L A P A C K

L -A P -A C -K

L A P A -C -K

L -A P -A -C K

L A -P -A C K

L -A -P A C -K

Users' Guide

Third Edition

E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra,
J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen

siam

SOFTWARE • ENVIRONMENTS • TOOLS

LAPACK Users' Guide

SOFTWARE · ENVIRONMENTS · TOOLS

The series includes handbooks and software guides as well as monographs on practical implementation of computational methods, environments, and tools. The focus is on making recent developments available in a practical format to researchers and other users of these methods and tools.

Editor-in-Chief

Jack J. Dongarra
University of Tennessee and Oak Ridge National Laboratory

Editorial Board

James W. Demmel, *University of California, Berkeley*
Dennis Gannon, *Indiana University*
Eric Grosse, *AT&T Bell Laboratories*
Ken Kennedy, *Rice University*
Jorge J. Moré, *Argonne National Laboratory*

Software, Environments, and Tools

E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide, Third Edition*
Michael W. Berry and Murray Browne, *Understanding Search Engines: Mathematical Modeling and Text Retrieval*
Jack J. Dongarra, Iain S. Duff, Danny C. Sorensen, and Henk A. van der Vorst, *Numerical Linear Algebra for High-Performance Computers*
R. B. Lehoucq, D. C. Sorensen, and C. Yang, *ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*
Randolph E. Bank, *PLTMG: A Software Package for Solving Elliptic Partial Differential Equations, Users' Guide 8.0*
L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley, *ScaLAPACK Users' Guide*
Greg Astfalk, editor, *Applications on Advanced Architecture Computers*
Françoise Chaitin-Chatelin and Valérie Frayssé, *Lectures on Finite Precision Computations*
Roger W. Hockney, *The Science of Computer Benchmarking*
Richard Barrett, Michael Berry, Tony F. Chan, James Demmel, June Donato, Jack Dongarra, Victor Eijkhout, Roldan Pozo, Charles Romine, and Henk van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*
E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen, *LAPACK Users' Guide, Second Edition*
Jack J. Dongarra, Iain S. Duff, Danny C. Sorensen, and Henk van der Vorst, *Solving Linear Systems on Vector and Shared Memory Computers*
J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W. Stewart, *LINPACK Users' Guide*

LAPACK Users' Guide

Third Edition

E. Anderson
Z. Bai
C. Bischof
S. Blackford
J. Demmel
J. Dongarra
J. Du Croz
A. Greenbaum
S. Hammarling
A. McKenney
D. Sorensen

SOFTWARE · ENVIRONMENTS · TOOLS

siam
Society for Industrial and Applied Mathematics
Philadelphia

© 1999 by the Society for Industrial and Applied Mathematics.

10 9 8 7 6 5 4 3

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

No warranties, express or implied, are made by the publisher, authors, and their employers that the programs contained in this volume are free of error. They should not be relied on as the sole basis to solve a problem whose incorrect solution could result in injury to person or property. If the programs are employed in such a manner, it is at the user's own risk and the publisher, authors, and their employers disclaim all liability for such misuse.

Library of Congress Cataloging-in-Publication Data

LAPACK users' guide / E. Anderson... [et al.]. — 3rd ed.
p.cm. — (Software, environments, tools)
Includes bibliographical references and index.
ISBN 0-89871-447-8 (pbk.)
1. FORTRAN (Computer program language) 2. C (Computer program language)
3. Subroutines (Computer programs) 4. LAPACK. I. Anderson, E., 1962- II. Series.

QA76.73.F25 L36 1999
512'.5'02855369--dc21

This book is also available in html form over the Internet. To view the html file use the following URL:
http://www.netlib.org/lapack/lug/lapack_lug.html

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

siam is a registered trademark.

Dedication

This work is dedicated to Jim Wilkinson whose ideas and spirit have given us inspiration and influenced the project at every turn.

