & & \\
* & * & * & * & * & * & * & \mathtt{d} & \mathtt{d} & \mathtt{d} & \mathtt{d} & \mathtt{d} & \mathtt{d} & \mathtt{d} & \mathtt{d} \\
* & * & * & * & * & * & * & * & & & & & & & \\
 & * & * & * & * & * & * & * & * & & & & & & \\
 & & * & * & * & * & * & * & * & & & & & & \\
 & & & \widetilde{\mathtt{d}} & * & * & * & * & * & * & \mathtt{d} & \mathtt{d} & \mathtt{d} & \mathtt{d} & \mathtt{d} \\
 & & & \widetilde{\mathtt{d}} & & * & * & * & * & * & * & & & & \\
 & & & \widetilde{\mathtt{d}} & & & * & * & * & * & * & * & & & \\
 & & & \widetilde{\mathtt{d}} & & & & * & * & * & * & * & \mathtt{d} & \mathtt{d} & \mathtt{d} \\
 & & & \widetilde{\mathtt{d}} & & & & & * & * & * & * & * & & \\
 & & & \widetilde{\mathtt{d}} & & & & & & \widetilde{\mathtt{d}} & * & * & * & * & \\
 & & & \widetilde{\mathtt{d}} & & & & & & \widetilde{\mathtt{d}} & & * & * & * & * \\
 & & & \widetilde{\mathtt{d}} & & & & & & \widetilde{\mathtt{d}} & & & * & * & * \\
 & & & \widetilde{\mathtt{d}} & & & & & & \widetilde{\mathtt{d}} & & & & * & *
\end{bmatrix}.$$

Here, the $*$'s denote potentially nonzero entries within the banded part, $T_{15}^{(\mathrm{b})}$, the $\mathtt{d}$'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 $\widetilde{\mathtt{d}}$'s denote potentially nonzero entries due to the deflations of $\widehat{v}_k$ 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 non-Hermitian eigenvalue problem (5.55) 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 (5.55) of the form $x_i^{(j)} = V_j z_i^{(j)}$ and that, after inserting this ansatz for $x_i^{(j)}$ into (5.55), we multiply the resulting relation from the left by W_j^T. This yields the generalized eigenvalue problem

$$W_j^T A V_j z_i^{(j)} = \theta_i^{(j)} W_j^T V_j z_i^{(j)}, \quad i = 1, 2, \ldots, j. \tag{5.69}$$

Using the bi-orthogonality relations (5.61) and (5.64), it is easy to verify that the matrix $T_j^{(\mathrm{pr})}$ defined in (5.67) satisfies

$$T_j^{(\mathrm{pr})} = \left(W_j^T V_j\right)^{-1} W_j^T A V_j. \tag{5.70}$$

By (5.70), the generalized eigenvalue problem (5.69) is equivalent to the standard eigenvalue problem

$$T_j^{(\mathrm{pr})} z_i^{(j)} = \theta_i^{(j)} z_i^{(j)}, \quad i = 1, 2, \ldots, j.$$

We stress that, in the algorithm, we use the formula in (5.67) to obtain the entries of $T_j^{(\mathrm{pr})}$, rather than (5.70).

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 $\widehat{v}_k$ vectors have occurred, and thus the right block-Krylov sequence (5.57) is exhausted. In the case $p_c = 0$, p deflations of $\widehat{w}_k$ vectors have occurred, and thus the left block-Krylov sequence (5.58) is exhausted.

First consider termination due to $m_c = 0$. Then, the relation for the right Lanczos vectors in (5.65) can be rewritten as follows:

$$AV_j - V_j T_j^{(\mathrm{pr})} = P_j \widehat{V}_j^{(\mathrm{dl})}. \tag{5.71}$$

Here, the matrix

$$P_j = I - V_j D_j^{-1} W_j^T, \quad \text{where} \quad D_j = W_j^T V_j, \tag{5.72}$$

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 $T_j^{(\mathrm{pr})}$, and assume that $z_i^{(j)}$ is normalized so that $\left\| z_i^{(j)} \right\|_2 = 1$. 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 (5.71), it follows that the residual of this approximate eigensolution can be bounded as follows:

$$\left\| A x_i^{(j)} - \theta_i^{(j)} x_i^{(j)} \right\|_2 = \left\| P_j \widehat{V}_j^{(\mathrm{dl})} z_i^{(j)} \right\|_2 \leq \| P_j \|_2 \cdot \left\| \widehat{V}_j^{(\mathrm{dl})} \right\|_2. \tag{5.73}$$

In particular, if only exact deflation is performed, then $\widehat{V}_j^{(\mathrm{dl})} = 0$ and (5.73) shows that each eigenvalue $\theta_i^{(j)}$ of $T_j^{(\mathrm{pr})}$ 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 (5.65) can be rewritten as follows:

$$A^T W_j - W_j \widetilde{T}_j^{(\mathrm{pr})} = P_j^T \widehat{W}_j^{(\mathrm{dl})}. \tag{5.74}$$

Here, P_j is again the matrix defined in (5.72). Now let $\theta_i^{(j)}$ and $\widetilde{z}_i^{(j)}$ be any of the eigenpairs of $\left(T_j^{(\mathrm{pr})} \right)^T$, and assume that $\widetilde{z}_i^{(j)}$ is normalized such that $\left\| D_j^{-1} \widetilde{z}_i^{(j)} \right\|_2 = 1$. Note that the complex conjugate of $\widetilde{z}_i^{(j)}$ is a left eigenvector of $T_j^{(\mathrm{pr})}$. The pair $\theta_i^{(j)}$ and $y_i^{(j)} = W_j D_j^{-1} \widetilde{z}_i^{(j)}$ then represents an approximate eigensolution of A^T. From (5.74), it follows that the residual of this approximate eigensolution can be bounded as follows:

$$\left\| A^T y_i^{(j)} - \theta_i^{(j)} y_i^{(j)} \right\| = \left\| P_j^T \widehat{W}_j^{(\mathrm{dl})} D_j^{-1} \widetilde{z}_i^{(j)} \right\|_2 \leq \left\| P_j^T \right\|_2 \cdot \left\| \widehat{W}_j^{(\mathrm{dl})} \right\|_2. \tag{5.75}$$

In particular, if only exact deflation is performed, then $\widehat{W}_j^{(\mathrm{dl})} = 0$ and (5.75) shows that each eigenvalue $\theta_i^{(j)}$ of $T_j^{(\mathrm{pr})}$ is indeed an eigenvalue of A^T and thus of A.

5.9.3 The Algorithm

A complete statement of the band Lanczos algorithm is as follows.

ALGORITHM 5.35: **Non-Hermitian band Lanczos**

Set $\widehat{v}_k = b_k$ *for* $k = 1, 2, \ldots, m$.
set $\widehat{w}_k = c_k$ *for* $k = 1, 2, \ldots, p$.
Set $m_c = m$, $p_c = p$, *and* $\mathcal{I}_v = \mathcal{I}_w = \emptyset$.
(2) *For* $j = 1, 2, \ldots,$ *until convergence or* $m_c = 0$ *or* $p_c = 0$ *do*:
(3) *Compute* $\|\widehat{v}_j\|_2$.
(4) *Decide if* $\widehat{v}_j$ *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* $k = j, j+1, \ldots, j + m_c - 1$, *set* $\widehat{v}_k = \widehat{v}_{k+1}$.
d) *Return to Step* (3).
(5) *Compute* $\|\widehat{w}_j\|_2$.
(6) *Decide if* $\widehat{w}_j$ *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* $k = j, j+1, \ldots, j + p_c - 1$, *set* $\widehat{w}_k = \widehat{w}_{k+1}$.
d) *Return to Step* (5).
(7) *Set* $t_{j,j-m_c} = \|\widehat{v}_j\|_2$ *and* $\widetilde{t}_{j,j-p_c} = \|\widehat{w}_j\|_2$.
Normalize $v_j = \widehat{v}_j / t_{j,j-m_c}$ *and* $w_j = \widehat{w}_j / \widetilde{t}_{j,j-p_c}$.
(8) *Compute* $\delta_j = w_j^T v_j$. *If* $\delta_j = 0$, *stop.*
(9) *Compute* $\widehat{v}_{j+m_c} = A v_j$.
(10) *Set* $k_v = \max\{1, j - p_c\}$.
Set $\mathcal{I}_v^{(\mathrm{e})} = \mathcal{I}_v \cup \{\, k_v, k_v + 1, \ldots, j - 1 \,\}$.
For $k \in \mathcal{I}_v^{(\mathrm{e})}$ (*in ascending order*), *set*
$t_{kj} = w_k^T \widehat{v}_{j+m_c} / \delta_k$ *and* $\widehat{v}_{j+m_c} = \widehat{v}_{j+m_c} - v_k t_{kj}$.
(11) *Compute* $\widehat{w}_{j+p_c} = A^T w_j$.
(12) *Set* $k_w = \max\{1, j - m_c\}$.
Set $\mathcal{I}_w^{(\mathrm{e})} = \mathcal{I}_w \cup \{\, k_w, k_w + 1, \ldots, j - 1 \,\}$.
For $k \in \mathcal{I}_w^{(\mathrm{e})}$ (*in ascending order*), *set*
$\widetilde{t}_{kj} = \widehat{w}_{j+p_c}^T v_k / \delta_k$ *and* $\widehat{w}_{j+p_c} = \widehat{w}_{j+p_c} - w_k \widetilde{t}_{kj}$.
(13) *For* $k = j+1, j+2, \ldots, j+m_c$, *set*
$t_{j,k-m_c} = w_j^T \widehat{v}_k / \delta_j$ *and* $\widehat{v}_k = \widehat{v}_k - v_j t_{j,k-m_c}$.
(14) *For* $k = j+1, j+2, \ldots, j+p_c$, *set*
$\widetilde{t}_{j,k-p_c} = \widehat{w}_k^T v_j / \delta_j$ *and* $\widehat{w}_k = \widehat{w}_k - w_j \widetilde{t}_{j,k-p_c}$.
(15) *For* $k \in \mathcal{I}_w$, *set* $t_{jk} = \widetilde{t}_{kj}\, \delta_k / \delta_j$.
(16) *Set* $T_j^{(\mathrm{pr})} = [\, t_{ik} \,]_{i,k=1,2,\ldots,j}$.
(17) *Test for convergence.*

We stress that the non-Hermitian band Lanczos method can be implemented by directly following the statement in Algorithm 5.35, together with the further details of some of the steps given below. To keep the length of the statement to one page, in Algorithm 5.35, all potentially nonzero entries of the banded parts of T_j and $\widetilde{T}_j$ 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

$$t_{ik} = \widetilde{t}_{ki}\, \delta_k / \delta_i, \tag{5.76}$$

which follows from the connection (5.68) of $T_j^{(\mathrm{pr})}$ and $\widetilde{T}_j^{(\mathrm{pr})}$. In particular, in the following discussion of Steps (10), (12), (13), and (14), we give formulas for the entries of T_j and $\widetilde{T}_j$ that use (5.76) whenever possible. Employing these formulas not only minimizes the number of inner products, but it typically also enhances numerical stability. Therefore, we recommend to use these formulas in an actual implementation of the non-Hermitian band Lanczos method.

Next, we discuss some of the steps of Algorithm 5.35 in more detail.

(4) Generally, the decision if $\widehat{v}_j$ needs to be deflated should be based on checking if

$$\|\widehat{v}_j\|_2 \leq \texttt{dtol}, \tag{5.77}$$

where `dtol` is a suitably chosen small deflation tolerance. The vector $\widehat{v}_j$ is then deflated if (5.77) 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 $\widetilde{T}_j^{(\mathrm{d})}$ of $\widetilde{T}_j$, and the current right block size is updated to $m_c = m_c - 1$. If $m_c = 0$, then the right block-Krylov sequence (5.57) is exhausted, and the algorithm terminates naturally. If $m_c > 0$, the vector $\widehat{v}_j$ is deleted, the index $k+1$ of each of the remaining right candidate vectors $\widehat{v}_{k+1}$ is reset to k, and finally, the algorithm returns to Step (3). If (5.77) is not satisfied, then no deflation is necessary and the algorithm proceeds with Step (5).

(6) In analogy to (5.77), the decision if $\widehat{w}_j$ needs to be deflated is based on checking if

$$\|\widehat{w}_j\|_2 \leq \texttt{dtol}. \tag{5.78}$$

The vector $\widehat{w}_j$ is then deflated if (5.78) 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 $T_j^{(\mathrm{d})}$ 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 (5.58) is exhausted, and the algorithm terminates naturally. If $p_c > 0$, the vector $\widehat{w}_j$ is deleted, the index $k+1$ of each of the remaining left candidate vectors $\widehat{w}_{k+1}$ is reset to k, and finally, the algorithm returns to Step (5). If (5.78) is not satisfied, then no deflation is necessary and the algorithm proceeds with Step (7).

(7) Both vectors $\widehat{v}_j$ and $\widehat{w}_j$ 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

$$\|v_j\|_2 = \|w_j\|_2 = 1 \quad \text{for all} \quad j.$$

(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, $\widehat{v}_{j+m_c}$, as the A-multiple of the latest right Lanczos vector v_j.

(10) The vector $\widehat{v}_{j+m_c}$ is bi-orthogonalized against the left Lanczos vectors w_k, $k \in \mathcal{I}_v^{(\mathrm{e})}$. Note that a two-sided modified Gram-Schmidt procedure is used to do this bi-orthogonalization. One of the inner products required in the computation of

t_{kj} can be saved by employing (5.76). More precisely, the following formula for t_{kj} should be used:

$$t_{kj} = \begin{cases} \widetilde{t}_{jk}\,\delta_j/\delta_k & \text{if } \; k = j - p_c, \\ w_k^T \widehat{v}_{j+m_c}/\delta_k & \text{otherwise.} \end{cases} \tag{5.79}$$

Note that the necessary entry $\widetilde{t}_{jk}$ in (5.79) is available from Step (7).

(11) In this step, we advance the left block-Krylov sequence by computing a new candidate vector, $\widehat{w}_{j+p_c}$, as the A^T-multiple of the latest left Lanczos vector w_j.

(12) The vector $\widehat{w}_{j+p_c}$ is bi-orthogonalized against the right Lanczos vectors v_k, $k \in \mathcal{I}_w^{(\mathrm{e})}$. Note that this bi-orthogonalization is done by means of a two-sided modified Gram-Schmidt procedure. Again, to save one inner product, the following formula for $\widetilde{t}_{kj}$ should be used:

$$\widetilde{t}_{kj} = \begin{cases} t_{jk}\,\delta_j/\delta_k & \text{if } \; k = j - m_c, \\ \widehat{w}_{j+p_c}^T v_k/\delta_k & \text{otherwise.} \end{cases}$$

(13) The right candidate vectors, $\widehat{v}_{j+1}, \widehat{v}_{j+1}, \ldots, \widehat{v}_{j+m_c}$, are bi-orthogonalized against the latest left Lanczos vector w_j. To save inner products and also enhance numerical stability, the following formula for $t_{j,k-m_c}$ should be used:

$$t_{j,k-m_c} = \begin{cases} w_j^T \widehat{v}_k/\delta_j & \text{if } \; k - m_c \le 0 \;\text{ or }\; k - m_c = j, \\ \widetilde{t}_{k-m_c,j}\,\delta_{k-m_c}/\delta_j & \text{otherwise.} \end{cases}$$

(14) The left candidate vectors, $\widehat{w}_{j+1}, \widehat{w}_{j+1}, \ldots, \widehat{w}_{j+p_c}$, are bi-orthogonalized against the latest right Lanczos vector v_j. To save inner products and also enhance numerical stability, the following formula for $\widetilde{t}_{j,k-p_c}$ should be used:

$$\widetilde{t}_{j,k-p_c} = \begin{cases} \widehat{w}_k^T v_j/\delta_j & \text{if } \; k - p_c \le 0, \\ t_{k-p_c,j}\,\delta_{k-p_c}/\delta_j & \text{otherwise.} \end{cases}$$

(15) The entries t_{jk} computed in this step are the potentially nonzero entries in the j-th row of $T_j^{(\mathrm{pr})}$ due to the vertical spikes caused by the deflation of $\widehat{v}_k$ vectors. Note that again formula (5.76) is employed.

(16) All the potentially nonzero elements in the j-th row and the j-th column of $T_j^{(\mathrm{pr})}$ have now been computed and they are added to the matrix $T_{j-1}^{(\mathrm{pr})}$ of the previous iteration $j-1$, to yield the current matrix $T_j^{(\mathrm{pr})}$. Here, we use the convention that entries t_{ik} that are not explicitly defined in Algorithm 5.35 are set to be zero. We remark that in the initial iterations, i.e., as long as $j \le m_c$, respectively $j \le p_c$, Algorithm 5.35 also produces entries t_{jk}, respectively $\widetilde{t}_{jk}$, with negative indices $k \le 0$. These entries arise from the bi-orthogonalization of the starting

vectors, and they are not part of the matrix $T_j^{(\mathrm{pr})}$. In particular, these entries are not needed when Algorithm 5.35 is used for eigenvalue computations. However, they are crucial when Algorithm 5.35 is applied to reduced-order modeling, as we will discuss in §5.9.4 below.

(17) For eigenvalue computations, one tests for convergence by computing the eigenvalues $\theta_i^{(j)}$, $i = 1, 2, \ldots, j$, of the $j \times j$ matrix $T_j^{(\mathrm{pr})}$. 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.

5.9.4 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., [85] for a recent survey. Such systems are characterized by matrix-valued transfer functions of the form

$$H(s) = C^T (I - sA)^{-1} B, \quad s \in \mathbb{C}, \tag{5.80}$$

Here, A is a square, in general non-Hermitian matrix, and

$$B = \begin{bmatrix} b_1 & b_2 & \cdots & b_m \end{bmatrix} \quad \text{and} \quad C = \begin{bmatrix} c_1 & c_2 & \cdots & c_p \end{bmatrix} \tag{5.81}$$

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 (5.81), 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 (5.80).

For reduced-order modeling, the entries t_{jk} and $\widetilde{t}_{jk}$ with negative indices $k \leq 0$ are also used. More precisely, in this case, Step (16) in Algorithm 5.35 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 $\rho = [\rho_{ik}]_{i=1,2,\ldots,j,\ k=1,2,\ldots,m}$.
> If $j \leq p_c$,
> set $\eta_{jk} = \widetilde{t}_{j,k-p}$ for all k with $j - p_c + p \leq k \leq p$,
> and set $\eta = [\eta_{ik}]_{i=1,2,\ldots,j,\ k=1,2,\ldots,p}$.

Here, again we use the convention that entries ρ_{ik} and η_{ik} that are not explicitly defined in Algorithm 5.35 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 the iteration j of Algorithm 5.35 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 $b_1, b_2, \ldots, b_m$ 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 5.35 for which $j = p_c$ is reached. The number p_1 is the value of p minus the number of left initial vectors $c_1, c_2, \ldots, c_p$ 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 $T_j^{(\mathrm{pr})}$, ρ, and η be the matrices generated after j iterations of Algorithm 5.35. These three matrices then define a j-th order reduced model of the original transfer function (5.80) as follows:

$$H_j(s) = \eta_j^T \left(I_j - sT_j^{(\mathrm{pr})}\right)^{-1} \rho_j, \quad s \in \mathbb{C}. \tag{5.82}$$

Here,

$$\rho_j = \begin{bmatrix} \rho \\ 0 \end{bmatrix} \in \mathbb{C}^{j \times m} \quad \text{and} \quad \eta_j = \begin{bmatrix} \eta \\ 0 \end{bmatrix} \in \mathbb{C}^{j \times p}$$

are the matrices ρ and η with $j - m_1$, respectively $j - p_1$, rows of zeros added. The reduced-order model (5.82) can be shown to be characterized as a certain matrix-Padé approximation of the original transfer function (5.80); see [85] and the references given there.

5.9.5 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 [132, 133]. 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, nor 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 that $m = p$ and that 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, underlies 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 5.35 first appeared in this form in [84]. A somewhat different version, which also includes a look-ahead procedure to remedy possible breakdowns, is described in [1].

The version of the band Lanczos method stated as Algorithm 5.35 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 pre-compute 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 5.35, one computes the matrix-matrix product

$$A \begin{bmatrix} v_j & \widehat{v}_{j+1} & \widehat{v}_{j+2} & \cdots & \widehat{v}_{j+m_c-1} \end{bmatrix}.$$

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

$$A\widehat{v}_{j+1}, A\widehat{v}_{j+2}, \ldots, A\widehat{v}_{j+m_c-1}. \tag{5.83}$$

In the following $m_c - 1$ iterations, we will need the vectors

$$Av_{j+1}, Av_{j+2}, \ldots, Av_{j+m'_c-1}. \tag{5.84}$$

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 §3.6.4, one can easily obtain the vectors (5.84) as suitable linear combinations of the pre-computed vectors (5.83). Similarly, at any stage of the algorithm, it is possible to pre-compute 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 $A^T W$ with a block W of p_c vectors. Indeed, instead of performing Step (11) in Algorithm 5.35, one computes the matrix-matrix product

$$A^T \left[\begin{array}{cccc} w_j & \widehat{w}_{j+1} & \widehat{w}_{j+2} & \cdots \quad \widehat{w}_{j+p_c-1} \end{array} \right]$$

and then proceeds analogously as for the vectors (5.83).

5.9.6 Software Availability

At the time of this writing, there is no software for the non-Hermitian band Lanczos method in public domain. However, Matlab implementations of the method, even with look-ahead, will be made available soon, and can then be obtained via `http://cm.bell-labs.com/who/freund`.

should be updated!!

5.10 Lanczos Method for Complex Symmetric Eigenproblems

R. Freund

In this section, we consider the complex symmetric eigenvalue problem,

$$Ax = \lambda x, \tag{5.85}$$

where $A = A^T \in \mathbb{C}^{n \times n}$.

5.10.1 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,

$$\lambda_1, \lambda_2, \ldots, \lambda_n \in \mathbb{C}, \tag{5.86}$$

there exists a complex symmetric $n \times n$ matrix A whose eigenvalues are just the prescribed numbers (5.86); see, e.g., [114, Theorem 4.4.9].

A complex symmetric matrix may not even be diagonalizable. For example, consider the complex symmetric matrix

$$A = \begin{bmatrix} 2\mathtt{i} & 1 \\ 1 & 0 \end{bmatrix}, \quad \text{where} \quad \mathtt{i} = \sqrt{-1}. \tag{5.87}$$

The only eigenvalue of this matrix is $\lambda = \mathtt{i}$, with algebraic multiplicity 2, but geometric multiplicity 1. In fact, the Jordan normal form of A is as follows:

$$Z^{-1}AZ = \begin{bmatrix} \mathtt{i} & 1 \\ 0 & \mathtt{i} \end{bmatrix}, \quad \text{where} \quad Z = \begin{bmatrix} \mathtt{i} & 1 \\ 1 & 0 \end{bmatrix}.$$

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., [114, 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

$$Z^T AZ = \Lambda = \operatorname{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n) \quad \text{and} \quad Z^T Z = I_n. \tag{5.88}$$

An matrix Z with n columns that satisfies $Z^T Z = I_n$ is called *complex orthogonal*. The complex orthogonality of Z in (5.88) reflects the complex symmetry of A.

We remark that the eigendecomposition (5.88) is the suitable adaptation of the corresponding decomposition for Hermitian matrices. Recall that for any matrix $A = A^H$, the eigenvector matrix Z can always be chosen to be unitary:

$$Z^H AZ = \Lambda = \operatorname{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n) \quad \text{and} \quad Z^H Z = I_n. \tag{5.89}$$

The unitarity of Z in (5.89) reflects the fact that A is Hermitian.

The reason why an eigendecomposition (5.88) does not always exist is that there are complex vectors z with

$$z^T z = 0, \quad \text{but} \quad z \neq 0. \tag{5.90}$$

Indeed, suppose A has an eigenvalue with a one-dimensional eigenspace and the vector z spanning that space satisfies (5.90). 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 (5.90), $z_k^T z_k = 0$, while the complex orthogonality condition, $Z^T Z = I_n$, in (5.88) would imply $z_k^T z_k = 1$. Note that for example (5.87), the vector $z = \begin{bmatrix} \mathtt{i} & 1 \end{bmatrix}^T$ spans the one-dimensional eigenspace associated with $\lambda = \mathtt{i}$ and it satisfies (5.90).

5.10.2 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 requirement of the general non-Hermitian Lanczos method described in §5.7. Indeed, while the non-Hermitian Lanczos method involves one matrix-product with A and one with A^H at each iteration, the complex symmetric Lanczos method only requires one matrix-product with A at each iteration.

After j iterations, the complex symmetric Lanczos method has generated j Lanczos vectors,

$$v_1, v_2, \ldots, v_j, \tag{5.91}$$

that span the j-th Krylov subspace $K^j(A, b)$ induced by the complex symmetric matrix A and any nonzero starting vector b. The vectors (5.91) are constructed to be complex orthogonal:

$$V_j^T V_j = I_j, \quad \text{where} \quad V_j = \begin{bmatrix} v_1 & v_2 & \cdots & v_j \end{bmatrix}. \tag{5.92}$$

Note that, in view of the eigendecomposition (5.88) of diagonalizable complex symmetric matrices A, the complex orthogonality (5.92) of the Lanczos vectors is natural.

The complex symmetric Lanczos algorithm computes the vectors (5.91) by means of three-term recurrences that can be summarized as follows:

$$AV_j = V_j T_j + \begin{bmatrix} 0 & \cdots & 0 & \widehat{v}_{j+1} \end{bmatrix}. \tag{5.93}$$

Here,

$$T_j = T_j^T = \begin{bmatrix} \alpha_1 & \beta_2 & & \\ \beta_2 & \alpha_2 & \ddots & \\ & \ddots & \ddots & \beta_j \\ & & \beta_j & \alpha_j \end{bmatrix} \tag{5.94}$$

is a complex symmetric tridiagonal matrix whose entries are the coefficients of the three-term recurrences. The vector $\widehat{v}_{j+1}$ is the candidate for the next Lanczos vector, v_{j+1}. It is constructed so that the orthogonality condition

$$V_j^T \widehat{v}_{j+1} = 0 \tag{5.95}$$

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

$$\widehat{v}_{j+1}^T \widehat{v}_{j+1} = 0, \quad \text{but} \quad \widehat{v}_{j+1} \neq 0. \tag{5.96}$$

If (5.96) occurs, then a next vector v_{j+1} cannot be obtained by simply normalizing $\widehat{v}_{j+1}$, as it would require division by zero. Therefore, (5.96) 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 (5.96) is encountered.

After j iterations of the complex symmetric Lanczos algorithm, approximate eigensolutions for the complex symmetric eigenvalue problem (5.85) are obtained by computing eigensolutions of T_j,

$$T_j z_i^{(j)} = \theta_i^{(j)} z_i^{(j)}, \quad i = 1, 2, \ldots, j. \tag{5.97}$$

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.,

$$T_j = V_j^T A V_j. \tag{5.98}$$

Indeed, the relation follows by multiplying (5.93) from the left by V_j^T and by using the orthogonality relations (5.92) and (5.95). Of course, in the complex symmetric Lanczos algorithm, the matrix T_j is not computed via the relation (5.98). Instead, the symmetric tridiagonal structure in the definition (5.94) is exploited and only the diagonal and subdiagonal entries of T_j are explicitly generated.

Editors' comment: it should be pointed out that V_j is complex orthogonal, but not unitary, which may have effects for the numerical stability.

5.10.3 The Algorithm

A complete statement of the complex symmetric Lanczos algorithm (without look-ahead) is as follows.

ALGORITHM 5.36: **Complex symmetric Lanczos**

(1) *Set* $\widehat{v}_1 = b$ *and* $v_0 = 0$.
(2) *For* $j = 1, 2, \ldots,$ *until convergence or* $\beta_j = 0$ *do*:
(3) *If* $\widehat{v}_j = 0$, *set* $j = j - 1$ *and stop*:
All eigenvalues of T_j *are also eigenvalues of* A.
(4) *Compute* $\beta_j = \sqrt{\widehat{v}_j^T \widehat{v}_j}$.
(5) *If* $\beta_j = 0$, *set* $j = j - 1$ *and stop* : *breakdown*.
(In this case, look-ahead would be needed to continue.)
(6) *Normalize* $v_j = \widehat{v}_j / \beta_j$.
(7) *Compute* $\widehat{v}_{j+1} = A v_j$.
(8) *Subtract* $\widehat{v}_{j+1} = \widehat{v}_{j+1} - v_{j-1}\beta_j$.
(9) *Compute* $\alpha_j = v_j^T \widehat{v}_{j+1}$.
(10) *Subtract* $\widehat{v}_{j+1} = \widehat{v}_{j+1} - v_j \alpha_j$.
(11) *Test for convergence*.

Next, we comment on a few of the steps of Algorithm 5.36.

(3) If $\widehat{v}_j = 0$ 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., [86, 88].

(11) To test for convergence, the eigenvalues $\theta_i^{(j)}$, $i = 1, 2, \ldots, j$, of the complex symmetric matrix tridiagonal 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.

5.10.4 Solving the reduced eigenvalue problems

Recall that Algorithm 5.36 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 non-Hermitian eigenvalue problems. Cullum and Willoughby [35, 36, 37] 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 [37] 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 (5.98), 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 (5.97), one could solve the equivalent complex symmetric generalized eigenvalue problem

$$D_j T_j z_i^{(j)} = \theta_i^{(j)} D_j z_i^{(j)}, \quad i = 1, 2, \ldots, j. \tag{5.99}$$

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 (5.99).

5.10.5 Software Availability

A Fortran implementation of Algorithm 5.36, even with the option to perform look-ahead if necessary, is included in the software package QMRPACK [88]. 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 [86]. QMRPACK also contains implementations of the complex symmetric QMR iterative solver based on Algorithm 5.36. QMRPACK is available from *netlib* or directly from

`http://cm.bell-labs.com/who/freund` .

Algorithm 5.36 can also be used for the iterative solution of complex symmetric linear systems. In [83], a QMR iterative solver based on Algorithm 5.36 is described. Some further theoretical results on complex symmetric matrices can also be found in [83].

5.10.6 Notes and References

A detailed discussion of basic properties of complex symmetric matrices can be found in [30] and [114, 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 [3] .