Authors' Affiliations:

E. Anderson
University of Tennessee, Knoxville

Z. Bai
University of Kentucky and
University of California, Davis

C. Bischof
Institute of Scientific Computing, Technical University Aachen, Germany

L. S. Blackford (formerly L. S. Ostrouchov)
University of Tennessee, Knoxville

J. Demmel
University of California, Berkeley

J. Dongarra
University of Tennessee, Knoxville and
Oak Ridge National Laboratory

J. Du Croz
Numerical Algorithms Group Ltd. (retired)

A. Greenbaum
University of Washington

S. Hammarling
Numerical Algorithms Group Ltd.

A. McKenney

D. Sorensen
Rice University

Contents

Preface to the Third Edition xv

Preface to the Second Edition xix

1 Guide 1

1 Essentials 3

1.1 LAPACK 3

1.2 Problems that LAPACK can Solve 3

1.3 Computers for which LAPACK is Suitable 4

1.4 LAPACK Compared with LINPACK and EISPACK 4

1.5 LAPACK and the BLAS 4

1.6 Availability of LAPACK 5

1.7 Commercial Use of LAPACK 6

1.8 Installation of LAPACK 6

1.9 Documentation for LAPACK 7

1.10 Support for LAPACK 7

1.11 Errata in LAPACK 8

1.12 Other Related Software 8

2 Contents of LAPACK 9

2.1 What's new in version 3.0? 9

2.2 Structure of LAPACK 11

2.2.1 Levels of Routines 11
2.2.2 Data Types and Precision 11
2.2.3 Naming Scheme 12
2.3 Driver Routines 14
2.3.1 Linear Equations 14
2.3.2 Linear Least Squares (LLS) Problems 15
2.3.3 Generalized Linear Least Squares (LSE and GLM) Problems 16
2.3.4 Standard Eigenvalue and Singular Value Problems 17
2.3.4.1 Symmetric Eigenproblems (SEP) 17
2.3.4.2 Nonsymmetric Eigenproblems (NEP) 18
2.3.4.3 Singular Value Decomposition (SVD) 19
2.3.5 Generalized Eigenvalue and Singular Value Problems 20
2.3.5.1 Generalized Symmetric Definite Eigenproblems (GSEP) 20
2.3.5.2 Generalized Nonsymmetric Eigenproblems (GNEP) 21
2.3.5.3 Generalized Singular Value Decomposition (GSVD) 23
2.4 Computational Routines 25
2.4.1 Linear Equations 25
2.4.2 Orthogonal Factorizations and Linear Least Squares Problems 31
2.4.2.1 QR Factorization 31
2.4.2.2 LQ Factorization 32
2.4.2.3 QR Factorization with Column Pivoting 32
2.4.2.4 Complete Orthogonal Factorization 34
2.4.2.5 Other Factorizations 34
2.4.3 Generalized Orthogonal Factorizations and Linear Least Squares Problems . 35
2.4.3.1 Generalized QR Factorization 35
2.4.3.2 Generalized RQ Factorization 37
2.4.4 Symmetric Eigenproblems 39
2.4.5 Nonsymmetric Eigenproblems 42
2.4.5.1 Eigenvalues, Eigenvectors and Schur Factorization 42

2.4.5.2 Balancing 42
2.4.5.3 Invariant Subspaces and Condition Numbers 43
2.4.6 Singular Value Decomposition 44
2.4.7 Generalized Symmetric Definite Eigenproblems 46
2.4.8 Generalized Nonsymmetric Eigenproblems 48
2.4.8.1 Eigenvalues, Eigenvectors and Generalized Schur Decomposition . . 48
2.4.8.2 Balancing 50
2.4.8.3 Deflating Subspaces and Condition Numbers 51
2.4.9 Generalized (or Quotient) Singular Value Decomposition 51

3 Performance of LAPACK **54**
3.1 Factors that Affect Performance 54
3.1.1 Vectorization 55
3.1.2 Data Movement 55
3.1.3 Parallelism 55
3.2 The BLAS as the Key to Portability 55
3.3 Block Algorithms and their Derivation 56
3.4 Examples of Block Algorithms in LAPACK 59
3.4.1 Factorizations for Solving Linear Equations 60
3.4.2 QR Factorization 63
3.4.3 Eigenvalue Problems 63
3.5 LAPACK Benchmark 67

4 Accuracy and Stability **77**
4.1 Sources of Error in Numerical Calculations 77
4.1.1 Further Details: Floating Point Arithmetic 78
4.2 How to Measure Errors 80
4.2.1 Further Details: How to Measure Errors 83
4.3 Further Details: How Error Bounds Are Derived 85
4.3.1 Standard Error Analysis 85