5.11 Jacobi-Davidson methods

G. Sleijpen and H. van der Vorst

5.11.1 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 §3.7 we have explained the method in detail, and the generalization to the non-Hermitian case for the basic algorithm, described in Alg. 3.13, is quite straight forward. 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 part (3) has to be replaced by:

   for $i = 1, .., j-1$
   $\quad m_{i,j} = v_i^* w_j$, $m_{j,i} = v_j^* w_i$
   $m_{j,j} = v_j^* w_j$

2. In (4) a routine has to be selected for the non-Hermitian dense matrix M

If the correction equation (in step (7) of the algorithm) is solved exactly, then the approximate eigenvalues have quadratic convergence towards the eigenvalues of A.

5.11.2 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 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 $AQ_k = Q_k R_k$, 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 Alg. 5.37. This scheme includes the possibility for restart when the dimension of the current subspace exceeds j_{max}.
The scheme computes k_{max} eigenvalues close to a target τ in the complex plane. Here we have to be necessarily unprecise, since the eigenvalues of a general non-Hermitian matrix are not ordered in the complex plane. Which eigenvalues are selected as close eigenvalues depends amongst others on the angles of the corresponding eigenvectors. Usually, the selection of desired eigenvalues 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 §5.11.3.

ALGORITHM 5.37: **Jacobi-Davidson for k_{max} exterior eigenvalues**

(1) $t = v_0$, $k = 0$, $j = 0$; $Q = []$, $R = []$
while $k < k_{max}$
(2) **for** $i = 1, .., j$
$t = t - (v_i^* t)v_i$
$j = j + 1$, $v_j = t/\|t\|_2$, $w_j = Av_j$
(3) **for** $i = 1, .., j - 1$
$m_{i,j} = v_i^* w_j$, $m_{j,i} = v_j^* w_i$
$m_{j,j} = v_j^* w_j$
(4) *Make a Schur decomposition* $M = STS^*$,
S unitary and $T = (t_{i,j})$ upper triangular,
such that: $|t_{i,i} - \tau| \leq |t_{i+1,i+1} - \tau|$
$\theta = t_{1,1}$, $u = Vs_1$, $\widetilde{w} = Ws_1$, $r = \widetilde{w} - \theta u$
$\widehat{s} = Q^* r$, $\widehat{r} = r - Q\widehat{s}$
(5) **while** $\|\widehat{r}\|_2 \leq \epsilon$
$R = \begin{pmatrix} R & \widehat{s} \\ 0 & \theta \end{pmatrix}$, $Q = [Q, u]$, $k = k + 1$
if $k = k_{max}$ **then** *STOP*
$j = j - 1$
(6) **for** $i = 1, .., j$
$v_i = Vs_{i+1}$, $w_i = Ws_{i+1}$, $s_i = e_i$
$M =$ *lower j by j block of T*
$\theta = m_{1,1}$, $u = v_1$, $r = w_1 - \theta u$, $\widehat{s} = Q^* r$, $\widehat{r} = r - Q\widehat{s}$
(7) **if** $j \geq j_{max}$ **then**
for $i = 2, .., j_{min}$
$v_i = Vs_i$, $w_i = Ws_i$
$M =$ *leading j_{min} by j_{min} block of T*
$v_1 = u$, $w_1 = \widetilde{w}$, $j = j_{min}$
(8) $\widetilde{Q} = [Q, u]$, *Solve t ($\perp \widetilde{Q}$) (approximately) from:*
$(I - \widetilde{Q}\widetilde{Q}^*)(A - \theta I)(I - \widetilde{Q}\widetilde{Q}^*)t = -\widehat{r}$

We will now discuss the components of the algorithm represented in Alg. 5.37.

Input: A device to compute Av for arbitrary v, 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 j_{max} denotes the maximum dimension of the search subspace. If it is exceeded then a restart takes place with a subspace of specified dimension j_{min}.

Output: k_{max} Schur pairs for which the eigenvalues are close to τ, and the corresponding upper triangular Schur matrix R of dimension k_{max}. The computed Schur forms satisfy $\|Aq_j - Q_j Re_j\|_2 \leq \epsilon$.

(1) Initialization phase.

(2) 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 given in Alg. 3.16.

If the angle between the new vector t and the search subspace is very small, then the resulting vector v_j has no significance and the method stagnates. A random vector t helps to overcome this stagnation. A more sophisticated strategy can be found in [98].

(3) We compute the last column and row of the dense matrix M (of order j).

(4) The Schur reduction for the j by j matrix M_j 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 §5.11.3.

In each step, the Schur form has to be sorted such that $t_{1,1}$ is closest to τ. Only if $j \geq j_{max}$ the sorting of the Schur form has to be be such that all of the j_{max} leading diagonal elements of T_j are closest to τ. For ease of presentation we sorted all diagonal elements here.

For a template an algorithm of a Schur form, see [250, 251] and [81, Chap. 6B].

(5) 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 relative inaccuracies (in norm) in the Schur vectors of $\mathcal{O}(\sqrt{\epsilon})$ (provided that the concerned eigenvalue is well separated from the others and the left and right eigenvector have a small angle).

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.

(6) After convergence of a Schur pair, we continue the search for a next pair, with the unconverged Schur vectors as a basis for the initial search space.

(7) We restart as soon as the dimension of the search space for the current Schur vector exceeds j_{max}. The process is restarted with the subspace spanned by the j_{min} Ritz vectors corresponding to the Ritz values closest to the target value τ.

(8) We have collected the locked (converged) Schur vectors in Q, and the matrix $\widetilde{Q}$ 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 (3.51). The new correction v 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. Because of the occurrence of $\widetilde{Q}$ one has to be careful with the usage 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 Alg. 3.18 for a Template. Make sure that the starting vector for an

iterative solver satisfies the orthogonality constraints. Note that significant savings can be made in Alg. 3.18 if P_j is kept the same for a (few) Jacobi-Davidson iterations. In that case columns of E 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 [47], here also works well: 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 [178, 82].

For a full theoretical background of this method, as well as details on the deflation technique with Schur vectors, see [82].

5.11.3 Interior Eigenvalues

5.11.3.1 The usage of Harmonic Ritz values for interior eigenvalues

For restart purposes, specially if one is heading for interior eigenvalues, the Harmonic Ritz vectors have been advocated for symmetric matrices [175], see also §3.7.4.

The concept of Harmonic Ritz values [187] is easily generalized for unsymmetric matrices [227]. As we have seen, the Jacobi-Davidson methods generate basis vectors $v_1, \ldots, v_k$, for a subspace $\mathcal{V}_k$. For the projection of A onto this subspace we have to compute the vectors $w_j = Av_j$. 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 $(\widetilde{\theta}^{(k)}, \widetilde{u}^{(k)})$ satisfies

$$A\widetilde{u}^{(k)} - \theta^{(k)}\widetilde{u}^{(k)} \perp \mathcal{W}_k = \mathrm{span}(w_1, \ldots, w_k), \tag{5.100}$$

for $\widetilde{u}^{(k)} \in V_k$ and $\widetilde{u}^{(k)} \neq 0$. This implies that the harmonic Ritz values are the eigenvalues of the pencil $(W_k^* A V_k, W_k^* V_k)$:

$$W_k^* A V_k s - \theta W_k^* V_k s = 0$$

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 $\mathcal{V}_k$ 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_k and W_k to orthonormal: W_k is such that $(A - \tau I)V_k = W_k H_k$ where H_k is k by k upper triangular.
In the resulting scheme we compute a (partial) Schur decomposition rather than a partial eigendecomposition. That is, we wish to compute vectors $q_1, \ldots, q_k$, such that $AQ_k = Q_k R_k$, 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.

5.11.3.2 A template for Jacobi-Davidson for interior eigenvalues

An algorithm for Jacobi-Davidson, with partial reduction to Schur form, and based on Harmonic Ritz values and vectors, is given in Alg. 5.38. The algorithm includes restart and deflation techniques. It can be used for the computation of a number of eigenvalues close to τ.

ALGORITHM 5.38: **Jacobi-Davidson for k_{max} interior eigenvalues close to τ**

(1) $t = v_0$, $k = 0$, $j = 0$; $Q = [\,]$, $R = [\,]$
while $k < k_{max}$
(2) **for** $i = 1, .., j$
$t = t - (v_i^* t)v_i$
$j = j + 1$, $v_j = t/\|t\|_2$, $v_j^A = Av_j - \tau v_j$, $w = v_j^A$
for $i = 1, .., k$
$w = w - (q_i^* w)q_i$
for $i = 1, .., j-1$
$h_{i,j} = w_i^* w$, $w = w - h_{i,j}w_i$
$h_{j,j} = \|w\|_2$, $w_j = w/h_{j,j}$
(3) **for** $i = 1, .., j-1$
$m_{i,j} = w_i^* v_j$, $m_{j,i} = w_j^* v_i$
$m_{j,j} = w_j^* v_j$
(4) *Make a QZ decomposition* $HU^R = U^L S$, $MU^R = U^L T$,
U^R, U^L *unitary and* $S = (s_{i,j})$, $T = (t_{i,j})$ *upper triangular,*
such that: $|s_{i,i}/t_{i,i}| \leq |s_{i+1,i+1}/t_{i+1,i+1}|$
$u = Vu_1^R$, $\widetilde{u} = V^A u_1^R$, $\theta_r = \overline{t}_{1,1}s_{1,1}$,
$r = \widetilde{u} - \theta_r u$, $\widehat{s} = Q^* r$, $\widehat{r} = r - Q\widehat{s}$
(5) **while** $\|\widehat{r}\|_2 \leq \epsilon$
$R = \begin{pmatrix} R & \widehat{s} \\ 0 & \theta_r + \tau \end{pmatrix}$, $Q = [Q, u]$, $k = k + 1$
if $k = k_{max}$ **then** *STOP*
(6) $j = j - 1$
for $i = 1, .., j$
$v_i = Vu_{i+1}^R$, $v_i^A = V^A u_{i+1}^R$, $w_i = Wu_{i+1}^L$, $u_i^R = u_i^L = e_i$
H, M *is the lower* j *by* j *block of* S, T, *resp.*
$u = v_1$, $\widetilde{u} = v_1^A$, $\theta_r = \overline{m}_{1,1}h_{1,1}$
$r = \widetilde{u} - \theta_r u$, $\widehat{s} = Q^* r$, $\widehat{r} = r - Q\widehat{s}$
(7) **if** $j \geq j_{max}$ **then**
for $i = 2, .., j_{min}$
$v_i = Vu_i^R$, $v_i^A = V^A u_i^R$, $w_i = Wu_i^L$
H, M *is the leading* j_{min} *by* j_{min} *block of* S, T, *resp.*
$v_1 = u$, $v_1^A = \widetilde{u}$, $w_1 = Wu_1^L$, $j = j_{min}$
(8) $\theta = \theta_r + \tau$, $\widetilde{Q} = [Q, u]$
Solve $t\ (\perp \widetilde{Q})$ *(approximately) from:*
$(I - \widetilde{Q}\widetilde{Q}^*)(A - \theta I)(I - \widetilde{Q}\widetilde{Q}^*)t = -\widehat{r}$

We will comment on some parts of the algorithm in view of our discussions in previous subsections.

Input: A device to compute Av for arbitrary v, 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 j_{max} denotes the maximum dimension of the search subspace. If it is exceeded then a restart takes place with a subspace of specified dimension j_{min}.

Output: k_{max} Schur pairs for which the eigenvalues are close to τ, and the corresponding upper triangular Schur matrix R of dimension k_{max}. The computed Schur forms satisfy $\|Aq_j - Q_j R e_j\|_2 \leq \epsilon$.

(1) Initialization phase.

(2) 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 the matrix $V_j^A = AV_j - \tau V_j$; v_j^A is the expansion vector for this matrix. The expansion vector for W_j is obtained by making v_j^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 Alg. 3.16.

(3) The values $m_{i,j}$ represent elements of the square j by j matrix $M_j \equiv W_j^* V_j$. The values $h_{i,j}$ represent elements of the j by j upper triangular H_j for which $AV_j - \tau V_j = W_j H_j + Q_k F$. Here $F = Q_k^*(AV_j - \tau V_j)$.

(Note that $V_j^A = AV_j - \tau V_j$ can be reconstructed from W_j, H_j, Q_k and F. In particular, r can be computed from these quantities. Instead of storing the n-dimensional matrix V_j, it suffices to store the k by j matrix F (of elements $q_i^* w$, computed in (2)). This approach saves memory space. However, for avoiding instabilities, the deflation procedure needs special attention.)

At this point the generalized Schur form of the matrix pencil (H_j, M_j) has to be be computed: $H_j U_j^R = U_j^L S_j$ and $M_j U_j^R = U_j^L T_j$, where U_j^R and U_j^L are unitary and S_j and T_j are upper triangular. This can be done with a suitable routine for dense matrix pencils dense matrices from LAPACK.

In each step, the Schur form has to be sorted such that $|s_{1,1}/t_{1,1}|$ is smallest. Only if $j \geq j_{max}$ the sorting of the Schur form has to be be such that all of the j_{max} leading diagonal elements of S_j and T_j represent the j_{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 [250, 251] and [81, Chap. 6B].
The value of θ needs some attention. We have chosen to compute the Rayleigh quotient θ_r (instead of the harmonic Ritz value) corresponding to the harmonic Ritz vector u (see [228]). The Rayleigh quotient follows from the requirement that $(A - \tau I)u - \theta_r u \perp u$ instead of $\perp W$: then $r \perp u$.

(5) 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 relative inaccuracies (in norm) in the eigenvectors of $\mathcal{O}(\sqrt{\epsilon})$ (provided that the concerned eigenvalue is well separated from the others and the left and right eigenvector have a small angle).

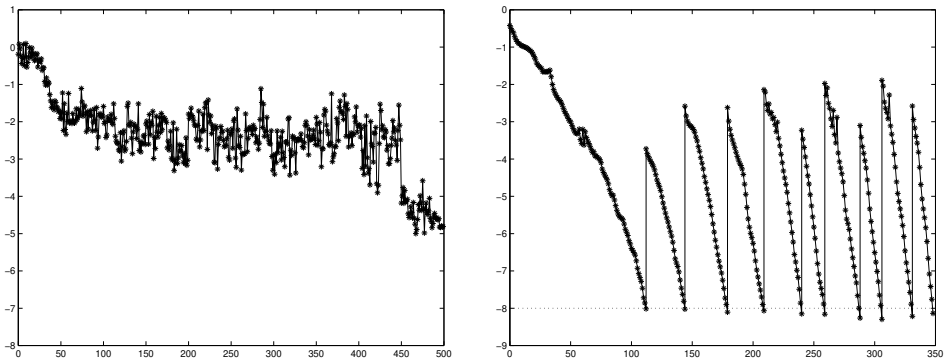


FIGURE 5.1: *Jacobi-Davidson for exterior eigenvalues (left figure) and interior eigenvalues (right figure).*

(6) This is a restart after convergence of a Schur pair.

(7) At this point we have a restart when the dimension of the subspace exceeds j_{max}. After a restart the Jacobi-Davidson iterations are resumed with a subspace of dimension j_{min}.

(8) The deflation with converged eigenvectors is represented by the factors with Q. The matrix Q has the converged eigenvectors as its columns. If a left preconditioner P_j 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 Alg. 3.18.

5.11.3.3 Numerical example

We discuss the results for a small example that can be repeated easily..

The unsymmetric matrix A is of dimension $n = 100$ and it is tridiagonal. The diagonal entries are $a_{i,i} = -2$, the codiagonal entries are $a_{i,i-1} = 1$, $a_{i,i+1} = 1.2$.

We use Alg. 5.37 and Alg. 5.38, 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 of Alg. 5.38, 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 seed = 0 in Matlab. The tolerance is $\epsilon = 10^{-8}$. The subspace dimension parameters are $j_{min} = 10$, $j_{max} = 15$. The correction equations have been approximately solved with 5 steps of GMRES.

We graphically show in Fig. 5.1 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 good approximation, and the iteration is continued with deflation for the next eigenvalue. The left picture represents the results obtained with Alg. 5.37, and we see that there is no convergence detected within 500 Jacobi-Davidson steps. In

the right picture we see the results for Alg. 5.38, and now 10 eigenvalues have been discovered within 350 iterations.

5.11.4 Software Availability

Matlab versions of the algorithms are available from either
`http://www.math.uu.nl/people/sleijpen`
or
`http://www.math.uu.nl/people/vorst`

Fortran implementations of the Jacobi-Davidson method are available via
`http://www.math.uu.nl/people/bomhof`
The Fortran software is of experimental type and does not reflect all insights and possibilities described in this section.

CHAPTER 6

Generalized Non-Hermitian Eigenproblem

6.1 Introduction

6.2 Transformation Methods

6.3 Single and Multiple Vector Iterations Methods

6.4 Arnoldi Method

6.5 Rational Krylov Subspace Method

6.6 Lanczos Method

6.7 Indefinite Symmetric Lanczos Method

6.8 Jacobi-Davidson Method

CHAPTER 7

Singular Value Decomposition and its Generalization

CHAPTER 8

More on Generalized Eigenvalue Problems

8.1 Introduction

Many eigenvalue problems arising in practical applications may not be the exact forms as discussed in the previous chapters. In this chapter, we will discuss some of those important eigenvalue problems.

In §8.2, basic theory and numerical treatment and software for the generalized eigenvalue problem

$$Ax = \lambda Bx$$

of a singular matrix pair $\{A, B\}$ is presented. Here, the matrix pair $\{A, B\}$ could be $m \times n$ non-square, or $\det(A - \lambda B) \equiv 0$ for all λ. Although the generalized eigenvalue problems $Ax = \lambda Bx$ has been treated extensively in Chapters 4 and 6, it is assumed that the matrix pair $\{A, B\}$ is square and generally regular.

§8.3 focuses on the quadratic eigenvalue problem

$$(\lambda^2 M + \lambda C + K)x = 0,$$

where the matrices M, C and K are given square matrices. The scalars λ and the corresponding non-trivial vectors x are the desired eigenvalues and eigenvalues. The quadratic eigenvalue problem is a natural and important generalization of the generalized eigenvalue problems discussed in Chapter 4 and 6. Numerical techniques of the quadratic eigenvalue problem is outlined in §8.3. These techniques are closely related to the numerical techniques presented in previous chapters.

§8.4 increases the scope of the algebraic eigenvalue problem by focusing on the geometrical properties of the eigenspaces. It presents a template that solves variational problems defined on sets of subspaces. Specifically, it provides a means 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. In §8.4, it is shown that many eigenvalue problems and their modifcations, such as the procrustes problem, nearest Jordan structure and trace minimization, can be solved using this template.

8.2 Singular 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 chapters is the appearance of infinite eigenvalues whenever B is singular. The standard method for solving dense generalized eigenvalue problems is the QZ algorithm, see section ??. However, there is more to the story. For example, if A and B have a common nullspace, 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 problems in a reliable way. It is necessary to apply a regularization technique by allowing a deflation criterion for range/nullspace 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 form (GUPTRI is an acronym for Generalized UPper TRIangular) [52, 53]:

$$P^*(A - \lambda B)Q = \begin{bmatrix} A_r - \lambda B_r & * & * \\ 0 & A_{reg} - \lambda B_{reg} & * \\ 0 & 0 & A_l - \lambda B_l \end{bmatrix}, \tag{8.1}$$

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 (8.1) 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.

8.2.1 Regular versus singular problems

Let us start by considering the following 2×2 generalized eigenvalue problem $Ax = \lambda Bx$, where

$$A = \begin{bmatrix} \alpha_1 & 0 \\ 0 & \alpha_2 \end{bmatrix}, \qquad B = \begin{bmatrix} \beta_1 & 0 \\ 0 & \beta_2 \end{bmatrix}.$$

The eigenvalues of $A - \lambda B$ are the pairs $(\alpha_1,\ \beta_1)$ and $(\alpha_2,\ \beta_2)$ with the associated eigenvectors $x_1 = (1 \quad 0)^T$ and $x_2 = (0 \quad 1)^T$, respectively. If β_i is non-zero, 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 nullspace. We say that x_2 is an eigenvector for an indeterminate eigenvalue 0/0. Note that the common nullspace 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 regular matrix pencil. The n eigenvalues of a regular pencil are points in the extended complex plane $\mathbf{C} \cup \infty$. 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 $(\alpha, \beta) \in \mathbf{C}^2$. 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 $\mathbf{C}^2$.

If $\det(A - \lambda B)$ (and $\det(\beta A - \alpha B)$) is identical 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 Section 8.2.4. Moreover, rectangular matrix pairs are singular and the corresponding $A - \lambda B$ is a singular pencil.

8.2.2 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 [95]. Suppose $A, B \in \mathbf{C}^{m \times n}$. Then there exist nonsingular $P \in \mathbf{C}^{m \times m}$ and $Q \in \mathbf{C}^{n \times n}$ such that

$$P^{-1}(A - \lambda B)Q = \widetilde{A} - \lambda \widetilde{B} \equiv \mathrm{diag}(A_1 - \lambda B_1, \ldots, A_b - \lambda B_b), \tag{8.2}$$

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 $\widetilde{A} - \lambda \widetilde{B}$:

$$P = [P_1, \ldots, P_b] \quad \text{and} \quad Q = [Q_1, \ldots, Q_b],$$

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

$$J_j(\alpha) \equiv \begin{bmatrix} \alpha-\lambda & 1 & & \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ & & & \alpha-\lambda \end{bmatrix} \quad \text{and} \quad N_j \equiv \begin{bmatrix} 1 & -\lambda & & \\ & \ddots & \ddots & \\ & & \ddots & -\lambda \\ & & & 1 \end{bmatrix}. \tag{8.3}$$

$J_j(\alpha)$ 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

$$\{e_j, J_j(\alpha)e_j, J_j(\alpha)^2 e_j, \ldots, J_j(\alpha)^{j-1}e_j\}$$

span the range. From the cyclicity of the transformation we get the vector chain relations

$$(J_j(\alpha) - (\alpha-\lambda)I)e_{k+1} = e_k \quad \text{for} \quad 1 \le k \le j-1.$$

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

$$L_j \equiv \begin{bmatrix} -\lambda & 1 & & \\ & \ddots & \ddots & \\ & & -\lambda & 1 \end{bmatrix} \quad \text{and} \quad L_j^T \equiv \begin{bmatrix} -\lambda & & & \\ 1 & \ddots & & \\ & \ddots & -\lambda & \\ & & 1 & \end{bmatrix}. \tag{8.4}$$

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 nullspace, $[1, \lambda, \ldots, \lambda^j]^T$, 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 nullspace 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 (8.1). 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 $A_{reg} - \lambda B_{reg}$.

8.2.3 Generic and non-generic 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

$$\mathrm{diag}(L_\ell, \ldots, L_\ell, L_{\ell+1}, \ldots, L_{\ell+1}), \tag{8.5}$$

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) [248, 49]. The same statement holds for $d = m - n > 0$ if we replace $L_\ell, L_{\ell+1}$ in (8.4) 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 [254]:

$$\mathrm{diag}(L_j, L_{n-j-1}^T), \quad j = 0, \ldots, n-1. \tag{8.6}$$

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 *non-generic* (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 [249, 51].

8.2.4 Ill-conditioning of eigenproblems

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:

$$A_1 - \lambda B_1 = \left[\begin{array}{c|c|c} 1 & 0 & 0 \\ \hline 0 & 0 & 0 \\ \hline 0 & 0 & -\lambda \end{array}\right], \qquad A_2 - \lambda B_2 = \left[\begin{array}{cc|c} 1 & -\lambda & 0 \\ \hline 0 & 0 & 1 \\ 0 & 0 & -\lambda \end{array}\right].$$

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: $A_1 - \lambda B_1 = \mathrm{diag}(N_1, L_0, L_0^T, J_1(0))$, and $A_2 - \lambda B_2 = \mathrm{diag}(L_1, L_1^T)$. From top to bottom, the diagonal blocks of $A_1 - \lambda B_1$ correspond to N_1, $\mathrm{diag}(L_0, L_0^T)$ 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 (α, β) = (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

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

which has one finite eigenvalue 1 and one indeterminate eigenvalue 0/0 (corresponding to a singular part $\mathrm{diag}(L_0, L_0^T)$). To see that neither the eigenvalue 1 nor the singular part is well determined by the data, consider the slightly perturbed problem

$$A' = \begin{bmatrix} 1 & \epsilon_1 \\ \epsilon_2 & 0 \end{bmatrix}, \qquad B' = \begin{bmatrix} 1 & \epsilon_3 \\ \epsilon_4 & 0 \end{bmatrix},$$

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 structure characteristics. For this example the regularization requires that the perturbed pencil also has 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 nullspaces. Let $x \neq 0$ belong to the intersection of the nullspaces 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 1-dimensional column (and row) nullspace spanned by $x = e_2$. The dimensions of the intersecting column and row nullspaces, respectively, are the number of L_0 and L_0^T blocks, respectively, in the pencils' KCF. Notice that the intersection of nullspaces 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 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 nullspace. For example, consider the 16×16

matrices

$$A'' = \begin{bmatrix} 0.1 & 1 & & & \\ & 0.1 & 1 & & \\ & & \ddots & \ddots & \\ & & & 0.1 & 1 \\ & & & & 0.1 \end{bmatrix} \qquad \text{and} \qquad B'' = A''.$$

Then (A'', B'') has all eigenvalues at one. 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 $\|A''x\|_2 = \|B''x\|_2 \approx \mathcal{O}(10^{-16})$. 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.

8.2.5 Generalized Schur-staircase form

In general, we cannot guarantee to compute the KCF of a pencil stably, 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 [52, 53]:

$$P^*(A - \lambda B)Q = \begin{bmatrix} A_r - \lambda B_r & * & * \\ 0 & A_{reg} - \lambda B_{reg} & * \\ 0 & 0 & A_l - \lambda B_l \end{bmatrix}, \tag{8.7}$$

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 (8.7). If $A - \lambda B$ is regular $A_r - \lambda B_r$ and $A_l - \lambda B_l$ are not present in (8.7) 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 $A_{reg} - \lambda B_{reg}$:

$$A_{reg} = \begin{bmatrix} A_z & * & * \\ 0 & A_f & * \\ 0 & 0 & A_i \end{bmatrix}, \qquad B_{reg} = \begin{bmatrix} B_z & * & * \\ 0 & B_f & * \\ 0 & 0 & B_i \end{bmatrix}. \tag{8.8}$$

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-staircase form (see Section 8.2.6) 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* [252, 52]. 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

$$P^*(A - \lambda B)Q = \begin{bmatrix} A_{11} - \lambda B_{11} & A_{12} - \lambda B_{12} \\ 0 & A_{22} - \lambda B_{22} \end{bmatrix}, \tag{8.9}$$

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 appear in several applications, e.g., computing controllable and observable subspaces of generalized linear systems, which represent ill-posed problems in control theory [249, 51].

8.2.6 GUPTRI algorithm

The algorithm for computing the GUPTRI form is based on two reductions (*staircase forms*) [52]. The first is the RZ-staircase form (RZ for Right-Zero) 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 $(= 1, 2, \ldots)$, n_k = dimension of the column nullspace of $A^{(k)}$ and $n_k - r_k$ = dimension of the intersecting column nullspace 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:

$$A_{rz} - \lambda B_{rz} = \begin{bmatrix} -\lambda B_{11} & A_{12} - \lambda B_{12} & A_{13} - \lambda B_{13} \\ 0 & -\lambda B_{22} & A_{23} - \lambda B_{23} \\ 0 & 0 & -\lambda B_{33} \end{bmatrix}$$

$$= \left[\begin{array}{cccc|cc|cc} 0 & 0 & 0 & 0 & \mathbf{x} & \mathbf{x} & x & x \\ 0 & 0 & 0 & 0 & \mathbf{x} & \mathbf{x} & x & x \\ 0 & 0 & 0 & 0 & \mathbf{x} & \mathbf{x} & x & x \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{y} & \mathbf{y} \\ 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{y} & \mathbf{y} \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right] - \lambda \left[\begin{array}{cccc|cc|cc} 0 & \mathbf{x} & \mathbf{x} & \mathbf{x} & x & x & x & x \\ 0 & 0 & \mathbf{x} & \mathbf{x} & x & x & x & x \\ 0 & 0 & 0 & \mathbf{x} & x & x & x & x \\ \hline 0 & 0 & 0 & 0 & \mathbf{y} & \mathbf{y} & y & y \\ 0 & 0 & 0 & 0 & 0 & \mathbf{y} & y & y \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{z} \end{array}\right].$$

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 font or math font. The super-diagonal 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:

$$\begin{aligned} n_1 &= 4 = \dim \mathcal{N}(A^{(1)}), \quad & r_1 &= 3 = n_1 - \dim \mathcal{N}(A^{(1)} \cap B^{(1)}), \\ n_2 &= 2 = \dim \mathcal{N}(A^{(2)}), \quad & r_2 &= 2 = n_2 - \dim \mathcal{N}(A^{(2)} \cap B^{(2)}), \\ n_3 &= 2 = \dim \mathcal{N}(A^{(3)}), \quad & r_3 &= 1 = n_3 - \dim \mathcal{N}(A^{(3)} \cap B^{(3)}), \\ n_4 &= 0 = \dim \mathcal{N}(A^{(4)}). \end{aligned}$$

Now, we can conclude that the KCF of $A_{rz} - \lambda B_{rz}$ is $\mathrm{diag}(L_0, L_2, J_1(0), J_3(0))$.

In general, let A and B be complex $m \times n$ matrices. Then it can be shown that there exist unitary matrices $U \in \mathbf{C}^{m \times m}$ and $V \in \mathbf{C}^{n \times n}$ such that the matrix pencil $A - \lambda B$ is transformed to the following so-called *RZ-staircase form* [52, 53]:

$$U^*(A - \lambda B)V = \begin{bmatrix} A_{rz} & A_{12} \\ 0 & A_{rest} \end{bmatrix} - \lambda \begin{bmatrix} B_{rz} & B_{12} \\ 0 & B_{rest} \end{bmatrix} \tag{8.10}$$

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 $A - \lambda B$:

$$\begin{aligned} A_{rz} &= \begin{bmatrix} 0 & A_{12} & * & * & * & * \\ 0 & 0 & A_{23} & * & * & * \\ 0 & 0 & 0 & \ddots & * & * \\ \vdots & \vdots & \vdots & \ddots & A_{j-2j-1} & * \\ 0 & \cdots & 0 & 0 & 0 & A_{j-1j} \end{bmatrix}, \\ B_{rz} &= \begin{bmatrix} B_{11} & B_{12} & * & * & * & * \\ 0 & B_{22} & B_{23} & * & * & * \\ 0 & 0 & B_{33} & \ddots & * & * \\ \vdots & \vdots & \vdots & \ddots & B_{j-2j-1} & * \\ 0 & \cdots & 0 & 0 & B_{j-1j-1} & B_{j-1j} \end{bmatrix}. \end{aligned} \tag{8.11}$$

The subblocks in (8.10) and (8.11) have the following properties:

- B_{ii} is $r_i \times n_i$ and $A_{i-1i}, B_{i-1,i}$ 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 $i = 1, \ldots, j-1$.
- A_{i-1i} has *full column rank* n_i for $i = 2, \ldots, j$.
- 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 (8.10) 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 (8.10). In all three cases, it is possible that the j-th block column of $A_{rz} - \lambda B_{rz}$ (8.11) does not appear; if it does, A_{j-1j} 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-staircase form (LI for Left-Infinity) 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 south-east 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 nullspaces 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:

$$A_r - \lambda B_r = \left[\begin{array}{cc|c|c} 0 & 0 & \mathbf{x} & x \\ \hline 0 & 0 & 0 & \mathbf{y} \end{array}\right] - \lambda \left[\begin{array}{cc|c|c} 0 & 0 & \mathbf{x} & x \\ \hline 0 & 0 & \mathbf{y} & \mathbf{z} \end{array}\right]$$

and

$$A_z - \lambda B_z = \left[\begin{array}{cc|c|c} 0 & 0 & \mathbf{x} & x \\ 0 & 0 & \mathbf{x} & x \\ \hline 0 & 0 & 0 & \mathbf{y} \\ \hline 0 & 0 & 0 & 0 \end{array}\right] - \lambda \left[\begin{array}{cc|c|c} \mathbf{x} & \mathbf{x} & x & x \\ 0 & \mathbf{x} & x & x \\ \hline 0 & 0 & \mathbf{y} & y \\ \hline 0 & 0 & 0 & \mathbf{z} \end{array}\right].$$

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 (8.7) and (8.8) 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.

8.2.6.1 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:

$$\mathrm{diag}(L_0, L_2, J_1(0), J_3(0), J_1(\alpha), J_1(\beta), J_1(\gamma), N_1, N_3, 2L_0^T, L_1^T, L_2^T, L_3^T)$$

Consequently, the GUPTRI form of this 24×21 pencil includes all types of diagonal blocks in (8.7–8.8). 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:

$$A_i - \lambda B_i = \left[\begin{array}{cc|cc} \mathbf{z} & y & x & x \\ \hline 0 & \mathbf{y} & x & x \\ \hline 0 & 0 & \mathbf{x} & \mathbf{x} \\ 0 & 0 & 0 & \mathbf{x} \end{array}\right] - \lambda \left[\begin{array}{cc|cc} 0 & \mathbf{y} & x & x \\ \hline 0 & 0 & \mathbf{x} & \mathbf{x} \\ \hline 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array}\right]$$

$$A_l - \lambda B_l = \left[\begin{array}{ccc|ccc} z & y & y & x & x & x \\ \hline \mathbf{z} & y & y & x & x & x \\ 0 & y & y & x & x & x \\ \hline 0 & \mathbf{y} & \mathbf{y} & x & x & x \\ 0 & 0 & \mathbf{y} & x & x & x \\ 0 & 0 & 0 & x & x & x \\ \hline 0 & 0 & 0 & \mathbf{x} & \mathbf{x} & \mathbf{x} \\ 0 & 0 & 0 & 0 & \mathbf{x} & \mathbf{x} \\ 0 & 0 & 0 & 0 & 0 & \mathbf{x} \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right] - \lambda \left[\begin{array}{ccc|ccc} \mathbf{z} & y & y & x & x & x \\ \hline 0 & \mathbf{y} & \mathbf{y} & x & x & x \\ 0 & \mathbf{y} & \mathbf{y} & x & x & x \\ \hline 0 & 0 & 0 & \mathbf{x} & \mathbf{x} & \mathbf{x} \\ 0 & 0 & 0 & \mathbf{x} & \mathbf{x} & \mathbf{x} \\ 0 & 0 & 0 & \mathbf{x} & \mathbf{x} & \mathbf{x} \\ \hline 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right]$$

Since these forms are computed using the LI-staircase reduction, the block indices n_i and r_i start to count from the south-east 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:

$$A_f - \lambda B_f = \begin{bmatrix} \mathbf{x} & x & x \\ 0 & \mathbf{x} & x \\ 0 & 0 & \mathbf{x} \end{bmatrix} - \lambda \begin{bmatrix} \mathbf{x} & x & x \\ 0 & \mathbf{x} & x \\ 0 & 0 & \mathbf{x} \end{bmatrix}$$

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 round-off the problem is regularized by allowing a deflation criterion for range/nullspace separations and thereby make 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 nullspaces associated with the matrix pair.

8.2.6.2 Criterion for determining the numerical rank

The GUPTRI algorithm makes use of the singular value decomposition (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 $\Sigma = \mathrm{diag}(\sigma_1, \ldots, \sigma_n)$ with $0 \leq \sigma_1 \leq \cdots \leq \sigma_n$.

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 [52, 53]. 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.

8.2.6.3 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/nullspace separations in the GUPTRI algorithm are well-defined with respect to the deflation tolerance parameters ϵ and *gap*, then $\|(\delta A, \delta B)\|_F = \mathcal{O}(\epsilon)\|(A, B)\|_F)$. The nearby problem represents a regularized problem that has a stable Kronecker structure with respect to the deflation criteria of the algorithm.

8.2.7 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 is published in ACM TOMS [52, 53] and available from Netlib. The software is developed for dense matrices stored as double precision complex data. 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. For complete information we refer to 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 Section 8.2.7.3).

`GUPTRI:` Reduces $A - \lambda B$ to a GUPTRI form (8.7)-(8.8) 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 Section 8.2.7.3).

`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 [53] 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 $A_{11} - \lambda B_{11}$ (8.9), i.e., at the top north-west 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 [50, 53]. 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 `DELTA` = $\|(\delta A, \delta B)\|_E$.

8.2.7.1 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 [53] 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).

8.2.7.2 Arithmetic and space complexity

It is a quite 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 [53] (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`.

8.2.7.3 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

$$[\mathtt{S}, \mathtt{T}, \mathtt{P}, \mathtt{Q}, \mathtt{kstr}] = \mathtt{guptri}(\mathtt{A}, \mathtt{B}, \mathtt{EPSU}, \mathtt{GAP}, \mathtt{ZERO})$$

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 (8.7–8.8):

$$S = \begin{bmatrix} A_r & * & * & * & * \\ 0 & A_z & * & * & * \\ 0 & 0 & A_f & * & * \\ 0 & 0 & 0 & A_i & * \\ 0 & 0 & 0 & 0 & A_l \end{bmatrix}, \qquad T = \begin{bmatrix} B_r & * & * & * & * \\ 0 & B_z & * & * & * \\ 0 & 0 & B_f & * & * \\ 0 & 0 & 0 & B_i & * \\ 0 & 0 & 0 & 0 & B_l \end{bmatrix}, \tag{8.12}$$

where the diagonal blocks of S and T in staircase forms describe the Kronecker structure of $A - \lambda B$ (see Section 8.2.6).

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 used to make rank decisions in order to determine the Kronecker structure of an input pencil (see Section 8.2.6.2). 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 Section 8.2.6.3). 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:

$$A = \begin{bmatrix} 22 & 34 & 31 & 31 & 17 \\ 45 & 45 & 42 & 19 & 29 \\ 39 & 47 & 49 & 26 & 34 \\ 27 & 31 & 26 & 21 & 15 \\ 38 & 44 & 44 & 24 & 30 \end{bmatrix}, \qquad B = \begin{bmatrix} 13 & 26 & 25 & 17 & 24 \\ 31 & 46 & 40 & 26 & 37 \\ 26 & 40 & 19 & 25 & 25 \\ 16 & 25 & 27 & 14 & 23 \\ 24 & 35 & 18 & 21 & 22 \end{bmatrix}.$$

By construction the KCF of this example is $\mathrm{diag}(L_0, J_2(0), J_1(2), N_1, L_0^T)$. We compute the GUPTRI form of $A - \lambda B$ by the call

$$[\mathtt{S}, \mathtt{T}, \mathtt{P}, \mathtt{Q}, \mathtt{kstr}] = \mathtt{guptri}(\mathtt{A}, \mathtt{B}),$$

i.e., we use the default values on `EPSU`, `GAP` and `ZERO`. The output from this call is displayed in Figure 8.1.

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 $n_k - r_k \equiv \mathtt{kstr}(1, k+1) - \mathtt{kstr}(2, k+1)$.
- The number of $J_k(0)$ blocks is $r_k - n_{k+1} \equiv \mathtt{kstr}(2, k) - \mathtt{kstr}(1, k+1)$.

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 ($\mathtt{kstr}(1,1) - \mathtt{kstr}(2,1) = 1$), one $J_2(0)$ block ($\mathtt{kstr}(2,2) - \mathtt{kstr}(1,3) = 1$), one $L^T(0)$ block ($\mathtt{kstr}(1,5) - \mathtt{kstr}(2,5) = 1$), one N_1 block ($\mathtt{kstr}(2,5) - \mathtt{kstr}(1,6) = 1$), and finally a 1×1 regular part corresponding to the finite nonzero eigenvalue 2 ($\mathtt{kstr}(1,8) = \mathtt{kstr}(2,8) = 1$). The L_0 block corresponds to that A and B have a commmon column nullspace (both S and T have a first column with zero entries). Similarly, the L_0^T block corresponds to that A and B have a common row nullspace (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 $A_i - \lambda B_i = S(4,5) - \lambda T(4,5)$.

```
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
```

FIGURE 8.1: **Output (in short format) from the `guptri` function in Matlab.**

The computed transformation matrices P and Q are orthogonal to machine precision accuracy. Moreover,

$$\delta = \|(A - A', B - B')\|_F = 2.8184\epsilon - 13$$

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 $\mathcal{O}(\max(\|A\|_F, \|B\|_F)\epsilon_{\mathrm{M}})$, where ϵ_{M} is the machine accuracy.

By setting `ZERO` $= 0$ and making the call (with the default values on `EPSU` and `GAP`)

$$[\mathtt{S}, \mathtt{T}, \mathtt{P}, \mathtt{Q}, \mathtt{kstr}] = \mathtt{guptri}(\mathtt{A}, \mathtt{B}, \mathtt{EPSU}, \mathtt{GAP}, \mathtt{ZERO}),$$

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 in Figure 8.1 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 diagonal 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 $\|V\| \cdot \|V^{-1}\| = 8.8817\epsilon + 08$. 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 problem. And they do exist in real applications! Examples include controllability and observability issues in linear systems theory (see [249, 51]).

8.2.8 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 [186, 249, 51] and references therein). The basic theory dates back to Kronecker in 1890 [145] and is well presented in Gantmacher [95]. The method described there for computing the KCF is numerically unstable. Several authors have more recently proposed algorithms that reduces 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 [149]. She used a normalized RQ factorization for rank determinations and nullspace separations. In 1970 Ruhe [201] 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 [102] and further analyzed and generalized in [125]. Kågström and Ruhe [128, 127] 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 [24] developed a non-staircase qualitative approach. Kågström and Wiberg [129] consider the problem of computing partial canonical information for large scale eigenvalue problems. They combine the implicitly restarted Arnoldi (IRA) method [236] for computing a partial Schur form with staircase algorithms for computing the Jordan-Schur and the Weierstrass-Schur form of matrices and regular matrix pencils, respectively.

Van Dooren [248, 252] gave a generalization of Kublanovskya's staircase algorithm [149] 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 [147, 148]). Kågström [126] gave an RGSVD/RGQZD algorithm that provided a base for later work on software. Beelen and Van Dooren presented a faster but less reliable algorithm which requires $\mathcal{O}(m^2 n)$ operations for an $m \times n$ pencil. Error bounds for computed quantities like reducing subspaces and eigenvalues are given by Demmel and Kågström [50], which also are implemented in their software [52, 53].

Recently, several papers are published that use geometry of matrix and matrix pencil spaces to understand Jordan and Kronecker structure problems and algorithms. Demmel and Edelman [49] use the staircase algorithm to calculate the dimension of matrices and pencils with a given canonical form. Elmroth and Kågström [74] make a comprehensive study of the set of 2×3 pencils, the smallest nontrivial rectangular case. Edelman and Ma present a geometrical explanation of staircase algorithms failures [69]. Edelman, Elmroth and Kågström [67, 68] study versatility and stratifications. They discuss and complete the mathematical theory for orbits and bundles of matrices (Arnold [6]) and pencils (Pokrzywa [199], De Hoyos [44]) and propose that knowledge of the closure relations may be applied in the staircase algorithm [68]. StratiGraph is a recent Java-based tool for computation and presentation of graphs displaying closure hierarchies of Jordan and Kronecker structures [123, 73]. 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 [112], Boley [21], and Garcia-Planas [96]. 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.

8.3 Quadratic Eigenvalue Problems

Z. Bai, G. Sleijpen and H. van der Vorst

8.3.1 Introduction

In this section, we consider the quadratic eigenvalue problem (QEP) of the form

$$(\lambda^2 M + \lambda C + K)x = 0 \quad \text{and} \quad y^*(\lambda^2 M + \lambda C + K) = 0, \tag{8.13}$$

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, [94, 151]. 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

$$M^* = M > 0, \quad C^* = C \quad \text{and} \quad K^* = K > 0. \tag{8.14}$$

These matrices are sometimes called mass, damping and stiffness matrices, respectively, referring to their origin in mechanical engineering models, see, for instance, [61]. 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 §8.3.3.

One of factors 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 do no longer form an independent set. This is illustrated by the following simple example, in which we have taken some of the matrices as unsymmetric, in order to make some phenomena more clearly visible. The triplet

$$M = \begin{pmatrix} 5 & 2 \\ 1 & 4 \end{pmatrix}, C = \begin{pmatrix} 0 & 1 \\ 0 & 7 \end{pmatrix}, K = \begin{pmatrix} 2 & 0 \\ 0 & 12 \end{pmatrix}.$$

has 4 different (but pairwise conjugate) eigenvalues (rounded to 5 decimals):

$$\begin{aligned} \lambda_1 &= -0.9396 + 1.5749i, \quad \lambda_2 = -0.9396 - 1.5749i, \\ \lambda_3 &= -0.0049 + 0.6296i, \quad \lambda_4 = -0.0049 - 0.6296i. \end{aligned}$$

The associated eigenvectors (normalized so that the first coordinate is equal to one) are:

$$\begin{aligned} x_1 &= (1, -2.4756 - 0.9779i)^T, \quad x_2 = (1, -2.4756 + 0.9779i)^T, \\ x_3 &= (1,\ 0.0326 - 0.0132i)^T, \quad x_4 = (1,\ 0.0326 + 0.0132i)^T. \end{aligned}$$

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 (8.13), i.e.,

$$(\lambda^2 M + \lambda C + K)x = 0,$$

one can form a *quadratic Rayleigh quotient*:

$$\lambda^2 (x^* M x) + \lambda (x^* C x) + (x^* K x) = 0. \tag{8.15}$$

However, this equation has two roots, and 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 the example in Section 8.3.1, with (λ_1, x_1), then clearly, the pair (λ_1, x_1) satisfies equation (8.15). If we solve equation (8.15), 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), 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 that μ_2 is not an eigenvalues. We can not exclude 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.

8.3.2 Reduction to Linear Form

It is easy to see that the QEP in (8.13) is equivalent to the following generalized "linear" eigenvalue problem [1]:

$$Az = \lambda Bz \quad \text{and} \quad w^*A = \lambda w^*B \tag{8.16}$$

where

$$A = \begin{bmatrix} 0 & I \\ -K & -C \end{bmatrix}, \quad B = \begin{bmatrix} I & 0 \\ 0 & M \end{bmatrix} \tag{8.17}$$

and

$$z = \begin{bmatrix} x \\ \lambda x \end{bmatrix}, \quad w = \begin{bmatrix} (\lambda M + C)^* y \\ y \end{bmatrix}. \tag{8.18}$$

The generalized eigenvalue problem (8.16) is commonly called the linearization of the QEP (8.13). 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 (8.18).

Note that from the factorization

$$A - \lambda B = \begin{bmatrix} 0 & I \\ -I & -\lambda M - C \end{bmatrix} \begin{bmatrix} \lambda^2 M + \lambda C + K & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} I & 0 \\ -\lambda I & I \end{bmatrix}, \tag{8.19}$$

we can conclude that the pencil $A - \lambda B$ is *equivalent*[2] to the matrix

$$\begin{bmatrix} \lambda^2 M + \lambda C + K & 0 \\ 0 & I \end{bmatrix} \tag{8.20}$$

and

$$\det(A - \lambda B) = \det(\lambda^2 M + \lambda C + K),$$

This means that the eigenvalues of the original QEP (8.13) coincide with the eigenvalues of the generalized eigenvalue problem (8.16). Furthermore, we have

[1] The term "linear" is in the quotation mark, because λ appears in the eigenvectors as well.

[2] Two 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).

- $L(\lambda)$ 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 theory on regular pencils (A, B), see for instance [239, Chapter VI]. We will assume that at least M is nonsingular throughout this section.

A disadvantage of the above reduction to generalized form is that if the matrices M, C and K are all Hermitian, then this is not reflected in the reduced form (8.17), where A is non-Hermitian. This can be repaired as follows.

In fact, the matrix pair (A, B) in (8.16) 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 (8.20) if and only if W is nonsingular, and because of (8.19)

$$\det(A - \lambda B) = \det(W) \cdot \det(\lambda^2 M + \lambda C + K).$$

For example, if the matrices M, K and C are all symmetric, as in the special case (8.14), and K is nonsingular, then we may choose $W = -K$, which leads to the following symmetric generalized "linear" eigenvalue problem

$$Az = \lambda B z \quad \text{and} \quad w^* A = \lambda w^* B \tag{8.21}$$

where

$$A = \begin{bmatrix} 0 & -K \\ -K & -C \end{bmatrix}, \quad B = \begin{bmatrix} -K & 0 \\ 0 & M \end{bmatrix}, \tag{8.22}$$

and

$$z = \begin{bmatrix} x \\ \lambda x \end{bmatrix}, \quad w = \begin{bmatrix} y \\ \bar{\lambda} y \end{bmatrix}. \tag{8.23}$$

Both A and B are symmetric, but may be indefinite.

8.3.3 Spectral Transformations for QEP

8.3.3.1 Invert QEP

For most iterative methods for solving a generalized eigenvalue problem, the formulation (8.16), with either (8.17) or with (8.22), 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 (8.13) with μ^2, we obtain the *invert QEP*:

$$\left(M + \mu C + \mu^2 K\right) x = 0. \tag{8.24}$$

Here it is assumed that that 0 is not an eigenvalues of the original QEP (8.13), i.e., K is nonsingular.

The QEP for the triplet $\{K, C, M\}$ can be linearized as discussed in §8.3.2, for instance as (8.16) with (8.17), where M interchanged with K. We can reformulate this generalized linearized eigenproblem in terms of λ, instead of μ, which leads to

$$Az = \frac{1}{\lambda} Bz \tag{8.25}$$

where

$$A = \begin{bmatrix} -C & -M \\ I & 0 \end{bmatrix}, \quad B = \begin{bmatrix} K & 0 \\ 0 & I \end{bmatrix}, \quad z = \begin{bmatrix} x \\ \lambda x \end{bmatrix}. \tag{8.26}$$

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 (8.13) coincide with the eigenvalues of the matrix pencil $B - \lambda A$.

For the special case (8.14), we may formulate the generalized eigenvalue problem $Az = \frac{1}{\lambda} Bz$, with

$$A = \begin{bmatrix} C & M \\ M & 0 \end{bmatrix}, \quad B = \begin{bmatrix} -K & 0 \\ 0 & M \end{bmatrix}. \tag{8.27}$$

In this case, both matrices are Hermitian, but indefinite. Linearization with (8.27) results after left multiplication of (8.26) with a block diagonal matrix $\mathrm{diag}(-I, -M)$. 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.

8.3.3.2 Shifted QEP

With a shift $\lambda = \mu + \sigma$, the *shifted QEP* is

$$\left(\mu^2 \widehat{M} + \mu \widehat{C} + \widehat{K}\right) x = 0, \tag{8.28}$$

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 (8.13) close to σ become 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 Hermitian of the matrix triplet $\{M, K, C\}$ wants to be preserved.

8.3.3.3 Shift-and-Invert QEP

Combining the above invert and shift spectral transformations, the so-called *shift-and-invert QEP* becomes

$$\left(\mu^2 \widehat{M} + \mu \widehat{C} + \widehat{K}\right) x = 0, \tag{8.29}$$

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 (8.29) approximate the eigenvalues λ of the original QEP (8.13) 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 Hermitian of the matrix triplet $\{M, K, C\}$ wants to be preserved.

8.3.3.4 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 (8.13) becomes

$$\left(\mu^2 \widehat{M} + \mu \widehat{C} + \widehat{K}\right) x = 0, \tag{8.30}$$

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 (8.13) close to the *anti-shift* τ are transformed into large (in modulus) eigenvalues μ of the QEP (8.30). Eigenvalues λ close to the *shift* β/α correspond to eigenvalues μ of (8.29) 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.

8.3.4 Numerical Methods for Solving Linearized Problems

As discussed in §8.3.2, one can use either the generalized eigenvalue problem (8.16) with (8.17) or the generalized eigenvalue problem (8.21) with (8.22), for solving the corresponding quadratic eigenvalue problem (8.13).

If all matrices M, C and K are Hermitian and M is positive definite, as in the special case (8.14), then the decision comes down to choosing either intrinsically non-Hermitian generalized eigenvalue problem (8.16) and (8.17), with a Hermitian positive definite B matrix, or a generalized eigenvalue problem (8.21) and (8.22), where both A and B matrices are Hermitian but neither of them will be positive definite.

Numerical methods discussed in Chapter ?? can be used for solving these generalized "linear" eigenvalue problems. The indefinite symmetric Lanczos method discussed in §?? is specifically targeted for Hermitian indefinite generalized eigenvalue problem (8.21). The symmetric Lanczos method, see §??, is formally extended to solve such symmetric indefinite generalized eigenvalue problem. The trouble is that the basis vectors are orthogonal with respect to an indefinite inner product. Therefore, these basis vectors may be linearly independent and the algorithm may be breakdown and numerical instable. Nevertheless this is an attractive way to solve the original QEP because of potential savings in memory requirements and floating point operations. See §?? for further details.

8.3.5 The 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.

As we will see below, this method can also be applied directly to polynomial eigenproblems

$$(C_0 + \lambda C_1 + \ldots + \lambda^\ell C_\ell)x = 0, \tag{8.31}$$

and no transformation to a generalized "linear" eigenvalue problem will be required, where C_i for $i = 0, 1, \ldots, \ell$, 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

$$\Psi(\lambda)x = 0 \tag{8.32}$$

where

$$\Psi(\lambda) = C_0 + \lambda C_1 + \lambda^2 C_2, \tag{8.33}$$

the projected polynomial problem

$$(V^* C_0 V + \theta V^* C_1 V + \lambda^2 V^* C_2 V)y = 0 \tag{8.34}$$

is solved. The columns v_i of the $n \times j$ 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 $j \ll n$. We will assume that the solution vectors y are normalized, $\|y\|_2 = 1$. First, a Ritz value θ with the desired properties, such as the largest real part or closet to some target τ, is selected and for the associated eigenvector y. Then the Ritz vector

$u = Vy$ and the residual $r = \Psi(\theta)u$ is computed. For expansion of the search space the vector p,

$$p = \Psi'(\theta)u$$

with

$$\Psi'(\theta) = C_1 + 2\theta C_2,$$

is also computed.

In the second part of the Jacobi-Davidson iteration step, the search subspace $\mathrm{span}(V)$ is expanded by a vector $z \perp u$ that solves (approximately) the correction equation

$$\left(I - \frac{p\,u^*}{u^*p}\right)\Psi(\theta)\,(I - u\,u^*)\,z = -r. \tag{8.35}$$

The next column of V is obtained by orthonormalizing the approximate solution against the previously computed columns $v_1, \ldots, v_j$.

This process is repeated until an eigenpair (λ, x) has been detected, i.e., until the residual vector r is sufficient small. The basic form the algorithm is presented in Algorithm 8.39. We refer to [229], for an example of quadratic eigenvalue problem arising from an accoustic problem.

ALGORITHM 8.39: **Jacobi-Davidson Method**

(1) *Choose an $n \times m$ orthonormal matrix V.*
(2) *For $i = 0, 1, 2$*
  compute $W_i = C_i V$ and $M_i = V^* W_i$.
(3) *Iterate until convergence*
(4) *Compute the eigenpairs (θ, y) of*
  $\left(M_0 + \theta M_1 + \theta^2 M_2\right) y = 0$.
(5) *Select the desired eigenpair (θ, y) with $\|y\|_2 = 1$.*
(6) *compute $u = Vy$, $p = \Psi'(\theta)u$, $r = \Psi(\theta)u$.*
(7) **If** $(\|r\|_2 < \epsilon)$, $\lambda = \theta$, $x = u$, **stop**.
(8) *Solve (approximately) a $z \perp u$ from*
  $$\left(I - \frac{p\,u^*}{u^*p}\right)\Psi(\theta)\,(I - u\,u^*)\,z = -r$$
(9) *Orthogonalize z against V, $v = z/\|z\|_2$.*
  For $i = 0, 1, 2$
    compute $w_i = C_i v$
    $$M_i = \begin{bmatrix} M_i & V^* w_i \\ v^* W_i & v^* w_i \end{bmatrix},\ W_i = [W_i, w_i]$$
(10) *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, $j = j_{\max}$, 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 $j_{\min}$ Ritz pairs from the last step, i.e., take $V = [u_1, \ldots, u_{j_{\min}}]$ where $u_1, \ldots, u_{j_{\min}}$ are the Ritz vectors associated with the best available $j_{\min}$ Ritz values $\theta_1, \ldots, \theta_{j_{\min}}$ apply modified Gram-Schmidt to orthonormalize V and restart with this matrix. Note that the new search matrix $V = V_{j_{\min}}$ can be expressed in terms of the old search matrix $V = V_{j_{\max}}$ as $V_{j_{\min}} = V_{j_{\max}} T$ for some $j_{\max} \times j_{\min}$ matrix T. The transformation matrix

T can explicitly be computed, 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 reconstruction known eigenvectors. The procedure is similar to the explicit deflation approach introduced in §??. For computational details, we refer to that section.

In §?? and ??, we suggested to use 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 can not easily be gereralized.

8.3.6 Notes and References

Numerical algorithm design and analysis for the solution of quadratic eigenvalue problem are still an active researcher subject. Besides those methods discussed in this section, some alternative methods are available in literature, see for example [151, 146, 202, 41, ?]. Most of these methods are variants of Newton's method. They generally have good local convergence properties, and find one eigenpair at time. In [?], a proper deflation technique is presented for finding more than one eigenpair.

Recently, a backward error analysis of the quadratic eigenvalue problems and more generally, the polynomail eigenvalue problems is presented in [?] is studied. In [?], a perturbation analysis of the quadratic eigenvalue problem (8.14) is presented.

8.4 Nonlinear Eigenvalue Problems with Orthogonality Constraints

A. Edelman and R. Lippert

8.4.1 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 [71, 70, 232]).

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 [45]) 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 $F(Y) = \mathrm{tr}(Y^* A Y)$, 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

$$Q = \begin{bmatrix} Q_1 & 0 & \cdots & 0 \\ 0 & Q_2 & \cdots & 0 \\ \cdot\cdot & \cdot\cdot & \cdot\cdot & \cdot\cdot \\ 0 & 0 & \cdots & Q_p \end{bmatrix},$$

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. Section 8.4.2 will discuss the basics of calling the template code. Section 8.4.3 discusses the objective functions and their derivatives for the example problems explored in this chapter. Section 8.4.4 contains sample runs of instances of the examples problems. Section 8.4.5 explains the code structure and the places where a user may wish to make modifications. Section 8.4.6 will cover some of the basic mathematics concerning the geometry of the Stiefel manifold.