4.3.2 Improved Error Bounds 87
4.4 Error Bounds for Linear Equation Solving 88
4.4.1 Further Details: Error Bounds for Linear Equation Solving 89
4.5 Error Bounds for Linear Least Squares Problems 91
4.5.1 Further Details: Error Bounds for Linear Least Squares Problems 93
4.6 Error Bounds for Generalized Least Squares Problems 94
4.6.1 Linear Equality Constrained Least Squares Problem 94
4.6.1.1 Further Details: Error Bounds for Linear Equality Constrained Least Squares Problems 97
4.6.2 General Linear Model Problem 98
4.6.2.1 Further Details: Error Bounds for General Linear Model Problems . 102
4.7 Error Bounds for the Symmetric Eigenproblem 103
4.7.1 Further Details: Error Bounds for the Symmetric Eigenproblem 104
4.8 Error Bounds for the Nonsymmetric Eigenproblem 106
4.8.1 Further Details: Error Bounds for the Nonsymmetric Eigenproblem 107
4.8.1.1 Overview 107
4.8.1.2 Balancing and Conditioning 109
4.8.1.3 Computing s and sep 110
4.9 Error Bounds for the Singular Value Decomposition 111
4.9.1 Further Details: Error Bounds for the Singular Value Decomposition 112
4.10 Error Bounds for the Generalized Symmetric Definite Eigenproblem 114
4.10.1 Further Details: Error Bounds for the Generalized Symmetric Definite Eigenproblem 117
4.11 Error Bounds for the Generalized Nonsymmetric Eigenproblem 119
4.11.1 Further Details: Error Bounds for the Generalized Nonsymmetric Eigenproblem 121
4.11.1.1 Overview 121
4.11.1.2 Balancing and Conditioning 124
4.11.1.3 Computing s_i, $l_{\mathcal{I}}$, $r_{\mathcal{I}}$ and Dif_u, Dif_l 125
4.11.1.4 Singular Eigenproblems 128
4.12 Error Bounds for the Generalized Singular Value Decomposition 129

4.12.1 Further Details: Error Bounds for the Generalized Singular Value Decomposition . . . 131

4.13 Error Bounds for Fast Level 3 BLAS . . . 132

5 Documentation and Software Conventions 134

5.1 Design and Documentation of Argument Lists . . . 134

5.1.1 Structure of the Documentation . . . 134

5.1.2 Order of Arguments . . . 134

5.1.3 Argument Descriptions . . . 135

5.1.4 Option Arguments . . . 136

5.1.5 Problem Dimensions . . . 136

5.1.6 Array Arguments . . . 136

5.1.7 Work Arrays . . . 137

5.1.8 LWORK Query . . . 137

5.1.9 Error Handling and the Diagnostic Argument INFO . . . 137

5.2 Determining the Block Size for Block Algorithms . . . 138

5.3 Matrix Storage Schemes . . . 139

5.3.1 Conventional Storage . . . 139

5.3.2 Packed Storage . . . 140

5.3.3 Band Storage . . . 141

5.3.4 Tridiagonal and Bidiagonal Matrices . . . 142

5.3.5 Unit Triangular Matrices . . . 142

5.3.6 Real Diagonal Elements of Complex Matrices . . . 143

5.4 Representation of Orthogonal or Unitary Matrices . . . 143

6 Installing LAPACK Routines 145

6.1 Points to Note . . . 145

6.2 Installing ILAENV . . . 146

7 Troubleshooting 150

7.1 Installation Debugging Hints . . . 151

7.2 Common Errors in Calling LAPACK Routines 151
7.3 Failures Detected by LAPACK Routines 152
7.3.1 Invalid Arguments and XERBLA 152
7.3.2 Computational Failures and INFO > 0 152
7.4 Wrong Results 153
7.5 Poor Performance 154