8.4.2 Matlab Templates

The templates are ready for immediate use or as a departure point for generalization, 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 `Y0`. 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

argument	description
rc	{'real', 'complex' }
mode	{'newton', 'dog', 'prcg', 'frcg' }
metric	{'flat', 'euclidean', 'canonical' }
motion	{'approximate', 'exact' }
verbose	{'verbose', 'quiet'}
ftol	first convergence tolerance
gradtol	second convergence tolerance
partition	partition of p indicating symmetries of f

TABLE 8.1: **A short list of the optional arguments for sg_min**

```
>> 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
  invdgrad: Hessian not positive definite, CG terminating early
1       1.342879e+01   2.011465e+01       163639       Newton step
  invdgrad: Hessian not positive definite, CG terminating early
2       1.130915e+01   1.368044e+01       344198       Newton step
  invdgrad: Hessian not positive definite, CG terminating early
3       5.974819e+00   1.063045e+01       515919       Newton step
  invdgrad: max iterations reached inverting the hessian by CG
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 8.1):

`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 Fletcher-Reeves conjugate gradient. Lastly, 'prcg' selects Polak-Ribiere conjugate gradient which has the advantage that it does not require a very accurate Hessian (and thus, 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.

`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.

8.4.3 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 $\mathrm{tr}(V^* dF(Y)) = \frac{d}{dt} F(Y(t))|_{t=0}$ where $Y(t) = Y + tV$ (or any curve $Y(t)$ for which $\dot{Y}(0) = V$). The reader may recall that $\mathrm{tr}(A^* B) = \sum_{i,j} A_{ij} B_{ij}$, so

it functions just like the real inner product for vectors[3] 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 $F(Y) = \mathrm{tr}(AY^*BY)$ one then has

$$\mathrm{tr}(V^*dF(Y)) = \mathrm{tr}(AV^*BY + AY^*BV).$$

Since the value of the trace is invariant under cyclic permutations of products and transposes, we may rewrite this equation as

$$\mathrm{tr}(V^*dF(Y)) = \mathrm{tr}(V^*BYA + V^*B^*YA^*),$$

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 `finitediff` 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 `finitediff` (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 $(2np)$.

8.4.3.1 The Procrustes Problem

The Procrustes problem (see [72]) is the minimization of $||AY - YB||_F$ for constant A and B over the manifold $Y^*Y = I$. This minimization determines the nearest matrix $\widehat{A}$ to A for which

$$Q^*\widehat{A}Q = \begin{bmatrix} B & * \\ 0 & * \end{bmatrix},$$

i.e. the columns of B span an invariant subspace of $\widehat{A}$.

The differential of $F(Y) = \frac{1}{2}||AY - YB||_F^2 = \frac{1}{2}\mathrm{tr}(AY - YB)^*(AY - YB)$ is given by

$$dF(Y) = A^*(AY - YB) - (AY - YB)B^*.$$

[3] over complex numbers, we always take the real part of $\mathrm{tr}(A^HB)$.

This can be derived following the process outlined above. Observe that

$$\begin{aligned}\frac{d}{dt}F(Y(t))|_{t=0} &= \frac{1}{2}\mathrm{tr}((AV-VB)^*(AY-YB)+(AY-YB)^*(AV-VB)) \\ &= \mathrm{tr}((AV-VB)^*(AY-YB)) \\ &= \mathrm{tr}(V^*(A^*(AY-YB))-V^*(AY-YB)B^*).\end{aligned}$$

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.

8.4.3.2 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:

$$B(Y) = \begin{bmatrix} \lambda_1 I & * & * \\ 0 & \lambda_2 I & * \\ 0 & 0 & \ddots \end{bmatrix}$$

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 $\mathrm{tr}(-(AY-YB(Y))^*Y\dot{B}) = \mathrm{tr}((B(Y)-Y^*AY)^*\dot{B}) = 0$ for the B selected as above for either orbit or 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 the B now depend on Y.

$$\frac{d}{dt}dF(Y(t))|_{t=0} = \frac{d}{dt}dF_{Proc}(Y(t))|_{t=0} - A^*Y\dot{B} - AY\dot{B}^* + YB\dot{B}^* + Y\dot{B}B^*,$$

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).

8.4.3.3 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 $F(Y) = \frac{1}{2}\text{tr}(Y^*AY)$ [4, 91, 218]. 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 $\dot{Y}(0) = H$.

8.4.3.4 Trace Minimization with a Nonlinear Term

We consider a related nonlinear eigenvalue problem, to the trace minimization of section 8.4.3.3: 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 (LDA) 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, [244, 242]). 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

$$\frac{d}{dt}dF(Y(t))|_{t=0} = \frac{d}{dt}dF_{\text{tracemin}}(Y(t))|_{t=0} + K\text{diag }(\rho)H + K\text{diag }(\frac{d}{dt}\rho)Y,$$

where $\dot{Y}(0) = H$, and $\frac{d}{dt}\rho_i|_{t=0} = \sum_j Y_{ij}H_{ij}$.

8.4.3.5 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 $Y \in O(n)$ such that Y^*AY and Y^*BY are both block upper triangular. 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 W. Schilders [220] 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,

$$\begin{aligned}\text{tr}(V^*dF(Y)) &= \text{tr}(\text{low}(Y^*AY)^*\text{low}(Y^*AV + V^*AY) + \\ &\quad \text{low}(Y^*BY)^*\text{low}(Y^*BV + V^*BY))\end{aligned}$$

$$\begin{aligned}
\mathrm{tr}(V^* dF(Y)) &= \mathrm{tr}(\mathrm{low}(Y^*AY)^*(Y^*AV + V^*AY) + \\
&\quad \mathrm{low}(Y^*BY)^*(Y^*BV + V^*BY)) \\
\mathrm{tr}(V^* dF(Y)) &= \mathrm{tr}(V^*(AY\mathrm{low}(Y^*AY)^* + A^*Y\mathrm{low}(Y^*AY) + \\
&= BY\mathrm{low}(Y^*BY)^* + B^*Y\mathrm{low}(Y^*BY))) \\
dF(Y) &= AY\mathrm{low}(Y^*AY)^* + A^*Y\mathrm{low}(Y^*AY) + \\
&\quad BY\mathrm{low}(Y^*BY)^* + B^*Y\mathrm{low}(Y^*BY),
\end{aligned}$$

where the second equation results from observing that $\mathrm{tr}(\mathrm{low}(M)^*\mathrm{low}(N)) = \mathrm{tr}(\mathrm{low}(M)^*N)$, and the third from properties of the trace.

With second derivatives given by,

$$\begin{aligned}
\frac{d}{dt}dF(Y(t))|_{t=0} &= AH\mathrm{low}(Y^*AY)^* + A^*H\mathrm{low}(Y^*AY) + AY\mathrm{low}(\frac{d(Y^*AY)}{dt})^* \\
&\quad + A^*Y\mathrm{low}(\frac{d(Y^*AY)}{dt}) + BH\mathrm{low}(Y^*BY)^* + B^*H\mathrm{low}(Y^*BY) \\
&\quad + BY\mathrm{low}(\frac{d(Y^*BY)}{dt})^* + B^*Y\mathrm{low}(\frac{d(Y^*BY)}{dt}),
\end{aligned}$$

where $\dot{Y}(0) = H$, $\frac{d(Y^*AY)}{dt}|_{t=0} = H^*AY + Y^*AH$, and $\frac{d(Y^*BY)}{dt}|_{t=0} = H^*BY + Y^*BH$.

8.4.3.6 Simultaneous Diagonalization

Another problem like 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 [45].

Phrased in terms of a minimization, one has a set of symmetric matrices A_i and wishes to find $Y \in O(n)$ that minimizes $F(Y) = \frac{1}{2}\sum_i ||[Y, A_i]||_F^2 = \frac{1}{2}\sum_i ||YA_i - A_iY||_F^2$.

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 $\dot{Y}(0) = H$.

8.4.4 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   invdgrad_CG.m      partition.m  sg_prcg.m
connection.m  examples     invdgrad_MINRES.m  sg_dog.m     tangent.m
dFline.m      finitediff   ip.m               sg_frcg.m
```

(note, 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     simschur    tracemin
```

Each of these subdirectories contains an implementation for $F(Y)$ ($dF(Y)$ and $ddF(Y,H)$), a `parameters` function to set a global, `FParameters`, of fixed 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/ldatoy
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.

8.4.4.1 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
```

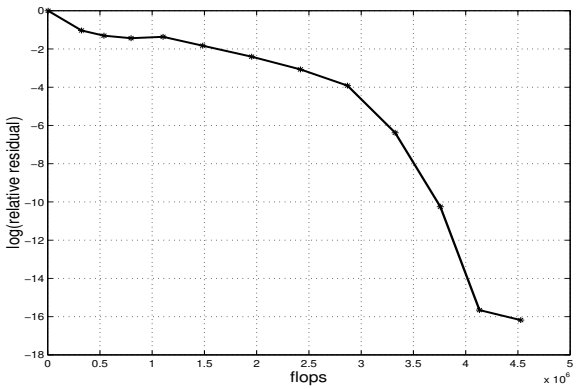


FIGURE 8.2: **Procrustes Problem**

```
  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 8.2 shows the convergence curve for this run.

8.4.4.2 A Sample Jordan Block Problem

We now look for a nearby matrix with a Jordan structure $J_4(\alpha) \oplus J_2(\beta) \oplus$ (don't care) 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')
>> Y0 = guess;
>> [fn,Yn] = sg_min(Y0,'newton','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
```

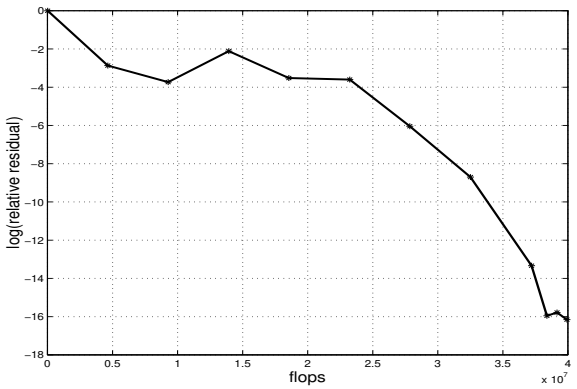


FIGURE 8.3: **Jordan Problem**

```
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 8.3 shows the convergence curve for this run.

8.4.4.3 A Sample Trace Minimization Problem

In this problem we minimize trace(Y^*AY) where A is a 12×12 second difference operator in 1-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 .
>> A = Kinetic(12);
>> parameters(A);
>> Y0 = guess(4);
>> [fn,Yn] = sg_min(Y0,'frcg','euclidean');
iter    grad            F(Y)              flops
0       2.394195e+00    3.795507e+00         3807
1       9.554757e-01    1.852416e+00        99121
2       8.846460e-01    1.539249e+00       188670
3       9.787770e-01    1.100793e+00       301611
4       3.524675e-01    8.621464e-01       391159
5       1.283673e-01    8.310326e-01       491803
6       4.247081e-02    8.274924e-01       614658
7       1.284919e-02    8.270902e-01       762413
8       6.044392e-03    8.270506e-01       913016
9       4.815684e-03    8.270365e-01      1074737
10      2.471180e-03    8.270285e-01      1221750
```

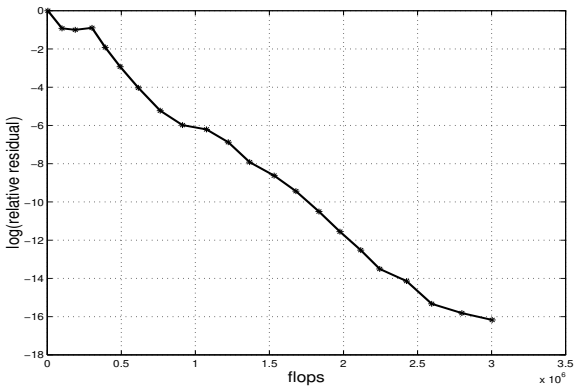


FIGURE 8.4: **Trace Minimization Problem**

```
11      8.682710e-04    8.270269e-01      1366396
12      4.306049e-04    8.270267e-01      1531697
13      1.912666e-04    8.270267e-01      1678710
14      6.577101e-05    8.270267e-01      1834249
15      2.297119e-05    8.270267e-01      1976117
16      8.729576e-06    8.270267e-01      2116416
17      3.269077e-06    8.270267e-01      2244448
18      1.731924e-06    8.270267e-01      2425696
19      5.268308e-07    8.270267e-01      2595368
20      3.240422e-07    8.270267e-01      2799779
21      2.264230e-07    8.270267e-01      3004260
```

Figure 8.4 shows the convergence curve for this run.

8.4.4.4 A Sample LDA Toy Problem (trace minimization with nonlinear term)

We attempt to solve a toy Local Density Approximation problem described in section 8.4.3.4, for four electrons on a 1D grid with twelves 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 8.5 shows the convergence curve for this run.

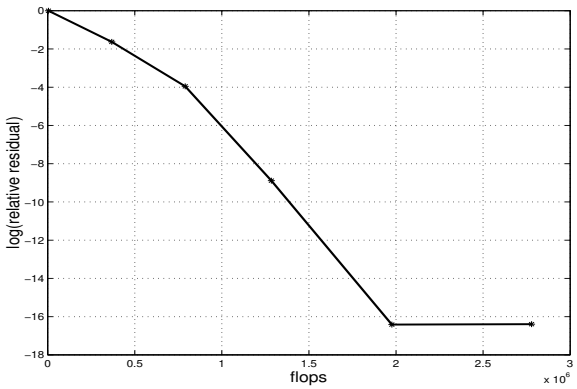


FIGURE 8.5: **LDA Toy Problem**

8.4.4.5 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
```

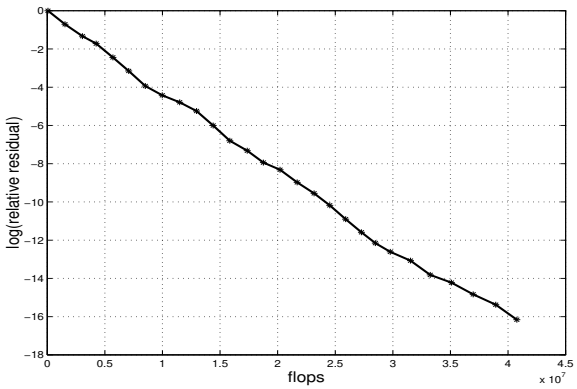


FIGURE 8.6: **Simultaneous Schur Problem**

```
26      4.631213e-08    4.913255e-03     38934071
27      2.119895e-08    4.913255e-03     40761932
```

Figure 8.6 shows the convergence curve for this run.

8.4.4.6 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/*.m .
>> 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
```

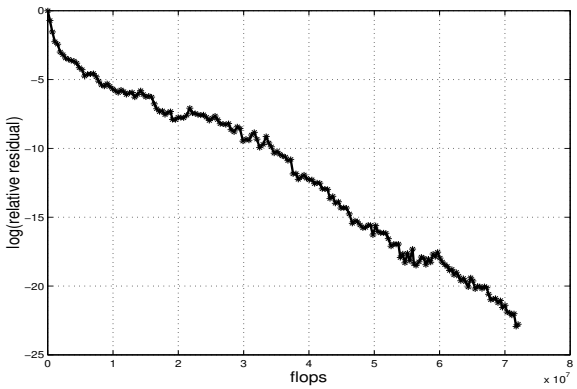


FIGURE 8.7: **Simultaneous Diagonalization Problem**

```
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 8.7 shows the convergence curve for this run.

8.4.5 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.

8.4.5.1 Subroutine Dependencies

Figure 8.8 shows the dependencies of the various subroutines on each other. We have grouped together the routines 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 interesting 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 group and High Level Algorithm group.

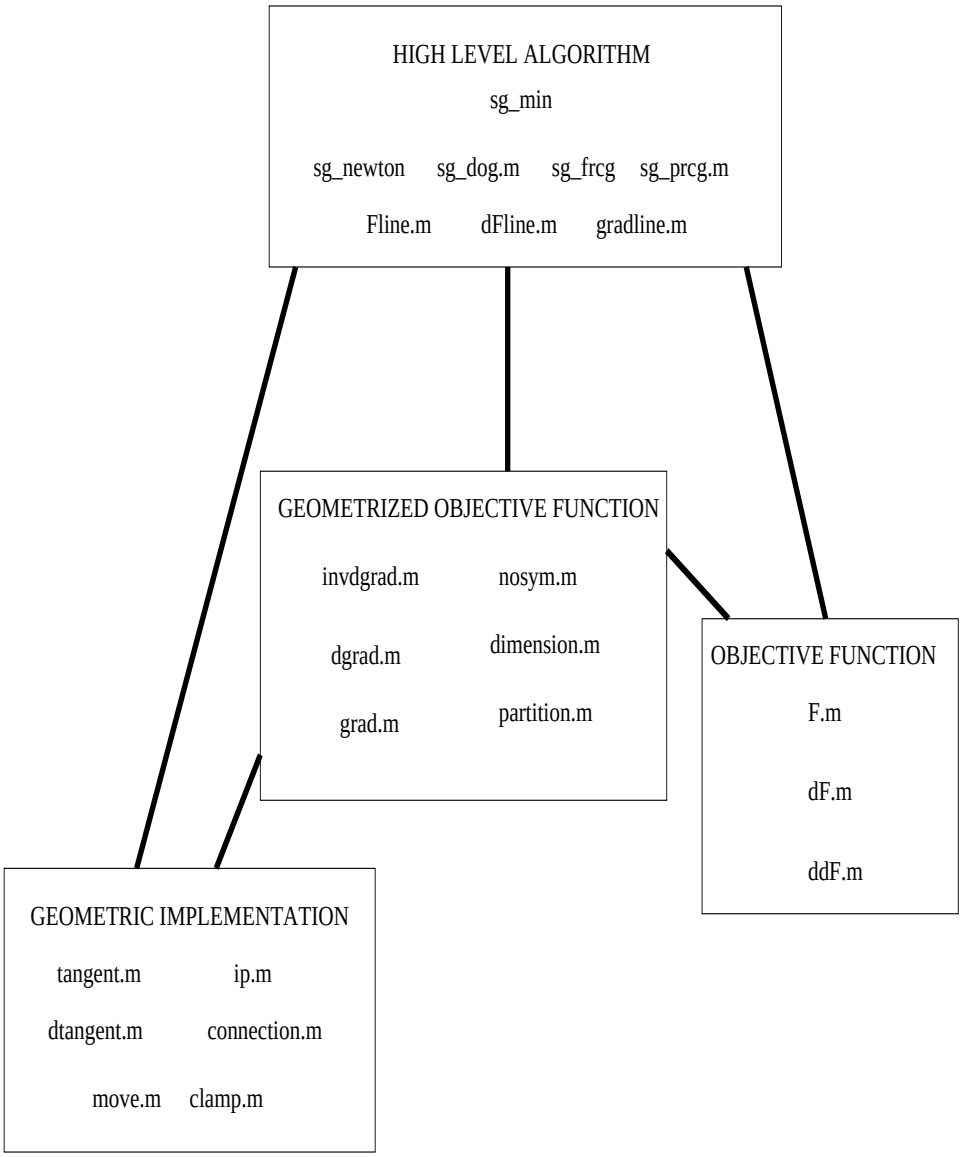


FIGURE 8.8: **Our code tree: dependencies of various modules**

- **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 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`.

8.4.5.2 Under the Hood

`sg_min` is built from a couple 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 below.

```
function [fn,Yn]= sg_newton(Y)
     g_initial = grad(Y); g = g_initial;
     f = F(Y); f_old = 2*f;
     while (||g|| > eps) & (||g||/||g_initial|| >gradtol) & (|f_old/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] · |f / ⟨g,sdir⟩|;
          fsb = minimize F(move(Y,sdir,sb)) for sb ∈ [-1,1] · |⟨g,sdir⟩ / ⟨sdir,dgrad·sdir⟩|;

          ndir = -dgrad^-1· g;
% newton direction and line minimization
          fna = minimize F(move(Y,ndir,na)) for na ∈ [-1,1] · |f / ⟨g,ndir⟩|;
          fnb = minimize F(move(Y,ndir,nb)) for nb ∈ [-1,1] · |⟨g,ndir⟩ / ⟨ndir,dgrad·ndir⟩|;

% 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; f_new = fst;
          else
               dir = ndir; t = nt; f_new = fnt;
          end
% move to the new point
          Y = move(Y,dir,t); f_old= f; f = f_new;
          g=grad(Y);
     end
     fn = f;
     Yn = Y;
```

```
function [fn,Yn]= sg_frcg(Y)
     g_initial = grad(Y); g = g_initial;
     f = F(Y); f_old = 2*f;
     while (||g|| > eps) & (||g||/||g_initial|| >gradtol) & (|f_old/f-1 | > ftol)
          dir = -g;
% get a Hessian conjugate direction
          if (not first iteration)
               dir = dir - (dir·(dgrad·dir_old)) / (dir_old·(dgrad·dir_old))*dir_old;
          end
% line minimization
          fcga = minimize F(move(Y,dir,cga)) for cga ∈ [-1,1] · |f / ⟨g,dir⟩|;
          fcgb = minimize F(move(Y,dir,cgb)) for cgb ∈ [-1,1] · |⟨g,dir⟩ / ⟨dir,dgrad·dir⟩|;