A Index of Driver and Computational Routines 156

B Index of Auxiliary Routines 169

C Quick Reference Guide to the BLAS 180

D Converting from LINPACK or EISPACK 185

E LAPACK Working Notes 194

2 Specifications of Routines 205

Bibliography 383

Index by Keyword 391

Index by Routine Name 397

List of Tables

2.1 Matrix types in the LAPACK naming scheme 13

2.2 Driver routines for linear equations 15

2.3 Driver routines for linear least squares problems 16

2.4 Driver routines for generalized linear least squares problems 17

2.5 Driver routines for standard eigenvalue and singular value problems 20

2.6 Driver routines for generalized eigenvalue and singular value problems 25

2.7 Computational routines for linear equations 29

2.8 Computational routines for linear equations (continued) 30

2.9 Computational routines for orthogonal factorizations 35

2.10 Computational routines for the symmetric eigenproblem 41

2.11 Computational routines for the nonsymmetric eigenproblem 44

2.12 Computational routines for the singular value decomposition 46

2.13 Reduction of generalized symmetric definite eigenproblems to standard problems . . 47

2.14 Computational routines for the generalized symmetric definite eigenproblem 48

2.15 Computational routines for the generalized nonsymmetric eigenproblem 52

2.16 Computational routines for the generalized singular value decomposition 53

3.1 Speed in megaflops of Level 2 and Level 3 BLAS operations on an SGI Origin 2000 . 57

3.2 Characteristics of the Compaq/Digital computers timed 59

3.3 Characteristics of the IBM computers timed 60

3.4 Characteristics of the Intel computers timed 60

3.5 Characteristics of the SGI computer timed 61

3.6 Characteristics of the Sun computers timed 61

3.7 Speed in megaflops of DGETRF for square matrices of order n 62

3.8 Speed in megaflops of DPOTRF for matrices of order n with UPLO = 'U' 62

3.9 Speed in megaflops of DSYTRF for matrices of order n with UPLO = 'U' on an IBM Power 3 62

3.10 Speed in megaflops of DGEQRF for square matrices of order n 63

3.11 Speed in megaflops of reductions to condensed forms on an IBM Power 3 65

3.12 Execution time and Megaflop rates for DGEMV and DGEMM 73

3.13 "Standard" floating point operation counts for LAPACK drivers for n-by-n matrices 73

3.14 Performance of DGESV for n-by-n matrices 73

3.15 Performance of DGEEV, eigenvalues only 74

3.16 Performance of DGEEV, eigenvalues and right eigenvectors 74

3.17 Performance of DGESDD, singular values only 75

3.18 Performance of DGESVD, singular values and left and right singular vectors 75

3.19 Performance of DGESDD, singular values and left and right singular vectors 76

4.1 Values of Machine Parameters in IEEE Floating Point Arithmetic 79

4.2 Vector and matrix norms 81

4.3 Bounding One Vector Norm in Terms of Another 84

4.4 Bounding One Matrix Norm in Terms of Another 85

4.5 Asymptotic error bounds for the nonsymmetric eigenproblem 108

4.6 Global error bounds for the nonsymmetric eigenproblem assuming $\|E\|_F < s_i \cdot \mathrm{sep}_i/4$ 108

4.7 Asymptotic error bounds for the generalized nonsymmetric eigenvalue problem . . . 122

4.8 Global error bounds for the generalized nonsymmetric eigenvalue problem assuming $\delta \equiv \|(E,F)\|_F/\Delta < 1$. 123

6.1 Use of the block parameters NB, NBMIN, and NX in LAPACK 147

Preface to the Third Edition

Since the release of version 2.0 of the LAPACK software and the second edition of the Users' Guide in 1994, LAPACK has been expanded further and has become an even wider community effort. The publication of this third edition of the Users' Guide coincides with the release of version 3.0 of the LAPACK software. Some of the software contributors to this release were not original LAPACK authors, and thus their names have been credited in the routines to which they contributed.

Release 3.0 of LAPACK introduces new routines, as well as extending the functionality of existing routines. The most significant new routines and functions are:

1. a faster singular value decomposition (SVD), computed by divide-and-conquer (xGESDD)
2. faster routines for solving rank-deficient least squares problems:
   - using QR with column pivoting (xGELSY, based on xGEQP3)
   - using the SVD based on divide-and-conquer (xGELSD)
3. new routines for the generalized symmetric eigenproblem:
   - xHEGVD/xSYGVD, xHPGVD/xSPGVD, xHBGVD/xSBGVD: faster routines based on divide-and-conquer
   - xHEGVX/xSYGVX, xHPGVX/xSPGVX, xHBGVX/xSBGVX: routines based on bisection/inverse iteration to more efficiently compute a subset of the spectrum
4. faster routines for the symmetric eigenproblem using the "relative robust representation" algorithm (xSYEVR/xHEEVR, xSTEVR, xSTEGR)
5. faster routine for the symmetric eigenproblem using "relatively robust eigenvector algorithm" (xSTEGR, xSYEVR/xHEEVR, SSTEVR)
6. new simple and expert drivers for the generalized nonsymmetric eigenproblem (xGGES, xGGEV, xGGESX, xGGEVX), including error bounds
7. a solver for the generalized Sylvester equation (xTGSYL), used in 5)
8. computational routines (xTGEXC, xTGSEN, xTGSNA) used in 5))
9. a blocked version of xTZRQF (xTZRZF), and associated xORMRZ/xUNMRZ

All LAPACK routines reflect the current version number with the date on the routine indicating when it was last modified. For more information on revisions to the LAPACK software or this Users' Guide please refer to the LAPACK `release_notes` file on netlib. Instructions for obtaining this file can be found in Chapter 1.