          if (fcga < fcgb) t=cga; f_new=fcga; else t=cgb; f_new=fcgb; end
% move to the new point
          [Y,dir_old] = move(Y,dir,t); f_old= f; f = f_new;
          g=grad(Y);
     end
     fn = f;
     Yn = Y;
```

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 [80, 99, 200, 181]). 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.

8.4.5.3 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` functions 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.

- **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 criteria [5] 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 inlined, that 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 $O(n)$ Geometry** Some problems are always set on the unit sphere or on $O(n)$, 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 a 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`.

8.4.6 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.

8.4.6.1 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 be discussing those conditions in this section.

We consider the manifold 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 Stief(n,k) of functions of nk variables which are $C^\infty(\mathcal{R}^{n\times k})$ about 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 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,

$$\widetilde{Y} = \begin{bmatrix} A \\ B \end{bmatrix}.$$

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 $C^\infty(R^{(n-k)\times k} \oplus R^{k\times k})$ for small enough B and Δ, we have shown that any point of 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 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 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).

8.4.6.2 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 Stief(n,k) which must all have constant value. If H is a tangent vector of 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 a $nk - k(k+1)/2$ 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 $\mathcal{R}^{n\times k}$,

$$D_H f = \mathrm{tr}(H^*Z),$$

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.

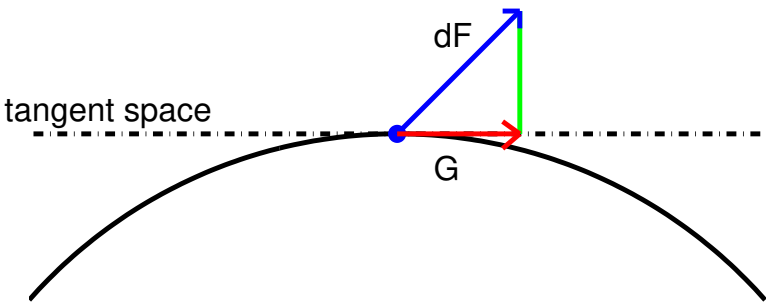


FIGURE 8.9: *The unconstrained differential of $F(Y)$ can be projected to the tangent space to obtain the covariant gradient, G, of F.*

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 $\mathrm{tr}(H^* df)$ to compute $D_H f$ for $H \in T_Y(\mathrm{Stief}(n,k))$. This is why it requires no geometric knowledge to produce the `dF` functions.

8.4.6.3 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 $\mathrm{Stief}(n,k)$, 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 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 $\mathrm{Stief}(n,k)$ at Y. Then one can define the gradient, G, implicitly by,

$$ip(G,H) = \mathrm{tr}(H^* df) = D_H f.$$

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 8.9). This operation is performed by `grad` to produce the gradient of the objective function.

8.4.6.4 Getting Around 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 [71, 70, 232]). 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.

8.4.6.5 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.

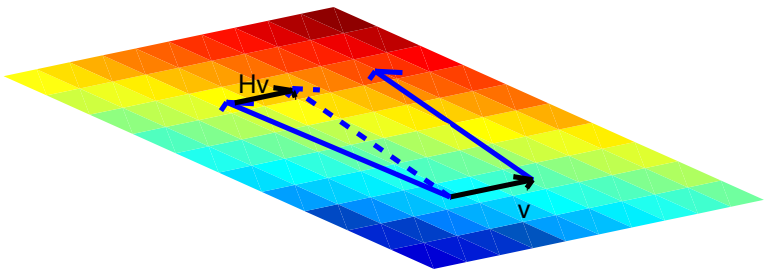


FIGURE 8.10: *In a flat space, comparing vectors at nearby points is not problematic since all vectors lie in the same tangent space.*

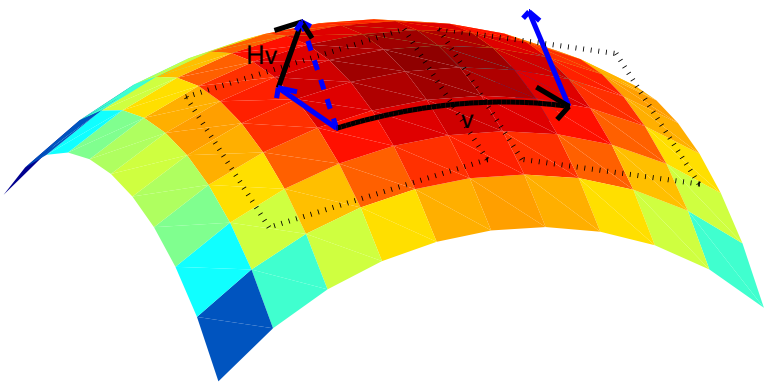


FIGURE 8.11: *In a curved manifold, comparing vectors at nearby points can result in vectors which do not lie in the tangent space.*

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 8.10), 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 8.11).

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

$$V \to V + \epsilon \Gamma(V, H),$$

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 found in the usual literature ([27, 257, 143, 109], 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,

$$[\frac{\partial^2 f}{\partial x_i \partial x_j}]\vec{h} = (\vec{h} \cdot \nabla) g \sim D_H G.$$

8.4.6.6 Inverting the Covariant Hessian (technical considerations)

Since the tangent space of 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.

$$Q = \begin{bmatrix} Q_1 & 0 & \cdots & 0 \\ 0 & Q_2 & \cdots & 0 \\ \cdot\cdot & \cdot\cdot & \cdot\cdot & \cdot\cdot \\ 0 & 0 & \cdots & Q_p \end{bmatrix},$$

where the Q_i are orthogonal matrices). In this case, all tangent vectors of the form

$$H = Y \begin{bmatrix} H_1 & 0 & \cdots & 0 \\ 0 & H_2 & \cdots & 0 \\ \cdot\cdot & \cdot\cdot & \cdot\cdot & \cdot\cdot \\ 0 & 0 & \cdots & H_p \end{bmatrix},$$

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 9

Common Issues

9.1 Accuracy Assessment and Numerical Stability

Z. Bai and R. Li

9.1.1 Introduction

In this section, we discuss the existing tools to assess the accuracy of computed eigenvalues and corresponding eigenvectors. 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. There are two different ways to look at the accuracy issues. One is to focus on the difference – known as *the forward error* – between a computed eigenvalue and the exact eigenvalue that is supposed to be computed. This is *the forward error analysis*, a direct way to assess the accuracy. Another way is to interpret the computed eigenvalue and eigenvectors as the exact ones of some matrix nearby the matrix of interest. This is *the backward error analysis*, and the difference between the original matrix and the nearby one is *the backward error*.

Numerical stability analysis concerns two issues: the stability of a problem and the stability of an algorithm. The former addresses how the solution to a problem changes if the data defining the problem are perturbed, while the latter addresses the accuracy of computed results by an algorithm. We say that a numerical algorithm is *forward stable* if it computes eigenvalues and eigenvectors with small errors; analogously a numerical algorithm is *backward stable* if it computes eigenvalues and eigenvectors with small backward errors. Both notions of forward and backward stabilities for dense matrix computations, e.g., the QR algorithm for the standard eigenvalue problem (see §5.2), are well defined and understood [101, 48], but the situations becomes much more complicated in the world of iterative methods. One main reason is that the numerical behavior of an iterative method critically depends on the intrinsic properties of the underlying matrices. Saying that an iterative algorithm is stable would not make any sense without taking into consideration the problem it applies to.

Knowing the differences between the computed and the exact ones is clearly a preferred choice; but estimating the differences may not be immediately possible upon

exit of some algorithms for which not enough information is provided unless additional computation is performed afterwards. However, backward errors are always readily available, given the residual errors. We caution the reader that in practice usually small backward errors imply small forward errors but not always.

Analysing forward errors in computed eigenvectors is much more complicated than in computed eigenvalues. In fact, the errors in eigenvectors depend not only on relevant residual errors but also on how eigenvalues other than those associated with the eigenvectors distribute and on the structure of the underlying matrix. For this reason, we shall present in rather detail about error bounds on eigenvalues, and only give brief discussions in §9.1.5 on how eigenvectors may behave. Interested readers will be pointed to available literature for further reading. For clarity, we shall deal with the following problems separately in the coming subsections:

1. the standard eigenvalue problem $Ax = \lambda x$ in §9.1.2, and
2. the generalized eigenvalue problem $Ax = \lambda Bx$, for which we further distinguish
   (a) B is Hermitian positive definite in §9.1.3), and
   (b) B is arbitrary and even singular, but the pencil $A-\lambda B$ is regular[1], in §9.1.4.

9.1.2 Standard eigenvalue problem

Let $\widetilde{\lambda}$ denote a computed eigenvalue, and let $\widetilde{x}$ be its corresponding computed eigenvector, i.e.,

$$A\widetilde{x} \approx \widetilde{\lambda}\widetilde{x}.$$

Sometimes, its corresponding left computed eigenvector $\widetilde{y}$ is also available:

$$\widetilde{y}^* A \approx \widetilde{\lambda}\widetilde{y}^*.$$

Notice that $\widetilde{y} = \widetilde{x}$ when A is Hermitian. Let

$$r = A\widetilde{x} - \widetilde{\lambda}\widetilde{x}$$

and

$$s^* = \widetilde{y}^* A - \widetilde{\lambda}\widetilde{y}^*$$

denote the corresponding residual vectors. We are interested in knowing the accuracy of computed eigenvalue and eigenvectors in terms of these residuals.

- If A is Hermitian, then there is an eigenvalue λ of A satisfying

$$|\lambda - \widetilde{\lambda}| \leq \frac{\|r\|_2}{\|\widetilde{x}\|_2}. \tag{9.1}$$

  A better bound in the order of $\|r\|_2^2$ can be proved if there is a reasonable gap between the λ and the rest of A's eigenvalues, see [192, 48].

  In many cases, one is interested in eigenvalues that are small compared to the norm $\|A\|$, and for such eigenvalues the inequality (9.1) 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, the reader is referred to works, for example, [17, 162, 163, 161, 118].

[1] For the definition of a regular pencil, see §?? and ??

- In general when A is arbitrary, then *up to the first-order* there is an eigenvalue λ of A satisfying

$$|\lambda - \widetilde{\lambda}| \lesssim \frac{\|\widetilde{x}\|_2 \, \|\widetilde{y}\|_2}{|\widetilde{y}^* \widetilde{x}|} \max \left\{ \frac{\|r\|_2}{\|\widetilde{x}\|}, \frac{\|s^*\|_2}{\|\widetilde{y}\|} \right\}. \tag{9.2}$$

  Note that the *individual condition number* c_λ is defined as

$$c_\lambda \equiv \frac{\|x\|_2 \, \|y\|_2}{|y^* x|} = \sec \theta(x, y),$$

  and x and y are the corresponding eigenvector and left eigenvector. But exact x and y usually are not known; for all practical purpose, one could simply estimates this condition number c_λ by $\frac{\|\widetilde{x}\|_2 \, \|\widetilde{y}\|_2}{|\widetilde{y}^* \widetilde{x}|}$. Notice that $c_\lambda \geq 1$, with equality if A is Hermitian.

- If the approximate left eigenvector $\widetilde{y}$ is not available, bounds directly on $|\lambda - \widetilde{\lambda}|$ do not exist in general. This is the case when we have to resort to another tool which is described below.

Further analysis – transfer residual errors to backward errors: It turns out that the computed eigenvalue and eigenvector can always be interpreted as the exact one of nearby matrices, i.e.,

$$(A + E)\widetilde{x} = \widetilde{\lambda}\widetilde{x}$$

and

$$\widetilde{y}^* (A + E) = \widetilde{\lambda}\widetilde{y}^*$$

if $\widetilde{y}$ is available, where the error matrices E are small in norm relatively to $\|A\|$. E will also be assumed Hermitian if A is. 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 direct bounds such as (9.1) and (9.2). But for brevity, we shall not go into detail here. 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, but to the best of our interest, we would like to know only the optimal or nearly optimal error matrices in the sense that certain norms (usually 2-norm $\|\cdot\|_2$ or the Frobenius norm $\|\cdot\|_{\mathrm{F}}$) are minimized among all feasible error matrices. In fact, practical purposes will be served if we can determined the norm sizes 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.*

For the simplicity of the following presentation, we normalize the approximate eigenvectors so that $\|\widetilde{x}\|_2 = 1$ and $\|\widetilde{y}\|_2 = 1$:

- When A is Hermitian, the optimal error matrices E_2 (in 2-norm) and E_{F} (in the Frobenius norm) for which $\widetilde{\lambda}$ and $\widetilde{x}$ are an exact eigenvalue and its corresponding eigenvector of $A + E$, i.e.,

$$(A + E)\widetilde{x} = \widetilde{\lambda}\widetilde{x} \tag{9.3}$$

  for $E = E_2, E_{\mathrm{F}}$, satisfy

$$\|E_2\|_2 = \|r\|_2 \quad \text{and} \quad \|E_{\mathrm{F}}\|_{\mathrm{F}} = \sqrt{2\|r\|_2^2 - (\widetilde{x}^H A \widetilde{x} - \widetilde{\lambda})^2}. \tag{9.4}$$

Here both E_2 and $E_{\rm F}$ are Hermitian. In particular, if $\widetilde{\lambda} = \widetilde{x}^H A\widetilde{x}$, the Rayleigh quotient, then $\|E_{\rm F}\|_{\rm F} = \sqrt{2}\|r\|_2$.

- In general when A is arbitrary, we distinguish two cases.

  1. Only $\widetilde{x}$ is available but $\widetilde{y}$ is not. Then the optimal error matrix E (in both 2-norm and the Frobenius norm) for which $\widetilde{\lambda}$ and $\widetilde{x}$ are an exact eigenvalue and its corresponding eigenvector of $A+E$, i.e.,

  $$(A+E)\widetilde{x} = \widetilde{\lambda}\widetilde{x}, \tag{9.5}$$

  satisfies

  $$\|E\|_2 = \|E\|_{\rm F} = \|r\|_2. \tag{9.6}$$

  2. Both $\widetilde{x}$ and $\widetilde{y}$ are available. Then the optimal error matrices E_2 (in 2-norm) and $E_{\rm F}$ (in the Frobenius norm) for which $\widetilde{\lambda}$, $\widetilde{x}$, and $\widetilde{y}$ are an exact eigenvalue and its corresponding eigenvectors of $A+E$, i.e.,

  $$(A+E)\widetilde{x} = \widetilde{\lambda}\widetilde{x} \quad \text{and} \quad \widetilde{y}^*(A+E) = \widetilde{\lambda}\widetilde{y}^* \tag{9.7}$$

  for $E = E_2, E_{\rm F}$, satisfy

  $$\|E_2\|_2 = \max\{\|r\|_2, \|s^*\|_2\} \tag{9.8}$$

  and

  $$\|E_{\rm F}\|_{\rm F} = \sqrt{\|r\|_2^2 + \|s^*\|_2^2 - (\widetilde{y}^* A\widetilde{x} - \widetilde{\lambda}\widetilde{y}^*\widetilde{x})^2}. \tag{9.9}$$

We now introduce a concept of τ-backward stability. We say the algorithm that delivers the eigenpair $(\widetilde{\lambda}, \widetilde{x})$ is *τ-backward stable* for the pair with respect to the norm $\|\cdot\|$ if it is an exact eigenpair for $A+E$ with $\|E\| \le \tau$; analogously the algorithm that delivers the eigen-triplet $(\widetilde{\lambda}, \widetilde{x}, \widetilde{y})$ is *τ-backward stable* for the triplet with respect to the norm $\|\cdot\|$ if it is an exact eigen-triplet for $A+E$ with $\|E\| \le \tau$. With these and equations (9.3) – (9.8) in mind, statements can be made about the backward stability of the algorithm which computes the eigenpair $(\widetilde{\lambda}, \widetilde{x})$ or the eigentriplet $(\widetilde{\lambda}, \widetilde{x}, \widetilde{y})$.

This τ-backward stability can also be meaningfully defined in a straightforward way for generalized eigenvalue problems to be discussed below. But we shall not do it explicitly.

9.1.3 Generalized eigenvalue problem, I

Let $\widetilde{\lambda}$ denote a computed eigenvalue, and let $\widetilde{x}$ be its corresponding computed eigenvector, i.e.,

$$A\widetilde{x} \approx \widetilde{\lambda}B\widetilde{x},$$

and $\widetilde{y}$ be its corresponding computed left eigenvector, i.e.,

$$\widetilde{y}^* A \approx \widetilde{\lambda}\widetilde{y}^* B.$$

and B is Hermitian positive definite. We assume that $\widetilde{x}$ is available always, but $\widetilde{y}$ may not be. When A is Hermitian, then $\widetilde{y} = \widetilde{x}$.

The residual vectors are defined as

$$r = A\widetilde{x} - \widetilde{\lambda} B\widetilde{x},$$

and

$$s^* = \widetilde{y}^* A - \widetilde{\lambda}\widetilde{y}^* B.$$

We shall also use the vector norm $\|\cdot\|_{B^{-1}}$ defined as

$$\|x\|_{B^{-1}} \equiv (x^* B^{-1} x)^{1/2}.$$

- When A is Hermitian, there is an eigenvalue λ of the matrix pair (A, B) satisfying

$$|\lambda - \widetilde{\lambda}| \leq \frac{\|r\|_{B^{-1}}}{\|B\widetilde{x}\|_{B^{-1}}}. \tag{9.10}$$

- In general when A is arbitrary, up to the first order, there is an eigenvalue λ of the matrix pair (A, B) satisfying

$$|\lambda - \widetilde{\lambda}| \lesssim \frac{\|B\widetilde{x}\|_{B^{-1}} \|B\widetilde{y}\|_{B^{-1}}}{|\widetilde{y}^* B\widetilde{x}|} \max\left\{ \frac{\|r\|_{B^{-1}}}{\|B\widetilde{x}\|_{B^{-1}}}, \frac{\|s\|_{B^{-1}}}{\|B\widetilde{y}\|_{B^{-1}}} \right\}. \tag{9.11}$$

Note that computing $\|r\|_{B^{-1}} = (r^* B^{-1} r)^{1/2}$ and $\|s\|_{B^{-1}} = (s^* B^{-1} s)^{1/2}$ require that linear systems be computed to get $B^{-1}r$ (and $B^{-1}s$). This may not be so bad as it appears since they are for error estimation. Generally crude solutions would be just enough for decent error bounds. There is no linear system solving involved in computing $\|B\widetilde{x}\|_{B^{-1}} = (\widetilde{x}^* B\widetilde{x})^{1/2}$ and $\|B\widetilde{y}\|_{B^{-1}} = (\widetilde{y}^* B\widetilde{y})^{1/2}$.

We now provide a few additional remarks:

1. When $B = I$, (9.10) reduces to (9.1), while (9.11) reduces to (9.2); similarly, the other results reduce to their counterparts for standard eigenproblems.

2. Analogously to the standard eigenvalue problem, the *individual condition number* can also be defined here as

$$c_\lambda \equiv \frac{\|Bx\|_{B^{-1}} \|By\|_{B^{-1}}}{|y^* Bx|} = \sec\theta(B^{1/2}x, B^{1/2}y),$$

where x and y are the corresponding eigenvector and left eigenvector. But again the exact x and y usually are not known; one simply computes this condition number with the exact x and y replaced by their approximations. Note that $c_\lambda \geq 1$, with equality if A is Hermitian.

Further analysis – transfer residual errors to backward errors: It can be shown that the computed eigenvalue and eigenvector(s) are the exact ones of nearby matrices, i.e.,

$$(A + E)\widetilde{x} = \widetilde{\lambda} B\widetilde{x},$$

and

$$\widetilde{y}^*(A + E) = \widetilde{\lambda}\widetilde{y}^* B$$

if $\widetilde{y}$ is available, where error matrices E are small relative to the norm of A. If A is Hermitian, then we will also select a Hermitian E.

For the simplicity of the following presentation, we normalize the approximate eigenvectors so that $\|\widetilde{x}\|_2 = 1$ and $\|\widetilde{y}\|_2 = 1$.

- When A is Hermitian, the optimal error matrices E_2 (in 2-norm) and $E_{\rm F}$ (in the Frobenius norm) for which $\widetilde{\lambda}$ and $\widetilde{x}$ are an exact eigenvalue and its corresponding eigenvector of the pair $(A+E, B)$, i.e.,

$$(A+E)\widetilde{x} = \widetilde{\lambda} B\widetilde{x} \tag{9.12}$$

for $E = E_2, E_{\rm F}$, satisfy

$$\|E_2\|_2 = \|r\|_2 \quad \text{and} \quad \|E_{\rm F}\|_{\rm F} = \sqrt{2\|r\|_2^2 - (\widetilde{x}^* A\widetilde{x} - \widetilde{\lambda}\widetilde{x}^* B\widetilde{x})^2}. \tag{9.13}$$

Here both E_2 and $E_{\rm F}$ are Hermitian.

- In general when A is arbitrary, we also distinguish two cases.

  1. Only $\widetilde{x}$ is available but $\widetilde{y}$ is not. Then the optimal error matrix E (in both 2-norm and the Frobenius norm) for which $\widetilde{\lambda}$ and $\widetilde{x}$ are an exact eigenvalue and its corresponding eigenvector of the pair $\{A+E, B\}$, i.e.,

$$(A+E)\widetilde{x} = \widetilde{\lambda} B\widetilde{x}, \tag{9.14}$$

  satisfies

$$\|E\|_2 = \|E\|_{\rm F} = \|r\|_2. \tag{9.15}$$

  2. Both $\widetilde{x}$ and $\widetilde{y}$ are available. Then the optimal error matrices E_2 (in 2-norm) and $E_{\rm F}$ (in the Frobenius norm) for which $\widetilde{\lambda}$, $\widetilde{x}$, and $\widetilde{y}$ are an exact eigenvalue and its corresponding eigenvectors of the pair $\{A+E, B\}$, i.e.,

$$(A+E)\widetilde{x} = \widetilde{\lambda} B\widetilde{x} \quad \text{and} \quad \widetilde{y}^*(A+E) = \widetilde{\lambda}\widetilde{y}^* B, \tag{9.16}$$

  for $E = E_2, E_{\rm F}$, satisfy

$$\|E_2\|_2 = \max\{\|r\|_2, \|s^*\|_2\} \tag{9.17}$$

  and

$$\|E_{\rm F}\|_{\rm F} = \sqrt{\|r\|_2^2 + \|s^H\|_2^2 - (\widetilde{y}^* A\widetilde{x} - \widetilde{\lambda}\widetilde{y}^* B\widetilde{x})^2}. \tag{9.18}$$

9.1.4 Generalized eigenvalue problem, II

The situation for general regular pairs (A, B) is more complicated, 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 mathematical 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

$$\lambda \equiv \alpha/\beta, \qquad |\alpha|^2 + |\beta|^2 > 0.$$

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 distance*: for $\lambda = \alpha/\beta$ and $\widetilde{\lambda} = \widetilde{\alpha}/\widetilde{\beta}$, their chordal distance is defined as

$$\begin{aligned} \chi(\lambda, \widetilde{\lambda}) &= \chi((\alpha, \beta), (\widetilde{\alpha}, \widetilde{\beta})) \\ &= \frac{|\alpha\widetilde{\beta} - \beta\widetilde{\alpha}|}{\sqrt{|\alpha|^2 + |\beta|^2}\sqrt{|\widetilde{\alpha}|^2 + |\widetilde{\beta}|^2}} \\ &= \frac{|\lambda - \widetilde{\lambda}|}{\sqrt{1 + |\lambda|^2}\sqrt{1 + |\widetilde{\lambda}|^2}}. \end{aligned}$$

We are now ready to address the issue of assessing the accuracy of computed approximations. Let $(\widetilde{\alpha}, \widetilde{\beta})$ denote a computed eigenvalue, and let $\widetilde{x}$ be its corresponding computed eigenvector, i.e.,

$$\widetilde{\beta} A \widetilde{x} \approx \widetilde{\alpha} B \widetilde{x}.$$

Sometimes, its corresponding left computed eigenvector $\widetilde{y}$ is also available:

$$\widetilde{\beta} \widetilde{y}^* A \approx \widetilde{\alpha} \widetilde{y}^* B.$$

Let

$$r = \widetilde{\beta} A \widetilde{x} - \widetilde{\alpha} B \widetilde{x},$$

and

$$s^* = \widetilde{\beta} \widetilde{y}^* A - \widetilde{\alpha} \widetilde{y}^* B$$

denote the corresponding residual vectors. Then it can be proved that up to the first order there is an eigenvalue (α, β) of the pair satisfying

$$\chi((\alpha, \beta), (\widetilde{\alpha}, \widetilde{\beta})) \lesssim \frac{\|\widetilde{x}\|_2 \|\widetilde{y}\|_2}{\sqrt{|\widetilde{y}^* A \widetilde{x}|^2 + |\widetilde{y}^* B \widetilde{x}|^2}} \max\left\{ \frac{\|r\|_2}{\|\widetilde{x}\|_2}, \frac{\|s^*\|_2}{\|\widetilde{y}\|_2} \right\} \frac{1}{\sqrt{|\widetilde{\alpha}|^2 + |\widetilde{\beta}|^2}}. \quad (9.19)$$

Based on this, the *individual condition number* $c_{(\alpha,\beta)}$ is defined as

$$c_{(\alpha,\beta)} = \frac{\|x\|_2 \|y\|_2}{\sqrt{|y^* A x|^2 + |y^* B x|^2}},$$

which in actual computation will be estimated with x and y replaced by their approximations $\widetilde{x}$ and $\widetilde{y}$.

Further analysis – 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.,

$$\widetilde{\beta}(A + E)\widetilde{x} = \widetilde{\alpha}(B + F)\widetilde{x},$$

and

$$\widetilde{\beta}\widetilde{y}^*(A+E) = \widetilde{\alpha}\widetilde{y}^*(B+F)$$

if $\widetilde{y}$ is available, where error matrices E and F are small relative to the norms of A and B.

For the simplicity of the following presentation, we normalize the approximate eigenvectors so that $\|\widetilde{x}\|_2 = 1$ and $\|\widetilde{y}\|_2 = 1$.

1. Only $\widetilde{x}$ is available but $\widetilde{y}$ is not. Then the optimal error matrix (E, F) (in both 2-norm and the Frobenius norm) for which $(\widetilde{\alpha}, \widetilde{\beta})$ and $\widetilde{x}$ are an exact eigenvalue and its corresponding eigenvector of the pair $(A+E, B+F)$, i.e.,

$$\widetilde{\beta}(A+E)\widetilde{x} = \widetilde{\alpha}(B+F)\widetilde{x}, \tag{9.20}$$

   satisfies

$$\|(E,F)\|_2 = \|(E,F)\|_{\mathrm{F}} = \|r\|_2/\sqrt{|\widetilde{\alpha}|^2 + |\widetilde{\beta}|^2}. \tag{9.21}$$

2. Both $\widetilde{x}$ and $\widetilde{y}$ are available. Then the optimal error matrices (E_2, F_2) (in 2-norm) and $(E_{\mathrm{F}}, F_{\mathrm{F}})$ (in the Frobenius norm) for which $(\widetilde{\alpha}, \widetilde{\beta})$, $\widetilde{x}$, and $\widetilde{y}$ are an exact eigenvalue and its corresponding eigenvectors of the pair $(A+E, B+F)$, i.e.,

$$\widetilde{\beta}(A+E)\widetilde{x} = \widetilde{\alpha}(B+F)\widetilde{x} \quad \text{and} \quad \widetilde{\beta}\widetilde{y}^*(A+E) = \widetilde{\alpha}\widetilde{y}^*(B+F) \tag{9.22}$$

   for $(E, F) = (E_2, F_2), (E_{\mathrm{F}}, F_{\mathrm{F}})$, satisfy

$$\|(E_2, F_2)\|_2 = \frac{\max\{\|r\|_2, \|s^H\|_2\}}{\sqrt{|\widetilde{\alpha}|^2 + |\widetilde{\beta}|^2}} \tag{9.23}$$

   and

$$\|(E_{\mathrm{F}}, F_{\mathrm{F}})\|_{\mathrm{F}} = \frac{\sqrt{\|r\|_2^2 + \|s^H\|_2^2 - (\widetilde{\beta}\widetilde{y}^H A\widetilde{x} - \widetilde{\alpha}\widetilde{y}^H B\widetilde{x})^2}}{\sqrt{|\widetilde{\alpha}|^2 + |\widetilde{\beta}|^2}}. \tag{9.24}$$

9.1.5 Errors in Computed Eigenvectors

It is well-known that each individual eigenvector associated with eigenvalue clusters are very sensitive to perturbations, and consequently such an eigenvector can not be accurately computed in general. Fortunately, it is often that for the situation of an eigenvalue cluster, it is the entire associated eigenspace that is of practical importance, and that eigenspace can be computed with satisfactory accuracy.

Instead of presenting available bounds which can be complicated and difficult to digest for nonsymmetric eigenvalue problems, in this subsection we shall give a brief actual analysis which will bring out the gist of what may contribute to the sensitivity of eigenvectors. Detailed treatment is beyond the scope of this book and the interested reader is referred to the literature, for example, see [239].

Let us start by looking at a Hermitian matrix A. As usual, suppose we have an approximate eigenpair $(\widetilde{\lambda}, \widetilde{x})$ to an exact pair (λ, x), and define the residual vector $r = A\widetilde{x} - \widetilde{\lambda}\widetilde{x}$. Let $A = Q\Lambda Q^*$ be its eigendecomposition, where $Q = [x, Q_2]$ is unitary and Λ is diagonal with λ at the $(1,1)$ position. We have

$$Q^*r = \Lambda Q^*\widetilde{x} - \widetilde{\lambda}Q^*\widetilde{x} = (\Lambda - \widetilde{\lambda}I)Q^*\widetilde{x}.$$

It can be seen that $\|Q_2^*\widetilde{x}\|_2 = \sin\theta(x,\widetilde{x})$. So the second to the last components of the above equation imply

$$\|r\|_2 = \|Q^*r\|_2 \geq \min_{\lambda\neq\mu\in\lambda(A)} |\mu-\widetilde{\lambda}|\,\|Q_2^*\widetilde{x}\|_2$$

which shows

$$\sin\theta(x,\widetilde{x}) \leq \frac{\|r\|_2}{\min_{\lambda\neq\mu\in\lambda(A)}|\mu-\widetilde{\lambda}|}. \tag{9.25}$$

This inequality says that $\sin\theta(x,\widetilde{x})$ can be large if there is another eigenvalue other than λ of A that comes very close to $\widetilde{\lambda}$.

Now for non-Hermitian A, first we consider the case of having an approximate eigenpair $(\widetilde{\lambda},\widetilde{x})$ to an exact pair (λ,x). Let $A = QTQ^*$ be the Schur decomposition of A, see §??, where $Q=[x,Q_2]$ is unitary and

$$T = \begin{bmatrix} \lambda & a^* \\ 0 & A_1 \end{bmatrix}.$$

Similarly, from the residual vector $r = A\widetilde{x}-\widetilde{\lambda}\widetilde{x} = QTQ^H\widetilde{x}-\widetilde{\lambda}\widetilde{x}$, we have

$$Q^*r = TQ^*\widetilde{x}-\widetilde{\lambda}Q^*\widetilde{x} = (T-\widetilde{\lambda}I)Q^*\widetilde{x}.$$

Again the second to the last components of the above equation imply

$$\|r\|_2 = \|Q^*r\|_2 \geq \|(A_1-\widetilde{\lambda}I)Q_2^*\widetilde{x}\|_2 \geq \|(A_1-\widetilde{\lambda}I)^{-1}\|_2^{-1}\|Q_2^*\widetilde{x}\|_2$$

which shows

$$\sin\theta(x,\widetilde{x}) \leq \|(A_1-\widetilde{\lambda}I)^{-1}\|_2\|r\|_2. \tag{9.26}$$

This inequality reveals another potential factor that makes $\sin\theta(x,\widetilde{x})$ large. This occurs then A_1 is too far from the set of normal matrices in which case $\|(A_1-\widetilde{\lambda}I)^{-1}\|_2$, the reciprocal of the smallest singular value of $A_1-\widetilde{\lambda}I$, can get huge even though none of the rest of A's eigenvalues comes close to $\widetilde{\lambda}$.

9.2 Balancing Matrices

T. Chen and J. Demmel

9.2.1 Introduction

Numerical algorithms that compute the eigenvalues of a nonsymmetric matrix A typically have backward errors of the magnitude of $\varepsilon\|A\|$, where ε is the machine precision. Therefore, applying a simple and accurate similarity transform DAD^{-1}, which reduces the norm of the matrix A, or which reduces the condition numbers of some subset of A's eigenvalues, before computing the eigenvalues of A can be advantageous.

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 first noted that the norm of a matrix A can often be reduced with a similarity transform of the form

$$B = DAD^{-1} \tag{9.27}$$

where D is a diagonal matrix [184]. 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 α-norm if for any i, the α-norm of row i is the same as the α-norm of column i. **have we defined what's α-norm ?**

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. Focusing on practice instead of theory, we present here two styles of algorithms for balancing sparse matrices. The algorithms are analyzed more thoroughly in [28] and [29]; Software is available at

`http://www.cs.berkeley.edu/~tzuyi/Balancing`.

9.2.2 Direct Balancing

The traditional dense balancing algorithm, described fully in [191] and referred to here as GEBAL, as available in LAPACK consists of two phases. 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 rows and columns — each row/column 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 §9.3. 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 tolosa matrix [**?**], 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 GEBAL, 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 on the sparse data structures the same balancing algorithm used in GEBAL.

9.2.3 Balancing Algorithms Which Use Krylov Information

Direct balancing algorithms such as GEBAL and SPBALANCE need to calculate exact row and column norms, making them inappropriate for sparse matrices whose entries are not given explicitly. In [28] and [29] we describe three new balancing algorithms which use only Krylov information, matrix-vector and matrix-transpose-vector multiplies, to access the original matrix. The following is pseudocode for the algorithm KRYLOV-CUTOFF, which performs best of all three Krylov algorithms on our test matrices.

ALGORITHM 9.40: KRYLOVCUTOFF

```
(1)     normA = ||A||_∞
(2)     for j = 1, 2, ..., t
(3)         z = a length n vector of random ±1s
(4)         p = Az
(5)         r = A^T z
(6)         for i = 1, 2, ..., n
(7)             if (|p(i)| ≤ cutoff · normA) or (r(i) == 0)
(8)                 D(i) = 1
(9)             else
(10)                D(i) = sqrt(|r(i)/p(i)|)
(11)            end if
(12)        end for
(13)        A = DAD^{-1}
(14)    end for
```

If A is not given explicitly, black box functions for computing Az and $A^T z$ are necessary. (See [29] for a description of a Krylov algorithm which uses only Az and not $A^T z$.) Lines (4) and (5) then compute $p = DAD^{-1}z$ and $r = D^{-1}A^T Dz$; line (10) updates D. Furthermore, in line (1) $||A||_\infty$ can be approximated by multiplying A with a vector of random ± 1s and taking the largest component of the absolute value of the result.

The number of iterations and the cutoff value are left to the user as we know neither the right stopping criteria nor how to choose the best cutoff value. Based on experimental evidence, we chose default values of 5 for the number of iterations and 10^{-8} for the cutoff.

9.2.4 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 9.1 and 9.2 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 from the dotted line, the greater the difference in eigenvalue accuracy with and without balancing.

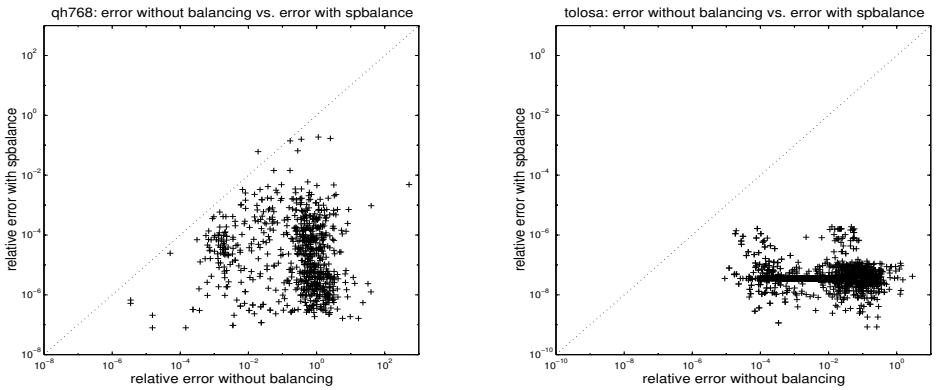


FIGURE 9.1: *Plots examining the relative accuracy of eigenvalues computed with and without direct balancing for the qh768 and tolosa matrices.*

Figure 9.1 clearly shows SPBALANCE is effective in improving the accuracy to which most eigenvalues of the qh768 and tolosa matrices are computed.

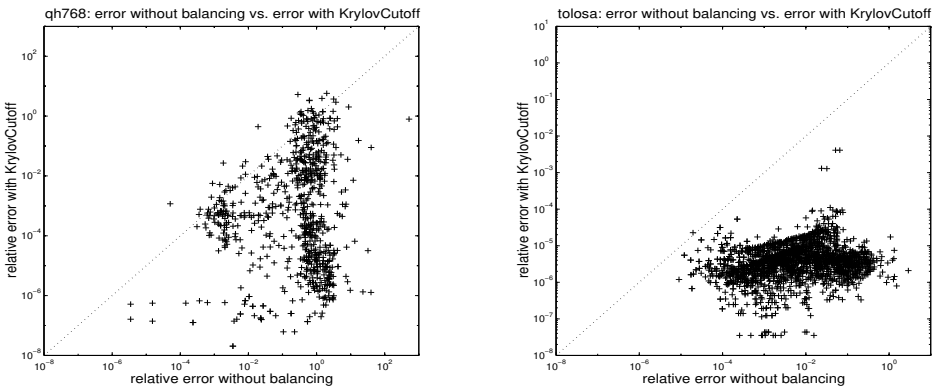


FIGURE 9.2: *Plots examining the relative accuracy of eigenvalues computed with and without Krylov balancing for the qh768 and tolosa matrices.*

Figure 9.2 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 9.2, after Krylov-based balancing, the eigenvalues are computed using a dense eigensolver. In practice, if a sparse matrix is given explicitly and one is willing to

run an $O(n^3)$ dense eigensolver, one could precondition the matrix using SPBALANCE and not a Krylov-based method. Therefore, a potentially more practical use of Krylov algorithms is as preconditioners for sparse eigensolvers which similarly access the matrix only through matrix-vector multiplications.

Figures 9.3 and 9.4 show the relative accuracy of the 10 largest and smallest eigenvalues when calculated without balancing, with SPBALANCE, and with three variants of Krylov balancing (KRYLOVAZ, which uses only Az, and KRYLOVATZ, which uses both Az and $A^T z$, are the two other Krylov-based techniques, both described in [29]). The eigenvalues were computed in Matlab using the EIGS function.

Figure 9.3 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 largest eigenvalues SPBALANCE is most effective, though all three Krylov balancing algorithms do significantly better than no balancing at all.

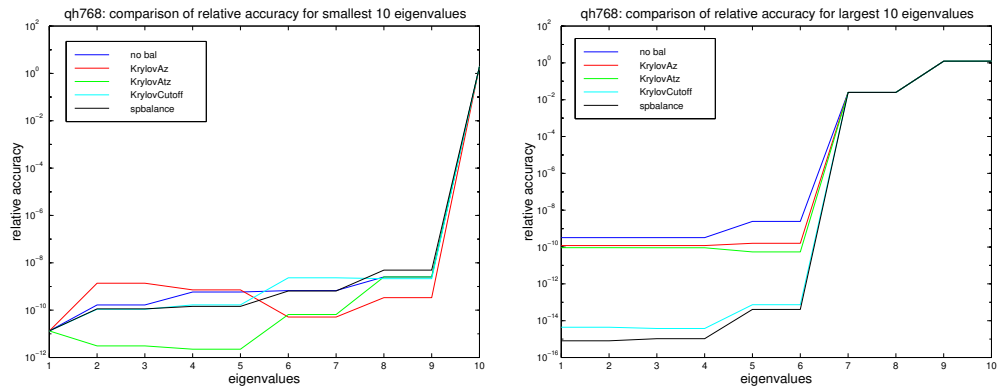


FIGURE 9.3: *Plots comparing the relative accuracy of the largest and smallest (in magnitude) eigenvalues computed with the different Krylov-based balancing algorithms for the qh768 matrix. We use the default settings for the number of iterations and the cutoff value.*

9.3 Data Structures

J. Dongarra

The efficiency of any of the methods considered in previous sections is determined primarily by the performance of the matrix-vector product and therefore on the storage scheme used for the matrix. Since the methods are typically used on sparse matrices, we will review here a number of sparse storage formats. 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 are used in numerical software packages such as `ITPACK` [?] and `NSPCG` [?]. After surveying the various formats, we demonstrate how the matrix-vector product is formulated using two of the sparse matrix formats.

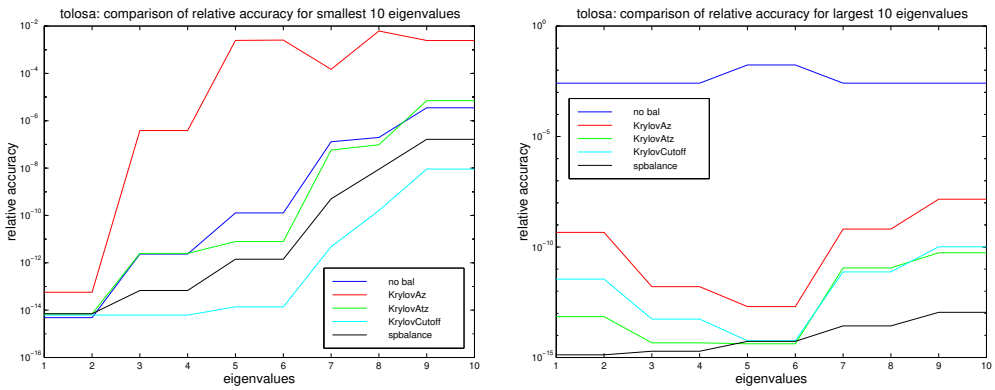


FIGURE 9.4: *Plots comparing the relative accuracy of the largest and smallest (in magnitude) eigenvalues computed with the different Krylov-based balancing algorithms for the tolosa matrix. We use the default settings for the number of iterations and the cutoff value.*

9.3.1 Sparse Matrix Storage Formats

If the coefficient matrix A is sparse, large-scale eigenvalue problem of the form $Ax = \lambda x$ can be most efficiently solved if the zero elements of A are not 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 [?] and Eijkhout [?]). Here we will discuss Compressed Row and Column Storage, Block Compressed Row Storage, Diagonal Storage, Jagged Diagonal Storage, and Skyline Storage.

9.3.1.1 Compressed Row Storage (CRS)

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 3 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 $\texttt{val(k)} = a_{i,j}$ then $\texttt{col_ind(k)} = j$. The `row_ptr` vector stores the locations in the `val` vector that start a row, that is, if $\texttt{val(k)} = a_{i,j}$ then $\texttt{row_ptr(i)} \leq k < \texttt{row_ptr(i+1)}$. By convention, we define $\texttt{row_ptr(n+1)} = nnz + 1$, 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

$$A = \begin{bmatrix} 10 & 0 & 0 & 0 & -2 & 0 \\ 3 & 9 & 0 & 0 & 0 & 3 \\ 0 & 7 & 8 & 7 & 0 & 0 \\ 3 & 0 & 8 & 7 & 5 & 0 \\ 0 & 8 & 0 & 9 & 9 & 13 \\ 0 & 4 & 0 & 0 & 2 & -1 \end{bmatrix} . \tag{9.28}$$

The CRS format for this matrix is then specified by the arrays {`val`, `col_ind`, `row_ptr`} given below

`val`	10	-2	3	9	3	7	8	7	$3 \cdots 9$	13	4	2	-1
`col_ind`	1	5	1	2	6	2	3	4	$1 \cdots 5$	6	2	5	6

`row_ptr`	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 trade-off is a more complicated algorithm with a somewhat different pattern of data access.

9.3.1.2 Compressed Column Storage (CCS)

Analogous to Compressed Row Storage there is Compressed Column Storage (CCS), which is also called the *Harwell-Boeing* sparse matrix format [**?**]. 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 (9.28) is given by

`val`	10	3	3	9	7	8	4	8	$8 \cdots 9$	2	3	13	-1
`row_ind`	1	2	4	2	3	5	6	3	$4 \cdots 5$	6	2	5	6

`col_ptr`	1	4	8	10	13	17	20

.

9.3.1.3 Block Compressed Row Storage (BCRS)

If the sparse matrix A is comprised 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 3 arrays for the BCRS format: a rectangular array for floating-point numbers (`val`$(1 : nnzb, 1 : n_b, 1 : n_b)$) 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`$(1 : n_d + 1)$) whose entries point to the beginning of each block row in `val(:,:,:)` and `col_ind(:)`. The savings in storage locations and reduction in indirect addressing for BCRS over CRS can be significant for matrices with a large n_b.

9.3.1.4 Compressed Diagonal Storage (CDS)

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, 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 [?], which unlike 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

$$A = \begin{bmatrix} 10 & -3 & 0 & 0 & 0 & 0 \\ 3 & 9 & 6 & 0 & 0 & 0 \\ 0 & 7 & 8 & 7 & 0 & 0 \\ 0 & 0 & 8 & 7 & 5 & 0 \\ 0 & 0 & 0 & 9 & 9 & 13 \\ 0 & 0 & 0 & 0 & 2 & -1 \end{bmatrix} . \tag{9.29}$$

Using the CDS format, we store this matrix A in an array of dimension `(6,-1:1)` using the mapping

$$\texttt{val(i,j)} = a_{i,i+j}. \tag{9.30}$$

Hence, the rows of the `val(:,:)` array are

`val(:,-1)`	0	3	7	8	9	2
`val(:, 0)`	10	9	8	7	9	-1
`val(:,+1)`	-3	6	7	5	13	0

.

Notice the two zeros corresponding to non-existing matrix elements.

A generalization of the CDS format more suitable for manipulating general sparse matrices on vector supercomputers is discussed by Melhem in [?]. This variant of CDS uses a *stripe* data structure to store the matrix A. This structure is more efficient in storage in the case of varying bandwidth, but it makes the matrix-vector product slightly more expensive, as it involves a gather operation.

As defined in [?], a stripe in the $n \times n$ matrix A is a set of positions $S = \{(i, \sigma(i));\ i \in I \subseteq I_n\}$, where $I_n = \{1, \ldots, n\}$ and σ is a strictly increasing function. Specifically, if

$(i, \sigma(i))$ and $(j, \sigma(j))$ are in S, then

$$i < j \rightarrow \sigma(i) < \sigma(j).$$

When computing the matrix-vector product $y = Ax$ using stripes, each $(i, \sigma_k(i))$ element of A in stripe S_k is multiplied with both x_i and $x_{\sigma_k(i)}$ and these products are accumulated in $y_{\sigma_k(i)}$ and y_i, respectively. For the nonsymmetric matrix A defined by

$$A = \begin{bmatrix} 10 & -3 & 0 & 1 & 0 & 0 \\ 0 & 9 & 6 & 0 & -2 & 0 \\ 3 & 0 & 8 & 7 & 0 & 0 \\ 0 & 6 & 0 & 7 & 5 & 4 \\ 0 & 0 & 0 & 0 & 9 & 13 \\ 0 & 0 & 0 & 0 & 5 & -1 \end{bmatrix}, \tag{9.31}$$

the 4 stripes of the matrix A stored in the rows of the `val(:,:)` array would be

`val(:,-1)`	0	0	3	6	0	5
`val(:, 0)`	10	9	8	7	9	-1
`val(:,+1)`	0	-3	6	7	5	13
`val(:,+2)`	0	1	-2	0	4	0

.

9.3.1.5 Jagged Diagonal Storage (JDS)

The Jagged Diagonal Storage format can be useful for the implementation of iterative methods on parallel and vector processors (see Saad [211]). Like the Compressed Diagonal format, it gives a vector length essentially of the size of 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. In the matrix from (9.31) all elements are shifted left:

$$\begin{bmatrix} 10 & -3 & 0 & 1 & 0 & 0 \\ 0 & 9 & 6 & 0 & -2 & 0 \\ 3 & 0 & 8 & 7 & 0 & 0 \\ 0 & 6 & 0 & 7 & 5 & 4 \\ 0 & 0 & 0 & 0 & 9 & 13 \\ 0 & 0 & 0 & 0 & 5 & -1 \end{bmatrix} \longrightarrow \begin{bmatrix} 10 & -3 & 1 & \\ 9 & 6 & -2 & \\ 3 & 8 & 7 & \\ 6 & 7 & 5 & 4 \\ 9 & 13 & & \\ 5 & -1 & & \end{bmatrix}$$

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 `val(:,:)`, an array of column indices, `col_ind(:,:)` is also stored:

`val`$(:, 1)$	10	9	3	6	9	5
`val`$(:, 2)$	-3	6	8	7	13	-1
`val`$(:, 3)$	1	-2	7	5	0	0
`val`$(:, 4)$	0	0	0	4	0	0

;

`col_ind`$(:, 1)$	1	2	1	2	5	5
`col_ind`$(:, 2)$	2	3	3	4	6	6
`col_ind`$(:, 3)$	4	5	4	5	0	0
`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 (`perm(1:n)`) which reorders the rows, a floating-point array (`jdiag(:)`) containing the jagged diagonals in succession, an integer array (`col_ind(:)`) containing the corresponding column indices indices, and finally a pointer array (`jd_ptr(:)`) whose elements point to the beginning of each jagged diagonal. The advantages of JDS for matrix multiplications are discussed by Saad in [211].

The JDS format for the above matrix A in using the linear arrays {`perm`, `jdiag`, `col_ind`, `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

.

9.3.1.6 Skyline Storage (SKS)

The final storage scheme we consider is for skyline matrices, which are also called variable band or profile matrices (see Duff, Erisman and Reid [?]). 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 9.5, we store the lower triangular elements in 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 [?], 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.

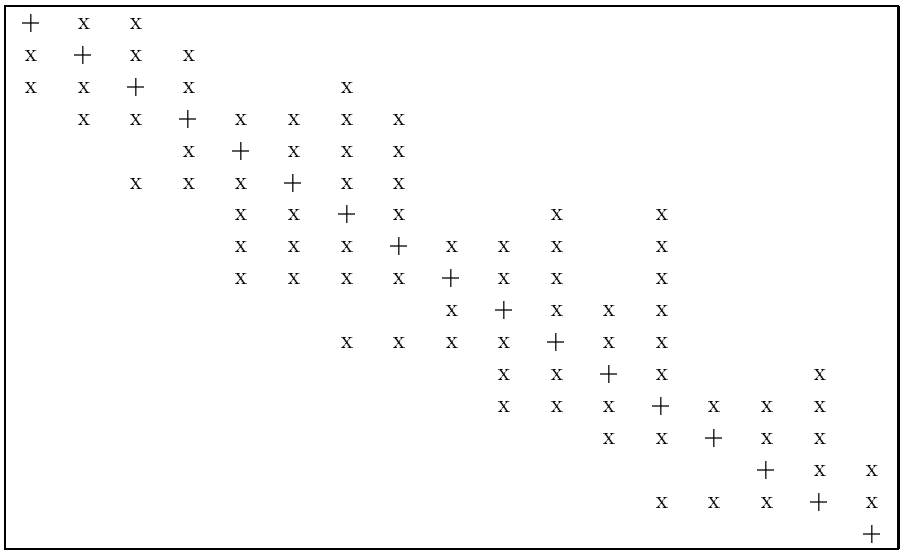


FIGURE 9.5: **Profile of a nonsymmetric skyline or variable-band matrix.**

9.3.2 Matrix vector products

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

$$y = Ax \qquad \text{and} \qquad y = A^T x.$$

We will present these algorithms for two of the storage formats from §9.3: CRS and CDS.

9.3.2.1 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

$$\text{for all } j, \text{ do for all } i: \qquad y_i \leftarrow y_i + a_{j,i} x_j .$$

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, 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.

9.3.2.2 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 jth 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 `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

$$1 \leq \mathtt{i}, \mathtt{i} + \mathtt{j} \leq n .$$

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.

9.4 Dense and Sparse BLAS

J. Dongarra

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 [?]: for vector operations, such as $y \leftarrow \alpha x + y$

Level 2 BLAS [?]: for matrix-vector operations, such as $y \leftarrow \alpha Ax + \beta y$

Level 3 BLAS [?]: 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 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.

9.5 A Brief Survey of Direct Linear Solvers

J. Demmel

9.6 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 ?? 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 imbedded 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 the *Templates for the solution of linear systems* [18]. 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 to solve 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 ?? 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 $K^i(A; r_0)$ 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 vectors x_0, with residual vector $r_0 = b - Ax_0$, is defined as the subspace spanned by the vectors $\{r_0, Ar_0, A^2r_0, ..., A^{i-1}r_0\}$.

The different methods can be classified as follows:

(a) If A is symmetric positive definite, then the *Conjugate Gradient* method [110] generates, using two 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 $K^i(A; r_0)$.

(b) If A is only symmetric but not positive definite, then the *Lanczos* [153] and the *MINRES* method [188] may be considered. In MINRES, the $x_i \in K^i(A; r_0)$ is determined for which the two-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.

(c) If A is unsymmetric, it is in general not possible to determine an optimal $x_i \in K^i(A, r_0)$ with short recurrences. This was proved in [76]. However, with short

recurrencies as in Conjugate Gradients (and MINRES), we can compute the $x_i \in K^i(A; r_0)$, for which $b - Ax_i \perp K^i(A^T; s_0)$ (usually, one selects $s_0 = r_0$). This leads to the *Bi-Conjugate Gradient method* [79]. A clever variant is QMR [87] that has smoother convergence behaviour and is more robust than Bi-Conjugate Gradients.

(d) If A is unsymmetric, then we can compute the $x_i \in K^i(A, r_0)$, for which the residual is minimized in the Euclidean norm. This is done by the GMRES method [214]. This requires i inner products at the i-th iteration step, as well as i vector updates, which meand 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 $< x, A^T y >$ equals $< Ax, y >$, where $< ... >$ represents the inner-product computation. Since the function of the multiplications by A^T in Bi-CG 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 CGS [233], Bi-CGSTAB [247], Bi-CGSTAB(ℓ) [225], TFQMR [89], and hybrids of QMR [25].

(f) For indefinite systems it may also be effective to apply the conjugate gradients method for the normal equations $A^T Ax = A^T b$. Carrying this out in a straightforward manner may lead to numerical instabilities, because $A^T A$ has a squared condition number. A clever and robust implementation is provided in LSQR [189].

Many of these methods have been described in the Templates for linear systems [18], and software for them is available. See [18] for guidelines on how to obtaiin the software. For some methods, descriptions in Templates-style have been given in [58]. That book also contains an overview on various preconditioning approaches.

9.7 Parallelism

J. Dongarra

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 $A^T p^{(i)}$),

9.7.1 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 or 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.

9.7.2 Vector updates

Vector updates are trivially parallelizable: each processor updates its own segment.

9.7.3 Matrix-vector products

The matrix–vector products are often easily parallelized on shared-memory machines by splitting the matrix in strips 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. For more detailed discussions on implementation aspects for distributed memory systems, see De Sturler [?] and Pommerell [?].

CHAPTER 10

Preconditioning Techniques

10.1 Introduction

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. However, much work has been done in Russian as will be mentioned in section 10.3. Ruhe [206, 203, 204] used SOR and the conjugate gradient method to find eigenvalues in the 1970's. Around the same time, quantum chemists, including Davidson [39], 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 [178]. Another development, the Jacobi-Davidson method, has been presented in detail in earlier chapters. A number of other approaches are mentioned in this chapter. 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 methods versus unpreconditioned. And finally, connections are given between various preconditioned eigenvalue methods.

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 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, if computing the eigenvectors is required, and if the matrix is nonsymmetric (and thus more difficult for Krylov methods). 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.

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 similtaneously. Sometimes a block approach is used to alleviate this problem for a preconditioning method. Another similarity among preconditioning methods is that global convergence is trickier. Generally they 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 between 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

$$M^{-1}(A-\nu I), \tag{10.1}$$

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. The four methods discussed are the preconditioned power method, Rayleigh quotient iteration, Davidson's method, and Jacobi-Davidson. The preconditioned power method is Algorithm 10.7 in section 10.3. 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 (10.1). The other algorithms in section 10.3 have something similar as operator.

Next, the Rayleigh quotient iteration (RQI) with preconditioned conjugate gradient method (PCG) 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 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 (10.1). So if the same preconditioning is used, RQI (with PCG) and Davidson's method have the same operator at the core of the method. The difference is that Davidson's method changes ν at every step, while RQI has ρ fixed during a run of PCG, and also Davidson's 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. What is interesting is that the operator (10.1) shows up twice. First, in the outer iteration, from the Jacobi-Davidson correction equation (3.50), 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 (10.1), 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 implemention 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 transformations (and Jacobi-Davidson) 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. And this depends on how good a preconditioner M is.

Davidson's 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 vector as right-hand side of the linear equations, instead of the vector itself.

The two sections that follow both look at some of the details of preconditioning eigenvalue problems, but with some different insights.

10.2 Inexact Methods

K. Meerbergen and R. Morgan

The spectral transformation Lanczos method for the Hermitian eigenvalue problem and the shift-invert Arnoldi method for the non-Hermitian eigenvalue problem 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-invert Arnoldi method and gradually move towards the (Jacobi) Davidson method and inexact rational Krylov method. First, in §10.2.1, we recall the (exact) matrix transformations used and give their main properties. In §10.2.2, we introduce the notion of inexact transformation [168] and explain its importance. In §10.2.3 and §10.2.4, we give some results for the Rayleigh-Ritz procedure with inexact transformations. This includes results for nonsymmetric problems (Arnoldi [168], Davidson [176]) and symmetric problems (Lanczos [179], Davidson [178, 31]) and the Jacobi-Davidson method. We can also use the matrix recurrence relation of the rational Krylov method [?] for computing eigenvalues, even when linear systems are solved inaccurately. This is studied in §10.2.7.

10.2.1 Matrix Transformations

Consider the eigenvalue problem $Ax = \lambda Bx$. The spectral transformation or shift-and-invert transformation is defined by

$$T_{\mathrm{SI}} = (A - \mu B)^{-1} B,$$

where μ is the shift or pole. If $Ax = \lambda Bx$ then $T_{\mathrm{SI}} x = \gamma x$ 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 $T_C x = \zeta x$ with $\zeta = (\lambda - \mu)^{-1}(\lambda - \nu)$. Since $T_C = I + (\mu - \nu) T_{\mathrm{SI}}$ and Krylov spaces are shift-invariant with respect to the matrix, we have that

$$\mathcal{K}^k(T_C, v_1) = \mathcal{K}^k(T_{\mathrm{SI}}, v_1) \ ,$$

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.

10.2.2 Inexact matrix transformations

The analysis is given for the Cayley transform only, though shift-invert could be used as well. The reasons are threefold. First, the shift-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 $j = 1, \ldots, k$, where s_j is the residual and v_j the last Krylov basis vector. The result w_j is orthonormalized against $v_1, \ldots, v_j$ 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

$$(A - \mu B) w_j = (A - \nu B) y_j - s_j, \tag{10.2}$$

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 (10.2) 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 [213] are typically used. GMRES [214], BiCGSTAB(ℓ) [226], and QMR [87] are among those most widely used. See [18] 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 $j = 1, \ldots, k-1$ together in $S_{k-1} = [s_1, \ldots, s_{k-1}]$, w_j in W_{k-1} and y_j in Y_{k-1}, we obtain

$$(A - \mu B) W_{k-1} = (A - \nu B) Y_{k-1} - S_{k-1} \ . \tag{10.3}$$

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

$$(A - \mu B + E_{k-1}) W_{k-1} = (A - \nu B) Y_{k-1} \ . \tag{10.4}$$

This is the relation that we should obtain after $k-1$ steps with the 'inexact' Cayley transform

$$T_{\mathrm{IC}} = (A - \mu B + E_{k-1})^{-1} (A - \nu B) \ .$$

Note that for a fixed ν and μ, T_{IC} depends on k and the way each of the linear systems (10.2) are solved. When (10.2) is solved by a preconditioner M or a stationary linear system solver, i.e. $w_j = M^{-1}(A - \nu B) y_j$ for $j = 1, \ldots, k-1$, 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 $\mathrm{span}(v_1, \ldots, v_k)$. In Arnoldi's method, this happens by the Hessenberg matrix that arises from the orthogonalization of W_{k-1} ; in Davidson's method, one uses the projection with A and B, e.g. $V_k^* A V_k z = \theta V_k^* B V_k z$.

$\mathrm{Span}(v_1, \ldots, v_k)$ 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 [168], 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 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 (10.2) for $j = 1, \ldots, k-1$ defines a Krylov subspace $\mathcal{K}_k(T_{\mathrm{IC}}, y_1)$.

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 §10.2.3 and §10.2.4, we discuss the Rayleigh-Ritz technique, i.e. eigenpairs are computed from the orthogonal projection of $Ax = \lambda Bx$ on the span$(v_1, \ldots, v_k)$. In §10.2.7, the eigenpairs are computed from the eigenpairs of T_{IC} directly by use of the rational Krylov recurrence relation. §10.2.6 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 Davidson's method. Since $T_{\mathrm{C}} = I$ 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 §10.2.5.

10.2.3 The inexact Cayley transform Arnoldi method

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 (10.2) is solved by a preconditioner M or a stationary linear system solver such that $w_j = M^{-1}(A - \theta B)y_j$ for $j = 1, \ldots, k-1$. Thus we can use the Arnoldi method to build the Krylov space of $T_{\mathrm{IC}} \equiv M^{-1}(A - \theta B)$.

ALGORITHM 10.41: **Inexact-Cayley-Arnoldi**

(1) *Given v_1, $\|v_1\| = 1$ and choose $\nu \neq \mu$.*
(2) *For $j = 1, \ldots, k-1$ do :*
(3) *Solve $Mw_j = (A - \nu B)v_j$.*
(4) *Orthonormalize w_j against $v_1, \ldots, v_j$ into v_{j+1}.*
(5) *Compute the projected eigenpairs from*
$V_k^* A V_k z = \theta V_k^* B V_k z$.
(6) *Select a Ritz pair $(\theta, x = V_k z)$.*
(7) *Convergence check : if $\|Ax - \theta Bx\| \leq TOL$ then 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 $T_{\rm IC} = M^{-1}(A - \lambda_1 B) \approx (A - \mu B)^{-1}(A - \lambda_1)$. As for the shift-invert transformation, 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 [168], the linear systems are solved with preconditioners such as incomplete factorizations and multigrid. The numerical examples show faster convergence of Algorithm 10.41 when μ lies a bit away from the spectrum.

10.2.4 The Davidson method

As mentioned, Algorithm 10.41 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.

Davidson's method [39] 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 [178], Davidson's 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 $M = (D - \theta I)^{-1}$.

ALGORITHM 10.42: **Generalized Davidson's Method**

(1) *Choose a preconditioner M which may either be fixed or changing. Begin with k orthonormal starting vectors $v_1, v_2, \ldots, v_k$, and let $j = k$.*

(2) *Compute (y, θ), the best approximation from $Span\{v_1, v_2, \ldots, v_j\}$ 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 $v_1, \ldots, v_j$ to form v_{j+1}. Let $j = j + 1$ and go to 2.*

We next discuss the asymptotic convergence of Davidson's method. Observe that if θ and M were fixed, Davidson's 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 $T_{\rm IC} \equiv M^{-1}(A - \lambda B)$. As mentioned in the discussion around Theorem 10.1, $T_{\rm IC}$ has the correct eigenvector, namely x, and the corresponding eigenvalue of $T_{\rm IC}$ is 0. If the other eigenvalues of $T_{\rm IC}$ 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 $T_{\rm IC}$ is really an improvement over that of the original eigenvalue problem. There is potential, because

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 1. Here A is symmetric and roughly models matrices from quantum chemistry. B is the identity matrix. The main diagonal of A has elements $1, 2, 3, \ldots, 100$ 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, ..., 101.58. So we let $\lambda_1 = -0.323$. The eigenvalues of $T_{\mathrm{IC}} = (D - \lambda_1 I)^{-1}(A - \lambda_1 I)$ are 0.0, 0.263, 0.387, 0.500, ..., 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 2. Consider the model problem from finite difference descretization 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 [171] 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 [106], convergence is $O(n^{1/4})$. It is shown in [174] 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 10.1 gives number 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 that 10^{-8}. No restarting is used. For large problems, MIC(0) preconditioning is 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 Davidson's method targets one eigenvalue at a time. IC(0)

TABLE 10.1: **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

uses 159 iterations, MIC(0) requires 158, and Lanczos 172. However is 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 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 §10.2.6. 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 10.2. 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 [178] and [82], 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 §10.1, another possibility is to use a Krylov subspace method in an initial phase to generate starting vectors for the Davidson method.

10.2.5 The Jacobi-Davidson method and the Cayley transform

The Jacobi-Davidson method is discussed in §§5.11. 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

$$\left(I - \frac{Bx_j x_j^*}{x_j^* B x_j}\right)(A - \theta_j B)\left(I - \frac{x_j x_j^*}{x_j^* x_j}\right) w_j = (A - \theta_j B)x_j \tag{10.5}$$

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 $T_{\mathrm{C}} = (A - \theta_j B)^{-1}(A - (\theta_j + \epsilon_j)B)$ to the most recent Ritz vector. Example 1 in [227] 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 $T_{\mathrm{IC}}u = \eta u$, motivated at the end of §10.2.2.

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.

10.2.6 The 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 $M^{-1}(A - \nu B)$ 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 behaviour 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 the Lanczos recurrence relation for the inexact Cayley transform $T_{\mathrm{IC}} = M^{-1}(A - \nu B)$ is

$$T_{\mathrm{IC}} V_k - V_k H_k = f_k e_k^T \ .$$

The recurrence relation can also be written in the form

$$T_{\mathrm{C}} V_k - V_k H_k = f_k e_k^T + (T_{\mathrm{C}} - T_{\mathrm{IC}}) V_k$$

We know that for a Ritz pair (x, θ), $(T_{\mathrm{C}} - T_{\mathrm{IC}})x$ *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 [179] proved the convergence for the following algorithm, for computing eigenvalues of $Ax = \lambda x$, i.e. with $B = I$.

ALGORITHM 10.43: **Preconditioned Lanczos**

(1) *Given* v_1, $\|v_1\| = 1$ *and* $\nu_1 = v_1^* A v_1 / v_1^* B v_1$.
(2) *For* $l = 1, 2, \ldots$ *do*
(3) *Choose a precondioner* M_l *for* $A - \mu_l B$.
(4) *Run* k *steps of the Lanczos method with* M_l *orthogonalization applied to* $T_{\mathrm{IC}}^{(l)} = M_l^{-1}(A - \nu_l B)$.
(5) *Compute a Ritz vector* $x_l = V_k z_l$ *with* $H_k z_l = \zeta_l z_l$, *and* ζ *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* $\|Ax_l - \theta_l B x_l\| \leq$ TOL, *then stop*

Note that if $\mu = \nu$, then the preconditioner must not be a too good approximation to $A - \mu B$, otherwise $T_{\mathrm{IC}} \approx I$. The value of k may differ for each l. Morgan and Scott suggest the stopping test $-\zeta_l > \|T_{\mathrm{IC}}^{(l)} x_l - \zeta_l x_l\|$, which is cheap within the Lanczos method. It is shown in [179] that if both M_l and M_l^{-1} are uniformly bounded in norm. Then θ_l converges to an eigenvalue of A. Moreover, the asymptotic convergence is quadratic.

10.2.7 The 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 10.44: **Inexact rational Krylov**

(1) *Given v_1, and a zero θ_0, Let the residual be $r_0 = Av_1 - \theta_0 Bv_1$.*
(2) *For $j = 1, 2, \ldots$ 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 $v_1, \ldots, v_j$ into v_{j+1}. Keep the Gram-Schmidt coefficients in the jth column h_j of the Hessenberg matrix $H_{j+1,j}$*
(5) *Build the jth column of $L_{j+1,j}$ and $K_{j+1,j}$, i.e.* $\left(h_j - \begin{bmatrix} t_j \\ 0 \end{bmatrix}\right)$, *and* $\left(h_j\mu_j - \begin{bmatrix} t_j \\ 0 \end{bmatrix}\nu_j\right)$ *respectively.*
(6) *Solve the $j \times j$ eigenvalue problem*
$L^{\dagger}_{j+1,j}K_{j+1,j}z_j = \theta_j z_j$
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 analyse this algorithm step by step. The solution of the linear system leads to the relations (10.2) 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 $x_{j-1} = V_j L_{j,j-1}z$, 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 (10.2) 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 $j = 1, \ldots, k$ we have

$$AV_{k+1}L_{k+1,k} = BV_{k+1}K_{k+1,k} - S_k\ , \tag{10.6}$$

which is another way of writing (10.3). Note that when S_k is omitted from this equation, we obtain the usual RKS recurrence relation. Also in this case, $L_{k+1,k}$ are $K_{k+1,k}$ are upper Hessenberg matrices and $L_{k+1,k}$ has full rank. Let $L^{\dagger}_{k+1,k}$ denote the generalized Moore-Penrose inverse of $L_{k+1,k}$. With $E_k = S_k L^{\dagger}_{k+1,k}V^*_{k+1}$, we can also write

$$(A + E_k)V_{k+1}L_{k+1,k} = BV_{k+1}K_{k+1,k} \tag{10.7}$$

which corresponds to (10.4). 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 10.1 *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 $x = V_{k+1}L_{k+1,k}z$ 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 10.1, 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 10.44 produces residual

$$(A + E_k)x - \theta Bx = f_k$$

defined by Lemma 10.1. The residual related to the original problem can be written as

$$Ax - \theta Bx = f_k - E_k x$$

or following (10.6),

$$Ax - \theta Bx = f_k - S_k z \ .$$

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 Meerbergen and Lehoucq [?] and De Samblanx [46]. We just give a summary of the theory and some numerical results. In [?] and [46], mathematical and heuristical arguments are given that $S_j z_j \approx s_j$ such that, with $\|s_j\| \leq \tau \|r_{j-1}\|$, 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 $j = 1, \ldots, k$, 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 §10.2.5), 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 quadratric behaviour 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 3. This is now illustrated by the following numerical example. The Olmstead model [182] 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 discretisation with central differences with grid-size $h = 1/(n/2)$, the equations may be written as $\dot{x} = f(x)$ with $x = [u_1, v_1, u_2, v_2, \ldots, u_{n/2}, v_{n/2}]^*$. 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.

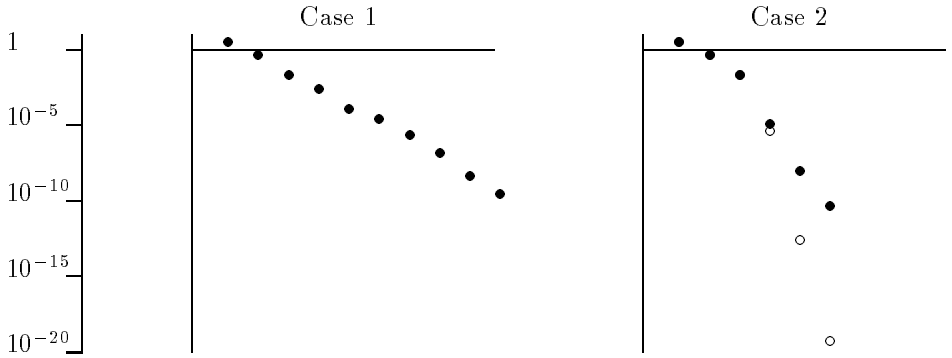
We solved linear systems by GMRES with 30 iteration vectors. The method was restarted by Morgan's implicitly restarted GMRES [172] keeping the 15 smallest Ritz pairs in the basis until the residual norm satisfied $\|s_j\| \leq \tau \|r_{j-1}\|$ with $\tau = 1.\,10^{-3}$. We used Algorithm 10.44 for computing 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 remaining steps solved the correction equation (10.5). From the results in Table 10.2 and Figure 10.1, we can see linear convergence for Case 1. Since the linear systems are solved more acurately ($\tau = 10^{-3}$) 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.

TABLE 10.2: *Residual norms of the inexact rational Krylov method for the Olmstead problem*

	Case 1		Case 2	
j	$\|r_j\|$	$\|f_j\|$	$\|r_j\|$	$\|f_j\|$
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	$1.2\,10^{-10}$	$1.3\,10^{-10}$		

FIGURE 10.1: *Logarithmic plots of residual norms of the inexact rational Krylov method for the Olmstead problem. The circles denote $\|f_j\|$ and the bullets $\|r_j\|$*



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 §10.2.3.

10.2.8 Inexact shift-and-invert

Recall from §10.2.1 that with exact arithmetic and exact linear system solvers, the shift-and-invert transformation 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 one.

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

$$(A - \mu B)z = Bx - t \qquad \text{where} \qquad \|t\| \leq \tau\|(A - \theta B)x\|/(\mu - \theta) \ .$$

In other words, we can use the shift-and-invert transformation, but instead of a relative residual tolerance, we should use a tolerance that is proportional to the norm of the Ritz residual, $\|(A - \theta B)x\|$.

10.3 Preconditioned Eigensolvers

A. Knyazev

10.3.1 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 $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$. If B is nonsingular, all eigenvalues are finite. If B is positive semi-definite (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

$$\mu = \frac{1}{\lambda}, \quad \mu_{\min} = \mu_n \leq \cdots \leq \mu_1 = \mu_{\max},$$

where we want to compute the largest m eigenvalues, $\mu_1, \ldots \mu_m$. 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:

$$(x_i, Ax_j) = (x_i, Bx_j) = 0, \ i \neq j.$$

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 shortness 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 shifted-and-inverted operator:

$$x^{(i+1)} = (B - \mu A)^{-1} A x^{(i)},$$

see §??. 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

$$\mu(x) = \frac{(x, Bx)}{(x, Ax)}, \tag{10.8}$$

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 method becomes a *preconditioned eigensolver*. We shall discuss this method later, in §10.3.7.

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.

10.3.2 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 also introduce a *preconditioner* T. Applying the preconditioner Tx to a vector x usually involves solving a linear system.

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

$$\delta_0(T^{-1}x, x) \leq (Ax, x) \leq \delta_1(T^{-1}x, x). \tag{10.9}$$

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 the preconditioner T 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 considerably computational costs.

We will *not assume* that the preconditioner T commutes with A, or $A^{-1}B$, despite of the fact that such an assumption would greatly simplify the theory of preconditioned methods.

We first define, follow [138], a preconditioned singe-vector iterative solver for the pencil $B - \mu A$, as a generalized polynomial method of the following kind:

$$x^{(k)} = P_{m_k}(TA, TB)x^{(0)}, \tag{10.10}$$

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 priory*, 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 $\mu(x)$ (10.8) is 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, With $B = I$ and $\lambda = 1/\mu$, we can obtain a similar algorithm for $A - \lambda I$.

The Preconditioned Eigensolver for $B - \mu A$

1. Start: select $x^{(0)}$.
2. Iterate k steps to compute $x^{(k)} = P_{m_k}(TA, TB)x^{(0)}$.
3. Compute $\mu^{(k)} = (x^{(k)}, Bx^{(k)})/(x^{(k)}, Ax^{(k)})$.

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 $x_j^{(k)}$, $j = 1, \ldots, m$ is computed simultaneously:

$$x_j^{(k)} = P_{m_k}(TA, TB)x_j^{(0)}, \ j = 1, \ldots, m, \tag{10.11}$$

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 (10.11) can be rewritten as the *Subspace Iterations*:

$$S^{(k)} = P_{m_k}(TA, TB)S^{(0)}, \tag{10.12}$$

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 [63, 64], but may converge slower in practice.

In (10.11), the iterative subspace $S^{(k)}$ is defined as a span

$$S^{(k)} = \text{Span}\ \{x_1^{(k)}, \ldots, x_m^{(k)}\}$$

of individual vectors $x_j^{(k)}$. It is common to use (10.11) recursively by combining it with a procedure of 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 $x_j^{(0)},\ j = 1, \ldots, m$.
2. Iterate k steps to compute $\widehat{x}_j^{(k)} = P_{m_k}(TA, TB)x_j^{(0)},\ j = 1, \ldots, m$.
3. Use the Rayleigh–Ritz method for the pencil $B - \mu A$ in the subspace Span $\{\widehat{x}_1^{(k)}, \ldots \widehat{x}_m^{(k)}\}$, 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.

10.3.3 The 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 10.45. 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 $\mu_{\min}$, or its estimate from below. If B is nonnegative definite, then $\mu_{\min}$ may simply be replaced with 0.

ALGORITHM 10.45: **The Preconditioned Power Method for** $B - \mu A$

(1) *Select a starting vector* $x^{(0)}$.
(2) *For* $i = 0, \ldots,$ *Until Convergence Do:*
(3) $\quad \mu^{(i)} := (x^{(i)}, B\,x^{(i)})/(x^{(i)}, A\,x^{(i)})$
(4) $\quad r := B\,x^{(i)} - \mu^{(i)} A x^{(i)}$
(5) $\quad w := Tr$
(6) $\quad x := w + \tau x$, *with the shift* $\tau = \delta_1(\mu^{(i)} - \mu_{\min})$
(7) $\quad x^{(i+1)} = x/\|x\|_A$

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 *et al.* [65, 62, 66] obtained explicit estimates of linear convergence for iterative Algorithm 10.45 using assumption (10.9). Somewhat simplified, see [135, 136, 138], the convergence rate estimate for Algorithm 10.45 can be written as

$$d\frac{\mu_1 - \mu^{(n)}}{\mu^{(n)} - \mu_2} \le (1 - \xi)^n \frac{\mu_1 - \mu^{(0)}}{\mu^{(0)} - \mu_2}, \quad \xi = \frac{\delta_0}{\delta_1}\frac{\mu_1 - \mu_2}{\mu_1 - \mu_{\min}}, \tag{10.13}$$

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 $\mu_p \ge \mu^{(0)} > \mu_{p+1}$ for some $p > 1$ holds [66].

In numerical experiments, Algorithm 10.45 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 to 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 10.45 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	$1-2$ 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 $\lambda_{\min}$ 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.

10.3.4 The Preconditioned Steepest Ascent/Descent Methods

These methods are similar to Algorithm 10.45 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 [219, 198]) than the power method with an *a priori* choice of the shift.

Here, we present the single-vector version of the preconditioned steepest ascent for the pencil $B - \mu A$ in Algorithm 10.46. For $B = I$, $\lambda = 1/\mu$, the method becomes a steepest descent method for $A - \lambda I$.

ALGORITHM 10.46: **The Preconditioned Steepest Ascent for** $B - \mu A$

(1) *select a starting vector* $x^{(0)}$.
(2) *For* $i = 0, \ldots,$ *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 the shift* τ *chosen to maximize the Rayleigh quotient* $\mu(x)$
(7) $x^{(i+1)} = x/\|x\|_A$

We note that the shift can be determined with the Rayleigh–Ritz method for the pencil $B - \mu A$ on the trial subspace 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 10.45. Therefore, the convergence results from the previous section can be applied without any changes, and similar observations can be made.

We have collected the dominant costs per iteration for Algorithm 10.46 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 compare 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 costs about twice than that of Algorithm 10.45. Though convergence is usually better than that of Algorithm 10.45 it is still worse than linear convergence of other preconditioned eigensolvers we consider below.

10.3.5 Preconditioned Lanczos Methods

The preconditioned Lanczos Method was suggested by Scott [221], though with $B = I$, and was analyzed by Knyazev [135, 136]. 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

$$T(B - \mu A)v = \nu v. \tag{10.14}$$

If $\mu = \mu_1$, then there exist a zero eigenvalue ν and the corresponding eigenvector of (10.14) 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 [135, 136].

In the method, we choose some initial value of parameter $\mu = \mu^{(0)}$, and then use a couple of Lanczos inner iterations to solve for the largest eigenvalue of (10.14). 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 10.47.

ALGORITHM 10.47: **The Preconditioned Lanczos Method for** $B-\mu A$

(1) *Select a starting vector* $x^{(0)}$.
(2) *For* $i = 0, \ldots,$ *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 [221]. 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 [135, 136]. 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 outer iterations of the Lanczos method, then the computational costs for each iteration is 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 $T^{-1}x$. 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:

$$\begin{aligned} \mu(x^{(i+1)}) &= \max_{x\in\mathcal{K}}\mu(x), \\ \mathcal{K} = \text{Span}\,\{x^{(i)}, T(B-\mu^{(i)}A)x^{(i)}, \ \ldots, [T(B-\mu^{(i)}A)]^{l_i}x^{(i)}\}, & \end{aligned} \tag{10.15}$$

where l_i is the number of inner iteration steps for the i-th outer iteration step.

This leads to a method that can be obtained from Algorithm 10.47 by modifying line (4) and (5):

ALGORITHM 10.48: **The Preconditioned Projection Method for** $B - \mu A$

(1) *Select a starting vector* $x^{(0)}$.
(2) *For* $i = 0, \ldots,$ *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) *On this Krylov subspace, apply the Rayleigh–Ritz method for the pencil* $B - \mu A$
(5) $x^{(i+1)} :=$ *Ritz vector, corresponding to the smallest Ritz value*

The method was suggested in [135, 136] and then was rediscovered in [179]. If only one inner iteration is used, the method in identical to the Preconditioned Steepest Descent/Ascent Algorithm 10.46.

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 [135, 136]. The Preconditioned Projection Algorithm 10.48 converges somewhat faster than the Preconditioned Lanczos Algorithm 10.47 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 increases significantly storage requirements.

10.3.6 The Davidson Method

In the Davidson method, an attempt is made to accelerate the convergence of the Preconditioned Projection Algorithm 10.48 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. [39, 40, 178, 173, 212]:

$$\begin{gathered}\mu(x^{(i+1)}) = \max_{x \in \mathcal{D}} \mu(x), \\ \mathcal{D}_i = \text{Span}\,\{x^{(0)}, T(B - \mu^{(0)}A)x^{(0)},\ T(B - \mu^{(1)}A)x^{(1)},\ \ldots, T(B - \mu^{(i)}A)x^{(i)}\}.\end{gathered} \tag{10.16}$$

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. A trivial procedure of a restart is to stop the method after 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 is quite intricate to study theoretically, cf. [31, 215].It is common to use the Davidson Method with an indefinite preconditioner (often the diagonal of $B - \mu^{(0)}A$), what further complicates the theoretical analysis.

The method was popular in particular in the Chemistry community, see [113, 144] where the involved matrices are typically diagonally dominant. For such problems, the method converges often quickly even with a simple diagonal preconditioner.

10.3.7 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 Rayleigh Quotient Iteration, applied to the pencil $B - \mu A$, see §4.3:

ALGORITHM 10.49: **The Rayleigh Quotient Iteration for $B - \mu A$ with Preconditioned Inner Iterative Solver**

(1) *select a starting vector* $x^{(0)}$.
(2) *For* $i = 0, \ldots,$ *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 the minimal residual method 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 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 Rayleigh Quotient Iterations, 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 Span $\{x^{(i)}\}$, using regularization, or/and choosing indefinite preconditioner T. It is not evident, however, whether an improvement of convergence of inner iterations is any 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., [258], or truncated rational Krylov methods, e.g., [207, ?],or inexact homotopy methods, e.g., [164, 116, 261, 165]; see also the previous section.

The Jacobi-Davidson method [227], described in §4.6 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 iterations 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.

10.3.8 The Preconditioned Conjugate Gradient Methods

In this subsection, we describe our favorite, in terms of practical efficiency, the *Locally Optimal Preconditioned Conjugate Gradient Method* [137]. It combines robustness and simplicity of the preconditioned steepest ascent Algorithm 10.46, three-term recurrence of the preconditioned Lanczos Algorithm 10.47, and, perhaps, fast convergence of the preconditioned projection Algorithm 10.48.

A simple variant of a preconditioned CG method for the pencil $B - \mu A$ can be written as

$$\begin{aligned} x^{(i+1)} &= w^{(i)} + \tau^{(i)} x^{(i)} + \gamma^{(i)} x^{(i-1)}, \\ w^{(i)} &= T(B x^{(i)} - \mu^{(i)} A x^{(i)}), \\ \mu^{(i)} &= \mu(x^{(i)}), \end{aligned} \tag{10.17}$$

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* [137], namely, select $\tau^{(i)}$ and $\gamma^{(i)}$ that maximize the Rayleigh quotient $\mu(x^{(i+1)})$ by using the Rayleigh–Ritz method, see Algorithm 10.50.

ALGORITHM 10.50: **The Preconditioned Conjugate Gradient Method for** $B - \mu A$

(1) *Select a starting vector* $x^{(0)}$.
(2) *For* $i = 0, \ldots,$ *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) *Use the Rayleigh–Ritz method for the pencil* $B - \mu A$
on the trial subspace Span $\{w, x^{(i)}, x^{(i-1)}\}$,
(7) $x^{(i+1)} :=$ *Ritz vector corresponding to the maximal Ritz value*

For the locally optimal version of the preconditioned conjugate gradient method, we can trivially apply convergence rate estimate (10.13) as the method converges not slower than the preconditioned steepest ascent in terms of the maximizing the Rayleigh quotient.

We have collected the dominant costs per iteration for Algorithm 10.50 in terms of storage and floating point operations, in the following tables, respectively:

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 has attracted attention recently; different versions have been suggested, see [243, 93, 259, 92, 78, 19].

We finally note that Algorithm 10.50 is unstable since the basis of the trial subspace is almost linear dependent. This can be cured by an orthogonalization prior to applying the Rayleigh–Ritz method.

We present a block version of Algorithm 10.50 in the next subsection.

10.3.9 Preconditioned Simultaneous Iterations

A well known idea of using *Simultaneous, or Block Iterations* provides an important improvement over singe-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 a 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.

The algorithm is a straightforward generalization of the singe-vector method and is typically combined with the Rayleigh–Ritz procedure. For every preconditioned eigensolvers discussed in previous subsections, a several variants of simultaneous iterations can be easily suggested; e.g., see block versions of the preconditioned power method with shift in [137, 64, 22]. For shortness, we describe here only one method - a simultaneous version of the locally optimal preconditioned conjugate gradient method - and only for the pencil $B - \mu A$.

ALGORITHM 10.51: **The Simultaneous Preconditioned Conjugate Gradient Method for** $B - \mu A$

(1) *Select a starting vector* $x_j^{(0)}$, $j = 1, \ldots, m$.
(2) *For* $i = 0, \ldots,$ *Until Convergence Do:*
(3) $\mu_j^{(i)} := (x_j^{(i)}, B\, x_j^{(i)})/(x_j^{(i)}, A\, x_j^{(i)})$, *for* $j = 1, \ldots, m$
(4) $r_j := B\, x_j^{(i)} - \mu_j^{(i)} A x_j^{(i)}$, *for* $j = 1, \ldots, m$
(5) $w_j := Tr_j$, *for* $j = 1, \ldots, m$
(6) *Use the Rayleigh–Ritz method for the pencil* $B - \mu A$
on the trial subspace
Span $\{w_1, \ldots, w_m, x_1^{(i)}, \ldots, x_m^{(i)}, x_1^{(i-1)}, \ldots, x_m^{(i-1)}\}$,
(7) $x_j^{(i+1)} :=$ *the* j*-th Ritz vector corresponding to the* j*-th from the top Ritz value for* $j = 1, \ldots, m$

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 10.51.

There is no theory available yet to predict accurately the speed of convergence of Algorithm 10.51. However, by analogy with known convergence theory of preconditioned CG system solvers, we expect convergence of $\mu_j^{(i)}$ to μ_j to be linear with the ratio

$$\left(\frac{1-\sqrt{\xi_j}}{1+\sqrt{\xi_j}}\right)^2, \quad \xi_j = \frac{\delta_0}{\delta_1}\frac{\mu_j - \mu_{m+1}}{\mu_j - \mu_{\min}}. \tag{10.18}$$

We compared numerically a block variant of the steepest ascent (SA) Algorithm (10.46) and Algorithm 10.51, 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 10.51. It is easy to separate the methods on a black-white print, too, as Algorithm 10.51 always converges much faster. Two straight lines correspond to linear convergence predicted by (10.18).

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

$$\lambda_i^{(i)} - \lambda_j, \ j = 1, 2.$$

The size of the problem is changing within 100-2000. 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 (10.9) on different Figures.

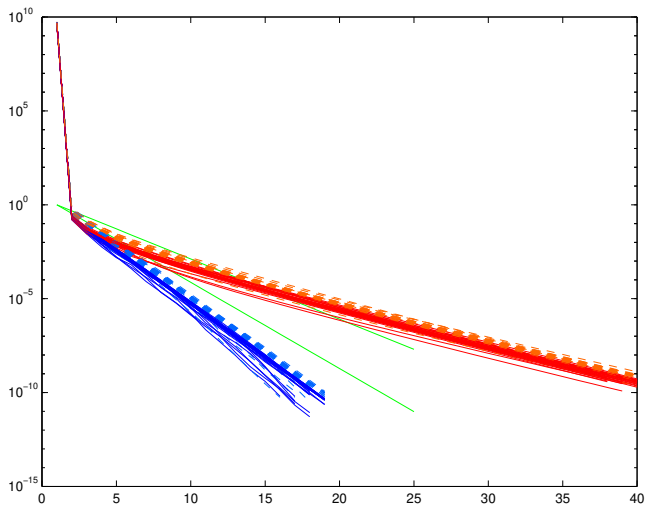


FIGURE 10.2: **CG vs. SA,** $\delta_1/\delta_0 = 10$

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 10.51 as the elements of a bundle, of convergence history lines, are quite close to each other on all figures. Moreover, our theoretical prediction (10.18) of the rate of convergence of Algorithm 10.51 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 10.51, exactly as we would expect. Finally, we observe that convergence for the first eigenvalue, in dark colors, is typically faster than that for the second one, in lighter colors and dashed.

There are known [35]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

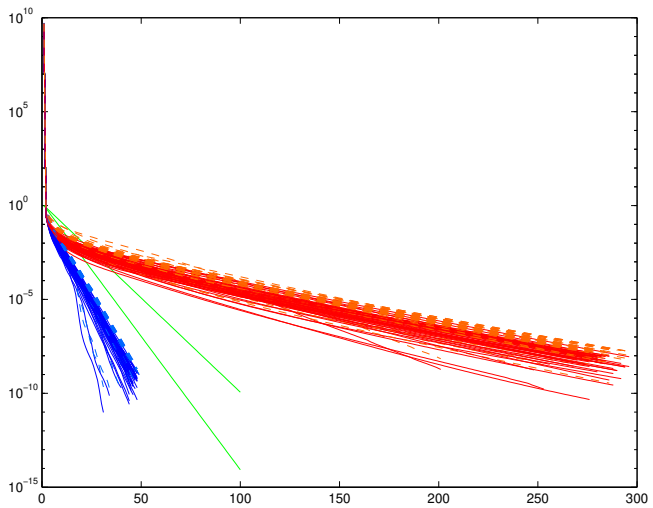


FIGURE 10.3: **CG vs. SA,** $\delta_1/\delta_0 = 100$

Rayleigh–Ritz method, e.g., [35, 244]and §7.3, here $B = I$. It leads to methods similar to Algorithm 10.51 if the CG method used for optimization.

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, e.g. [212, 142]. 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 singe-vector iterative methods when we try to 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 10.45, with additional orthogonalization, defined by an orthogonal projector $P^{\perp}$ onto an orthogonal complement of the subspace, spanned by already computed eigenvectors,

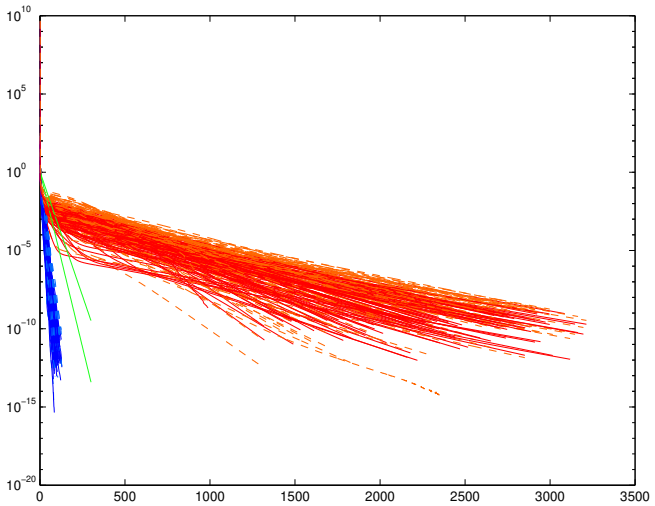


FIGURE 10.4: **CG vs. SA,** $\delta_1/\delta_0 = 1000$

which can be written as

$$x^{(i+1)} = P^{\perp} w^{(i)} + \tau^{(i)} x^{(i)}, \; w^{(i)} = T(Bx^{(i)} - \mu^{(i)} Ax^{(i)}). \tag{10.19}$$

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 (10.19) is loosing 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 [66, 64], 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, see [135, 136], instead of direct analysis of [66, 64], but the actual computation of $P^{\perp} w$ for a given w requires special attention.

Follow [135, 136], we take the original subspace, spanned by the frozen approximate eigenvectors, let us call it $\widetilde{X}$, and find a T^{-1}-orthogonal basis of the subspace $TA\widetilde{X}$. Mathematically, a T^{-1}-orthogonal complement to the latter subspace coincides with an A-orthogonal complement of the original subspace, $\widetilde{X}$. Now, we can use a standard

T^{-1}-orthogonal projector onto a T^{-1}-orthogonal complement of $TA\widetilde{X}$. Clearly, the T^{-1}-based scalar product must be possible to compute to use this trick.

We also note, that the use of T^{-1}-based scalar product, as well as any scalar product based on an ill-conditioned matrix may lead to unstable computations.

10.3.10 Software Availability

The Internet page

`http:// www-math.cudenver.edu/~aknyazev/ research/ eigensolvers/`

with the mirror at

`http:// www.cudenver.edu/~aknyazev/ research/ eigensolvers/`

is maintained by Andrew Knyazev. It contains an interactive discussion section, 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-77 code that computes eigenvalues of the Laplacian in the L-shaped domain on a uniform mesh, using preconditioned domain decomposition Lanczos-type method, see [139, 137, 140, 141].

This software is of experimental type and does not reflect all insights and possibilities described in this section.

CHAPTER 11

Of Things Not Treated

Appendix A. Collections of test matrices

In this appendix, we can list some collections of test matrices (such as Harwell-Boeing collection, Matrix Market, SPARSKIT etc), and list some practical applications of eigenvalue problems. If possible, include performance of algorithms on some of test matrices.

Glossary

Bibliography

[1] J. I. Aliaga, D. L. Boley, R. W. Freund, and V. Hernández. A Lanczos-type algorithm for multiple starting vectors, revised version. Technical Report Numerical Analysis Manuscript No. 98–3–05, Bell Laboratories, Murray Hill, New Jersey, USA, 1998.

[2] 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*. SIAM, Philadelphia, PA, USA, second edition, 1995.

[3] P. J. Anderson and G. Loizou. A Jacobi type method for complex symmetric matrices. *Numer. Math.*, 25:347–363, 1976.

[4] 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.

[5] M. Arioli, I. S. Duff, and D. Ruiz. Stopping criteria for iterative solvers. Report RAL-91-057, Central Computing Center, Rutherford Appleton Laboratory, 1992.

[6] V. I. Arnold. On matrices depending on parameters. *Russian Math. Survyes*, 26:29–43, 1971.

[7] W. E. Arnoldi. The principle of minimized iterations in the solution of the matrix eig envalue problem. *Quart. Appl. Math.*, 9:17–29, 1951.

[8] 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.

[9] J. Baglama, D. Calvetti, and L. Reichel. Fast Leja points. *ETNA*, 7:124–140, 1998.

[10] 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. *Journal of Computational Physics*, 146:203–226, 1998.

[11] Z. Bai. Error analysis of the Lanczos algorithm for the nonsymmetric eigenvalue problem. *Math. Comp.*, 62:209–226, 1994.

[12] Z. Bai. A spectral transformation block Lanczos algorithm for solving sparsenon-Hermitian eigenproblems. In J. G. Lewis, editor, *Proceedings of the Fifth SIAM Conference on Applied Linear Algebra*, pages 307–311, Philadelphia, PA, USA, June 1994. SIAM.

[13] Z. Bai, D. Day, and Q. Ye. ABLE: an adaptive block Lanczos method for non-Hermitian eigenval ue problems. Research Report 95-04, University of Kentucky, Department of Mathematics, Lexington, KY, USA, 1995. To appear in SIMAX.

[14] 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, 1998.

[15] 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, New Jersey, USA, 1998.

[16] 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. Softw.*, 23:494–513, 1998.

[17] Jesse Barlow and James Demmel. Computing accurate eigensystems of scaled diagonally dominant matrices. *SIAM J. Numer. Anal.*, 27(3):762–791, 1990.

[18] 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, PA, USA, 1994.

[19] Luca Bergamaschi, Giuseppe Gambolati, and Giorgio Pini. Asymptotic convergence of conjugate gradient methods for the partial symmetric eigenproblem. *Numer. Linear Algebra Appl.*, 4(2):69–84, 1997.

[20] 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 User's Guide.* SIAM, Philadelphia, 1997.

[21] D. Boley. The Algebraic Structure of Pencils and Block Toeplitz Matrices. *Lin. Alg. Appl.*, 279:255–279, April 1998.

[22] James H. Bramble, Joseph E. Pasciak, and Andrew V. Knyazev. A subspace preconditioning algorithm for eigenvector/eigenvalue computation. *Adv. Comput. Math.*, 6(2):159–189, 1996.

[23] C. Brezinski, M. Redivo Zaglia, and H. Sadok. Avoiding breakdown and near-breakdown in Lanczos type algorithms. *Numer. Alg.*, 1:261–284, 1991.

[24] F. Chaitin-Chatelin and V. Frayssé. *Lectures on Finite Precision Computations.* Society for Industrial and Applied Mathematics, 1996.

[25] 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.

[26] F. Chatelin. *Eigenvalues of matrices.* John Wiley and Sons, 1993.

[27] I. Chavel. *Riemannian Geometry—A Modern Introduction.* The Cambridge University Press, 1993.

[28] T.-Y. Chen. Balancing sparse matrices for computing eigenvalues. Master's thesis, University of California at Berkeley, May 1998.

[29] T.-Y. Chen and J. Demmel. Balancing sparse matrices for computing eigenvalues. submitted to Lin. Alg. Appl.

[30] B. D. Craven. Complex symmetric matrices. *J. Austral. Math. Soc.*, 10:341–354, 1969.

[31] M. Crouzeix, B. Philippe, and M. Sadkane. The Davidson method. *SIAM J. Sci. Comput.*, 15:62–76, 1994.

[32] J. Cullum and R. A. Willoughby. Computing eigenvalues of very large symmetric matrices—an implementation of a Lanczos algorithm with no reorthogonalization. *Journal of Computational Physics*, 44:329–358, 1981.

[33] J. Cullum and R. A. Willoughby. A practical procedure for computing eigenvalues of large sparse nonsymmetric matrices. In J. Cullum and R. A. Willoughby, editors, *Large Scale Eigenvalue Problems*, pages 193–240. Elsevier Science Publishers B. V., 1986.

[34] 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, 1974.

[35] J. K. Cullum and R. A. Willoughby. *Lanczos Algorithms for Large Symmetric Eigenvalue Computations, Volume 1, Theory.* Birkhäuser, Boston, 1985.

[36] J. K. Cullum and R. A. Willoughby. *Lanczos Algorithms for Large Symmetric Eigenvalue Computations, Volume 2, Programs.* Birkhäuser, Boston, 1985.

[37] 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.

[38] 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.

[39] 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.

[40] E. R. Davidson. Matrix eigenvector methods. In G. H. F. Direcksen and S. Wilson, editors, *Methods in Computational Molecular Physics*, pages 95–113. Reidel, Boston, 1983.

[41] G. J. Davis. Numerical solution of a quadratic matrix equation. *SIAM J. Sci. Comput.*, 2:164–175, 1981.

[42] D. Day. *Semi-duality in the two-sided Lanczos algorithm.* PhD thesis, UC Berkeley, 1993.

[43] D. Day. An efficient implementation of the nonsymmetric Lanczos algorithm. *SIAM J. Matrix Anal. Applic.*, 18:566–589, 1997.

[44] I. De Hoyos. Points of continuity of the Kronecker canonical form. *SIAM J. Matrix Anal. Appl.*, 11(2):278–300, April 1990.

[45] 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 Publishing Co., 1982.

[46] G. De Samblanx. *Filtering and restarting projection methods for eigenvalue problems.* PhD Thesis, Katholieke Universiteit Leuven, Department of Computer Science, 3001 Heverlee, Belgium, 1998.

[47] R. S. Dembo, Eisenstat S. C., and Steihaug T. Inexact newton methods. *SIAM J. Numer. Anal.*, 19:400–408, 1982.

[48] J. Demmel. *Applied Numerical Linear Algebra.* SIAM, Philadelphia, PA, 1997.

[49] J. Demmel and A. Edelman. The Dimension of Matrices (Matrix Pencils) with Given Jordan (Kronecker) Canonical Forms. *Lin. Alg. Appl.*, 230:61–87, 1995.

[50] J. Demmel and B. Kågström. Computing stable eigendecompositions of matrix pencils. *Lin. Alg. Appl.*, 88/89:139–186, April 1987.

[51] J. Demmel and B. Kågström. Accurate solutions of ill-posed problems in control theory. *SIAM J. Matrix. Anal. Appl.*, 9(1):126–145, January 1988.

[52] 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*, Vol.19(No. 2):160–174, June 1993.

[53] 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*, Vol.19(No. 2):175–201, June 1993.

[54] James Demmel and Krešimir Veselić. Jacobi's method is more accurate than QR. *SIAM J. Matrix Anal. Applic.*, 13(4):1204–1245, 1992.

[55] I. Dhillon. *A New $O(n^2)$ Algorithm for the Symmetric Tridiagonal Eigenvalue/Eigenvector Problem.* PhD thesis, University of California, Berkeley, 1997.

[56] J. J. Dongarra, J. DuCroz, I. S. Duff, and S. Hammarling. A set of Level 3 Basic Linear Algebra Subprograms. *ACM Transactions on Mathematical Software*, 16(1):1–17, 1990.

[57] J. J. Dongarra, J. DuCroz, S. Hammarling, and R. J. Hanson. An extended set of Fortran Basic Linear Algebra Subprograms. *ACM Trans. on Math. Software*, 14(1):1–17, 1988.

[58] 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.

[59] 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, England, 1995.

[60] I. S. Duff and J. A. Scott. Computing selected eigenvalues of large sparse unsymmetric matrices using subspace iteration. *ACM Trans. Math. Softw.*, 19:137–159, 1993.

[61] R. J. Duffin. A minimax theory for overdamped networks. *J. of Rational Mechanics and Analysis*, 4:221–233, 1955.

[62] E. G. D'yakonov. Iteration methods in eigenvalue problems. *Math. Notes*, 34:945–953, 1983.

[63] 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.

[64] E. G. D'yakonov and A. V. Knyazev. On an iterative method for finding lower eigenvalues. *Russian J. of Numerical Analysis and Math. Modelling*, 7(6):473–486, 1992.

[65] E. G. D'yakonov and M. Yu. Orekhov. Minimization of the computational labor in determining the first eigenvalues of differential operators. *Mathematical Notes*, 27(5–6):382–391, 1980.

[66] Eugene G. D'yakonov. *Optimization in solving elliptic problems.* CRC Press, Boca Raton, FL, 1996. Translated from the 1989 Russian original, Translation edited and with a preface by Steve McCormick.

[67] 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.

[68] 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.

[69] A. Edelman and Y. Ma. Staircase Failures Explained by Orthogonal Versal Forms. *SIAM J. Matrix Anal. Appl.*, 1999. *Submitted.*

[70] A. Edelman and S. T. Smith. On conjugate gradient-like methods for eigen-like problems. *BIT*, to appear. See also *Proc. Linear and Nonlinear Conjugate Gradient-Related Methods*, eds. Loyce Adams and J. L. Nazareth. SIAM, Philadelphia, PA, 1996.

[71] E. Edelman, T. A. Arias, and S. T. Smith. The geometry of algorithms with orthogonality constraints. *SIAM J. Matrix Anal. Applic.*, 20:303–353, 1999.

[72] L. Eldén. Algorithms for the regularization of ill-conditioned least-squares problems. *BIT*, 17:134–145, 1977.

[73] 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, S-901 87 Umeå, Sweden, 1999.

[74] 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.

[75] 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.

[76] 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.

[77] 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. Association for Computing Machinery, Inc., 1995.

[78] 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.

[79] R. Fletcher. *Conjugate gradient methods for indefinite systems*, volume 506 of *Lecture Notes Math.*, pages 73–89. Springer-Verlag, Berlin–Heidelberg–New York, 1976.

[80] R. Fletcher. *Practical Methods of Optimization*. John Wiley & Sons, New York, 2d edition, 1987.

[81] D. R. Fokkema. *Subspace methods for linear, nonlinear, and eigen problems*. PhD thesis, Utrecht University, Utrecht, 1996.

[82] 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.

[83] R. W. Freund. Conjugate gradient-type methods for linear systems with complex symmetric coefficient matrices. *SIAM J. Sci. Statist. Comput.*, 13:425–448, 1992.

[84] 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, 1997.

[85] R. W. Freund. Reduced-order modeling techniques based on Krylov subspaces and their use in circuit simulation. Technical Report Numerical Analysis Manuscript No. 98–3–02, Bell Laboratories, Murray Hill, New Jersey, USA, 1998. To appear in *Applied and Computational Control, Signals, and Circuits*.

[86] 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.

[87] R. W. Freund and N. M. Nachtigal. QMR : a quasi-minimal residual method for non-Hermitian linear systems. *Numer. Math.*, 60:315–339, 1991.

[88] R. W. Freund and N. M. Nachtigal. QMRPACK: a package of QMR algorithms. *ACM Trans. Math. Software*, 22:46–77, 1996.

[89] Roland Freund. A transpose-free quasi-minimal residual algorithm for non-Hermitian linear systems. *SIAM J. Sci. Comput.*, 14:470–482, 1993.

[90] 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, 1998.

[91] Z. Fu and E. M. Dowling. Conjugate gradient eigenstructure tracking for adaptive spectral estimation. *IEEE Trans. Signal Processing*, 43(5):1151–1160, 1995.

[92] Giuseppe Gambolati, Giorgio Pini, and Mario Putti. Nested iterations for symmetric eigenproblems. *SIAM J. Sci. Comput.*, 16(1):173–191, 1995.

[93] Giuseppe Gambolati, Flavio Sartoretto, and Paolo Florian. An orthogonal accelerated deflation technique for large symmetric eigenproblems. *Comput. Methods Appl. Mech. Engrg.*, 94(1):13–23, 1992.

[94] F. Gantmacher. *The Theory of Matrices, vol. I and II (transl.)*. Chelsea, New York, 1959.

[95] F. Gantmacher. *The Theory of Matrices, Vol. I and II (transl.)*. Chelsea, New York, 1959.

[96] I. Garcia-Planas. Kronecker Stratification of the Space of Quadruples of Matrices. *SIAM J. Matrix Anal. Appl.*, 19(4):872–885, 1998.

[97] G. Geist, G. Howell, and D. Watkins. The BR eigenvalue algorithm. *SIAM J. Matrix Anal. Applic.*, 00:to appear, 1999.

[98] M. Genseberger and G. L.G. Sleijpen. Alternative corection equations in the Jacobi-Davidson method. Preprint 1073, Department of Mathematics, Utrecht University, Utrecht, the Netherlands, 1998.

[99] P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization*. Academic Press, New York, 2d edition, 1981.

[100] G. Golub and R. Underwood. The block Lanczos method for computing eigenvalues. In J. Rice, editor, *Mathematical Software III*. Academic Press, 1977.

[101] G. Golub and C. Van Loan. *Matrix computations*. The Johns Hopkins University Press, 3rd edition, 1996.

[102] G. Golub and J. H. Wilkinson. Ill-conditioned eigensystems and the computation of the Jordan canonical form. *SIAM Review*, 18(4):578–619, 1976.

[103] Joseph F. Grcar. *Analyses of the Lanczos algorithm and the approximation problem in Richardson's method.* PhD thesis, University of Illinois at Urbana-Champaign, 1981.

[104] 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. Applic.*, 15:228–272, 1994.

[105] E. Grimme, D. Sorensen, and P. Van Dooren. Model reduction of state space systems via an implicitly restarted lanczos method. *Numerical Algorithms*, 12:1–32, 1996.

[106] I. Gustafsson. A class of first order factorization methods. *BIT*, 18:142–156, 1978.

[107] M. H. Gutknecht. A completed theory of the unsymmetric Lanczos process and related algorithms, Part I. *SIAM J. Matrix Anal. Applic.*, 13:594–639, 1992.

[108] M. H. Gutknecht. A completed theory of the unsymmetric Lanczos process and related algorithms, Part II. *SIAM J. Matrix Anal. Applic.*, 15:15–58, 1994.

[109] S. Helgason. *Differential Geometry, Lie Groups, and Symmetric Spaces.* Academic Press, New York, 1978.

[110] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *J. Res. Natl. Bur. Stand.*, 49:409–436, 1954.

[111] Vincent Heuveline and Miloud Sadkane. Arnoldi-faber method for large non-Hermitian eigenvalue problems. *ETNA*, 7:62–76, 1997.

[112] D. Hinrichsen and J. O'Halloran. Orbit Closures of Singular Pencils. *J. Pure and Applied Algebra*, 81:117–137, 1992.

[113] K. Hirao and H. Nakatsuji. A generalization of the Davidson's method to large nonsymmetric eigenvalue problem. *J. Comput. Phys*, 45(2):246–254, 1982.

[114] R. A. Horn and C. R. Johnson. *Matrix Analysis.* Cambridge University Press, Cambridge, UK, 1985.

[115] A. S. Householder. *The Theory of Matrices in Numerical Analysis.* Blaisdell, New York, NY, 1964. Dover edition 1975.

[116] Liang Jiao Huang and Tien-Yien Li. Parallel homotopy algorithm for symmetric large sparse eigenproblems. *J. Comput. Appl. Math.*, 60(1-2):77–100, 1995. Linear/nonlinear iterative methods and verification of solution (Matsuyama, 1993).

[117] The Math Works Inc. Partial differential toolbox for use with Matlab. ??, 1995.

[118] I. C. F. Ipsen. Relative perturbation results for matrix eigenvalues and singular values. In A. Iserles, editor, *Acta Numerica, volume 7.* Cambridge University Press, 1998.

[119] C. G. J. Jacobi. Ueber ein leichtes Verfahren, die in der Theorie der Säcularstörungen vorkommenden Gleichungen numerisch aufzulösen. *Journal für die Reine und Angewandte Mathematik*, 30:51–94, 1846.

[120] Z. Jia. A block incomplete orthogonalisation method for large nonsymmetric eigenproblems. *BIT*, 35:516–539, 1995.

[121] Z. Jia. Polynomial characterizations of the approximate eigenvectors by the refined Arnoldi method and an implicitly restarted refined Arnoldi algorithm. *Linear Alg. Appl.*, 287:191–214, 1998.

[122] Z. Jia. A refined iterative algorithm based on the block Arnoldi process for large unsymmetric eigenproblems. *Linear Alg. Appl.*, 270:171–189, 1998.

[123] P. Johansson. StratiGraph Users' Guide. Version 1.1. Technical Report UMINF-99.11, Department of Computing Science, Umeå University, S-901 87 Umeå, Sweden, 1999.

[124] M. T. Jones and M. L. Patrick. The Lanczos algorithm for the generalized symmetric eigenproblem on shared-memory architectures. *Applied Numerical Mathematics*, 12:377–389, 1993.

[125] 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, 1981.

[126] B. Kågström. RGSVD - an algorithm for computing the Kronecker canonical form and reducing subspaces of singular $A - \lambda B$ pencils. *SIAM J. Sci. Stat. Comp.*, 7(1):185–211, 1986.

[127] 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.

[128] 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.

[129] 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, S-901 87 Umeå, Sweden, 1998. Submitted to *Numerical Algorithms*.

[130] W. Kahan, B. N. Parlett, and E. Jiang. Residual bounds on approximate eigensystems of nonnormal matrices. *SIAM J. Numer. Anal.*, 19:470–484, 1982.

[131] W. Karush. An iterative method for finding characteristics vectors of a symmetric matrix. *Pacific J. Mathematics*, 1:233–248, 1951.

[132] 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.

[133] 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.

[134] S. Kim and A. Chronopoulos. An efficient nonsymmetric Lanczos method on parallel vector computers. *J. Comp. and Appl. Math.*, 42:357–374, 1992.

[135] A. V. Knyazev. *Computation of eigenvalues and eigenvectors for mesh problems: algorithms and error estimates.* Dept. Numerical Math. USSR Academy of Sciences, Moscow, 1986. (In Russian).

[136] A. V. Knyazev. Convergence rate estimates for iterative methods for mesh symmetric eigenvalue problem. *Soviet J. Numerical Analysis and Math. Modelling*, 2(5):371–396, 1987.

[137] 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, Basel, 1991. Birkhauser.

[138] A. V. Knyazev. Preconditioned eigensolvers - an oxymoron? *ETNA*, 7:104–123, 1998.

[139] 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. Numerical Analysis and Math. Modelling*, 4(4):283–310, 1989.

[140] A. V. Knyazev and A. L. Skorokhodov. The preconditioned gradient-type iterative methods in a subspace for partial generalized symmetric eigenvalue problem. *Soviet Math. Doklady*, 45(2):474–478, 1993.

[141] A. V. Knyazev and A. L. Skorokhodov. The preconditioned gradient-type iterative methods in a subspace for partial generalized symmetric eigenvalue problem. *SIAM J. Numerical Analysis*, 31(4):1226–1239, 1994.

[142] Andrew V. Knyazev. New estimates for Ritz vectors. *Math. Comp.*, 66(219):985–995, 1997.

[143] S. Kobayashi and K. Nomizu. *Foundations of Differential Geometry.* Wiley, New York, 1969.

[144] N. Kosugi. Modification of the Liu-Davidson method for obtaning one or simultaneously several eigensolutions of a large real symmetric matrix. *J. Comput. Phys.*, 55(3):426–436, 1984.

[145] L. Kronecker. Algebraische Reduction der Schaaren Bilinearer Formen. S. B. Akad., 1890. pp. 763–776.

[146] 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.

[147] V. N. Kublanovskaya. An approach to solving the spectral problem of $A - \lambda B$. In B. Kågström and A. Ruhe, editors, *Matrix Pencils*, pages 17–29. Springer-Verlag, Berlin, 1983. Lecture Notes in Mathematics, vol. 973, Proceedings, Pite Havsbad, 1982.

[148] V. N. Kublanovskaya. AB-algorithm and its modifications for the spectral problem of linear pencils of matrices. *Num. Math.*, 43:329–342, 1984.

[149] 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).

[150] Yu-Ling Lai, Kun-Yi Lin, and Wen-Wei Lin. An inexact inverse iteration for large sparse eigenvalue problems. *Num. Lin. Alg. Appl.*, 4:425–437, 1997.

[151] P. Lancaster. *Lambda-matrices and vibrating systems.* Pergamon Press, Oxford, England, 1966.

[152] C. Lanczos. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *J. Res. Nat. Bur. Stand.*, 45:255–282, 1950.

[153] C. Lanczos. Solution of systems of linear equations by minimized iterations. *J. Res. Natl. Bur. Stand*, 49:33–53, 1952.

[154] R. B. Lehoucq. *Analysis and implementation of an implicitly restarted Arnoldi iteration.* Phd thesis, Rice University, Houston, TX, USA, 1995.

[155] 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.

[156] R. B. Lehoucq and K. Meerbergen. Using generalized Cayley transformations within an inexact rational Krylov sequence method. *SIAM J. Matrix Anal. Applic.*, 20(1):131–148, 1998.

[157] 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, USA, 1995.

[158] 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, New Mexico, 1997.

[159] R. B. Lehoucq and D. C. Sorensen. Deflation techniques within an implicitly restarted Arnoldi iteration. *SIAM J. Matrix Anal. Applic.*, 17:789–821, 1996.

[160] R. B. Lehoucq, D. C. Sorensen, and C. Yang. *ARPACK USERS GUIDE: Solution of Large Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods.* SIAM, Phildelphia, PA, 1998.

[161] Ren-Cang Li. Relative perturbation theory. III. More bounds on eigenvalue variation. *Linear Algebra Appl.*, 266:337–345, 1997.

[162] Ren-Cang Li. Relative perturbation theory. I. Eigenvalue and singular value variations. *SIAM J. Matrix Anal. Appl.*, 19:956–982, 1998.

[163] Ren-Cang Li. Relative perturbation theory. II. Eigenspace and singular subspace variations. *SIAM J. Matrix Anal. Appl.*, 20:471–492, 1999.

[164] 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.

[165] 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.

[166] T. Manteuffel. The Tchebyshev iteration for nonsymmetric linear systems. *Numer. Math.*, 28:307–327, 1977.

[167] K. Meerbergen and D. Roose. Matrix transformations for computing rightmost eigenvalues of real nonsymmetric matrices. *IMA J. Numer. Anal.*, 16:297–346, 1996.

[168] 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. Applic.*, 18:1–20, 1997.

[169] 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.

[170] Karl Meerbergen. A theoretical comparison between inner products in the shift-invert Arnoldi method and the spectral transformation Lanczos method. *ETNA*, 7:90–103, 1998.

[171] 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.

[172] R. B. Morgan. Implicitly restarted GMRES and Arnoldi methods for nonsymmetric systems of equations. *SIAM J. Matrix Anal. Applic.* To appear.

[173] R. B. Morgan. Davidson's method and preconditioning for generalized eigenvalue problems. *J. Comp. Phys.*, 89:241–245, 1990.

[174] R. B. Morgan. Theory for preconditioning eigenvalue problems. *Proceedings of the Copper Mountain Conference on Iterative Methods*, 1990.

[175] R. B. Morgan. Computing interior eigenvalues of large matrices. *Linear Alg. Appl.*, 154–156:289–309, 1991.

[176] R. B. Morgan. Generalisations of Davidson's method for computing eigenvalues of large nonsymmetric matrices. *J. Comput. Phys.*, 101:287–291, 1992.

[177] R. B. Morgan. On restarting the Arnoldi method for large nonsymmetric eigenvalue problems. *Math. Comp.*, 65:1213–1230, 1996.

[178] 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.

[179] R. B. Morgan and D. S. Scott. Preconditioning the Lanczos algorithm for sparse symmetric eigenvalue problems. *SIAM J. Sci. Comput.*, 14:585–593, 1993.

[180] R. B. Morgan and M. Zeng. Harmonic projection methods for large non-symmetric eigenvalue problems. *Num. Lin. Alg. Appl.*, 5:33–55, 1998.

[181] S. G. Nash and A. Sofer. *Linear and Nonlinear Programming.* McGraw-Hill, New York, 1995.

[182] W. E. Olmstead, W. E. Davis, S. H. Rosenblat, and W. L. Kath. Bifurcation with memory. *SIAM J. Appl. Math.*, 40:171–188, 1986.

[183] J. Olsen, P. Jørgensen, and J. Simons. Passing the one-billion limit in full configuration-interaction (FCI) calculations. *Chemical Physics Letters*, 169:463–472, 1990.

[184] E. E. Osborne. On pre-conditioning of matrices. *J. Assoc. Comput. Mach.*, 7:338–345, 1960.

[185] C. Paige. *The computation of eigenvalues and eigenvectors of very large sparse matrices.* PhD thesis, London University, London, England, 1971.

[186] C. Paige. Properties of numerical algorithms related to computing controllability. *IEEE Trans. Autom. Contr.*, AC-26(1):130–138, 1981.

[187] C. C. Paige, B. N. Parlett, and H. A. van der Vorst. Approximate solutions and eigenvalue bounds from Krylov subspaces. *Num. Lin. Alg. Appl.*, 2:115–133, 1995.

[188] C. C. Paige and M. A. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM J. Numer. Anal.*, 12:617–629, 1975.

[189] C. C. Paige and M. A. Saunders. LSQR: An algorithm for sparse linear equations and sparse least squares. *ACM Trans. Math. Softw.*, 8:43–71, 1982.

[190] B. Parlett. Reduction to tridiagonal form and minimal realizations. *SIAM J. Mat. Anal. Appl.*, 13(2):567–593, 1992.

[191] B. Parlett and C. Reinsch. Balancing a matrix for calculation of eigenvalues and eigenvectors. *Numer. Math.*, 13:293–304, 1969.

[192] B. N. Parlett. *The symmetric eigenvalue problem.* Prentice Hall series in computational mathematics. Prentice-Hall, 1980. Reprinted by SIAM Press, 1997.

[193] B. N. Parlett. Invariant subspaces for tightly clustered eigenvalues of tridiagonals. *BIT*, 36:542–562, 1996.

[194] B. N. Parlett and I. S. Dhillon. Fernando's solution to Wilkinson's problem: an application of double factorization. *Linear Alg. Appl.*, 267:247–279, 1997.

[195] B. N. Parlett and J. Le. Forward instability of tridiagonal QR. *SIAM J. Matrix Anal. Applic.*, 14:279–316, 1993. 316.

[196] B. N. Parlett and D. Scott. The Lanczos algorithm with selective orthogonalization. *Mathematics of Computation*, 33:217–238, 1979.

[197] B. N. Parlett, D. R. Taylor, and Z. S. Liu. A look-ahead Lanczos algorithm for nonsymmetric matrices. *Math. Comp.*, 44:105–124, 1985.

[198] W. V. Petryshyn. On the eigenvalue problem $Tu - \lambda Su = 0$ with unbounded and non-symmetric operators T and S. *Philos. Trans. R. Soc. Math. Phys. Sci.*, 262:413–458, 1968.

[199] A. Pokrzywa. On perturbations and the equivalence orbit of a matrix pencil. *Lin. Alg. Appl.*, 82:99–121, 1986.

[200] E. Polak. *Computational Methods in Optimization*. Academic Press, New York, 1971.

[201] A. Ruhe. An algorithm for numerical determination of the structure of a general matrix. *BIT*, 10:196–216, 1970.

[202] A. Ruhe. Algorithms for the nonlinear eigenvalue problem. *SIAM J. Numer. Anal.*, 10:674–689, 1973.

[203] A. Ruhe. SOR-methods for the eigenvalue problem with large sparse matrices. *Math. Comp.*, 28:695–710, 1974.

[204] A. Ruhe. Iterative eigenvalue algorithms based on convergent splittings. *J. Comput. Phys.*, 19:110–120, 1975.

[205] A. Ruhe. Implementation aspects of band Lanczos algorithms for computation of eigenvalues of large sparse symmetric matrices. *Math. Comp.*, 33:680–687, 1979.

[206] A. Ruhe and T. Wiberg. The method of conjugate gradients used in inverse iteration. *BIT*, 12:543–554, 1972.

[207] Axel Ruhe. Rational Krylov: a practical algorithm for large sparse nonsymmetric matrix pencils. *SIAM J. Sci. Comput.*, 19(5):1535–1551 (electronic), 1998.

[208] Y. Saad. Least squares polynomials in the complex plane with applications to solving sparse nonsymmetric matrix problems. Research Report YALEU/DCS/RR-276, Yale University, New Haven, CT, USA, 1983.

[209] Y. Saad. Chebyshev acceleration techniques for solving nonsymmetric eigenvalue problems. *Math. Comp.*, 42(166):567–588, 1984.

[210] 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.

[211] Y. Saad. Numerical solution of large nonsymmetric eigenvalue problems. *Comput. Phys. Comm.*, 53:71–90, 1989.

[212] Y. Saad. *Numerical methods for large eigenvalue problems.* Halsted Press, Div. of John Wiley & Sons, Inc., New York, 1992.

[213] Y. Saad. *Iterative Methods for Linear Systems.* PWS Publishing, Boston, 1996.

[214] 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.

[215] M. Sadkane. Block-Arnoldi and Davidson methods for unsymmetric large eigenvalue problems. *Numer. Math.*, 64:195–211, 1993.

[216] M. Sadkane. A block Arnoldi-Chebyshev method for computing the leading eigenpairs of large sparse unsymmetric matrices. *Numer. Math.*, 64:181–193, 1993.

[217] G. De Samblanx and A. Bultheel. Nested lanczos: Implicitly restarting a lanczos algorithm. *Numerical Algorithms*, 18:31–50, 1998.

[218] A. H. Sameh and J. A. Wisniewski. A trace minimization algorithm for the generalized eigenvalue problem. *SIAM J. Numer. Anal.*, 19:1243–1259, 1982.

[219] B. A. Samokish. The steepest descent method for an eigenvalue problem with semi-bounded operators. *Izvestiya Vuzov, Math.*, ?(5):105–114, 1958. (In Russian).

[220] W. H. A. Schilders. Personal communication, Semptember, 1997.

[221] D. S. Scott. Solving sparse symmetric generalised eigenvalue problems without factorisation. *SIAM J. Numer. Anal.*, 18:102–110, 1981.

[222] J. A. Scott. An Arnoldi code for computing selected eigenvalues of sparse unsymmet ric matrices. *ACM Trans. Math. Softw.*, 21:432–475, 1995.

[223] H. Simon. Analysis of the symmetric Lanczos algorithm with reorthogonalization methods. *Lin. Alg. Appl.*, 61:101–132, 1984.

[224] H. Simon. The Lanczos algorithm with partial reorthogonalization. *Math. Comp.*, 42:115–142, 1984.

[225] G. L. G. Sleijpen and D. R. Fokkema. Bi-CGSTABℓ methods for linear equations involving matrices with complex spectrum. *ETNA*, 1:11–32, 1993.

[226] G. L. G. Sleijpen, D. R. Fokkema, and H. A van der Vorst. BiCGSTAB(ℓ) and other hybrid Bi-CG methods. *Numer. Algor.*, 7:75–109, 1994.

[227] G. L. G. Sleijpen and H. A. van der Vorst. A Jacobi-Davidson iteration method for linear eigenvalue problems. *SIAM J. Matrix Anal. Applic.*, 17:401–425, 1996.

[228] 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. *ETNA*, 7:75–89, 1998.

[229] 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.

[230] P Smit and MHC Paardekooper. The effects of inexact linear solvers in algorithms for symmetric eigenvalue problems. *Linear Alg. Appl.*, 287:337–357, 1998.

[231] B. T. Smith, J. M. Boyle, J. J. Dongarra B. S. Garbow, Y. Ikebe, V. C. Klema, and C. B. Moler. *EISPACK Guide*. Springer–Verlag, Berlin, second edition, 1976. Volume 6 of *Lecture Notes in Computer Science*.

[232] S. T. Smith. Optimization techniques on Riemannian manifolds. *Fields Institute Communications*, 3:113–146, 1994.

[233] P. Sonneveld. CGS: a fast Lanczos-type solver for nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 10:36–52, 1989.

[234] D. C. Sorensen. Implicit application of polynomial filters in a k-step Arnoldi method. *SIAM J. Matrix Anal. Applic.*, 13:357–385, 1992.

[235] D. C. Sorensen. Deflation for implicitly restarted Arnoldi methods. Technical report TR98-12, Department of Computational and Applied Mathematics, Rice University, 1998.

[236] D.C. Sorensen. Implicit application of polynomial filters in a K–step Arnoldi method. *SIAM J. Matrix Anal. Appl.*, 13:357–385, 1992.

[237] 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.

[238] G. W. Stewart. Simultaneous iteration for computing invariant subspaces of non-Hermitian matrices. *Numer. Math.*, 25:123–136, 1976.

[239] G. W. Stewart and J.-G. Sun. *Matrix Perturbation Theory*. Academic Press, New York, 1990.

[240] W. J. Stewart and A. Jennings. Algorithm 570: LOPSI a simultaneous iteration method for real matrices. *ACM Trans. Math. Softw.*, 7:230–232, 1981.

[241] W. J. Stewart and A. Jennings. A simultaneous iteration algorithm for real matrices. *ACM Trans. Math. Softw.*, 7:184–198, 1981.

[242] 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.

[243] 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.

[244] A. Edelman T. Arias 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, Philadelphia, 1994. SIAM.

[245] D. R. Taylor. *Analysis of the Look-Ahead Lanczos algorithm*. PhD thesis, UC at Berkeley, 1982.

[246] H. van der Veen and C. Vuik. Bi-lanczos with partial orthogonalization. *Computers and Structures*, 56:605–613, 1995.

[247] 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.

[248] P. Van Dooren. The computation of Kronecker's canonical form of a singular pencil. *Lin. Alg. Appl.*, 27:103–141, 1979.

[249] P. Van Dooren. The generalized eigenstructure problem in linear system theory. *IEEE Trans. Autom. Contr.*, AC-26(1):111–129, 1981.

[250] P. Van Dooren. A generalized eigenvalue approach for solving Ricatti equations. *SIAM J. Sci. Comput.*, 2:121–135, 1981.

[251] P. Van Dooren. Algorithm 590, DUSBSP and EXCHQZ: FORTRAN subroutines for computing deflating subspaces with specified spectrum. *ACM Trans. Math. Softw.*, 8:376–382, 1982.

[252] P. Van Dooren. Reducing subspaces: Definitions, properties and algorithms. In B. Kågström and A. Ruhe, editors, *Matrix Pencils*, pages 58–73. Springer-Verlag, Berlin, 1983. Lecture Notes in Mathematics, vol. 973, Proceedings, Pite Havsbad, 1982.

[253] H. F. Walker. Implementation of the GMRES method using Householder transformations. *SIAM J. Sci. Statist. Comput.*, 9:152–163, 1988.

[254] W. Waterhouse. The codimension of singular matrix pairs. *Lin. Alg. Appl.*, 57:227–245, 1984.

[255] J. H. Wilkinson. *The algebraic eigenvalue problem.* Clarendon Press, Oxford, UK, 1965.

[256] J. H. Wilkinson and C. Reinsch. *Handbook for Automatic Computation. Vol. II Linear Algebra.* Springer, New York, 1971.

[257] Y.-C. Wong. Differential geometry of Grassmann manifolds. *Proc. Natl. Acad. Sci. USA*, 57:589–594, 1967.

[258] Kesheng Wu, Yousef Saad, and Andreas Stathopoulos. Inexact Newton preconditioning techniques for eigenvalue problems. Technical report, 1998. Lawrence Berkeley National Laboratory report number 41382. Minnesota Super number UMSI 98-10.

[259] H. Yang. Conjugate gradient methods for the Rayleigh quotient minimization of generalized eigenvalue problems. *Computing*, 51(1):79–94, 1993.

[260] Q. Ye. A breakdown-free variation of the nonsymmetric Lanczos algorithms. *Math. Comp.*, 62:179–207, 1994.

[261] 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 (electronic), 1998.

Index