The following additions/modifications have been made to this third edition of the Users' Guide:

Chapter 1 (Essentials) includes updated information on accessing LAPACK and related projects via the World Wide Web.

Chapter 2 (Contents of LAPACK) has been expanded to discuss the new routines.

Chapter 3 (Performance of LAPACK) has been updated with performance results for version 3.0 of LAPACK.

Chapter 4 (Accuracy and Stability) has been extended to include error bounds for generalized least squares.

Appendices A and B have been expanded to cover the new routines.

Appendix E (LAPACK Working Notes) lists a number of new Working Notes, written during the LAPACK 2 and ScaLAPACK projects (see below) and published by the University of Tennessee. The Bibliography has been updated to give the most recent published references.

The Specifications of Routines have been extended and updated to cover the new routines and revisions to existing routines.

The original LAPACK project was funded by the NSF. Since its completion, four follow-up projects, LAPACK 2, ScaLAPACK, ScaLAPACK 2 and LAPACK 3 have been funded in the U.S. by the NSF and ARPA in 1990–1994, 1991–1995, 1995–1998, and 1998–2001, respectively.

In addition to making possible the additions and extensions in this release, these grants have supported the following closely related activities.

A major effort is underway to implement LAPACK-type algorithms for distributed memory machines. As a result of these efforts, several new software items are now available on netlib. The new items that have been introduced are distributed memory versions of the core routines from LAPACK; sparse Gaussian elimination – SuperLU, SuperLU_MT, and distributed-memory SuperLU; a fully parallel package to solve a symmetric positive definite sparse linear system on a message passing multiprocessor using Cholesky factorization; a package based on Arnoldi's method for solving large-scale nonsymmetric, symmetric, and generalized algebraic eigenvalue problems; and templates for sparse iterative methods for solving $Ax = b$. For more information on the availability of each of these packages, consult the following URLs:

`http://www.netlib.org/scalapack/`
`http://www.netlib.org/linalg/`

Alternative language interfaces to LAPACK (or translations/conversions of LAPACK) are available in Fortran 95, C, and Java. For more information consult Section 1.12 or the following URLs:

`http://www.netlib.org/lapack90/`

```
http://www.netlib.org/clapack/
http://www.netlib.org/java/f2j/
```

The performance results presented in this book were obtained using computer resources at various sites:

- Compaq AlphaServer DS-20, donated by Compaq Corporation, and located at the Innovative Computing Laboratory, in the Department of Computer Science, University of Tennessee, Knoxville.
- IBM Power 3, donated by IBM, and located at the Innovative Computing Laboratory, in the Department of Computer Science, University of Tennessee, Knoxville.
- Intel Pentium III, donated by Intel Corporation, and located at the Innovative Computing Laboratory, in the Department of Computer Science, University of Tennessee, Knoxville.
- Clusters of Pentium IIs, PowerPCs, and Alpha EV56s, located at the LIP (Laboratoire de l'Informatique du Parallélisme), ENS (École Normale Supérieure), Lyon, France.
- SGI Origin 2000, located at the Army Research Laboratory in Aberdeen Proving Ground, Maryland, and supported by the DoD High Performance Computing Modernization Program ARL Major Shared Resource Center through Programming Environment and Training (PET) under Contract Number DAHC-94-96-C-0010, Raytheon E-Systems, subcontract no. AA23.

We would like to thank the following people, who were either not acknowledged in previous editions, or who have made significant additional contributions to this edition:

Henri Casanova, Tzu-Yi Chen, David Day, Inderjit Dhillon, Mark Fahey, Patrick Geoffray, Ming Gu, Greg Henry, Nick Higham, Bo Kågström, Linda Kaufman, John Lewis, Ren-Cang Li, Osni Marques, Rolf Neubert, Beresford Parlett, Antoine Petitet, Peter Poromaa, Gregorio Quintana, Huan Ren, Jeff Rutter, Keith Seymour, Vasile Sima, Ken Stanley, Xiaobai Sun, Françoise Tisseur, Zachary Walker, and Clint Whaley.

As before, the royalties from the sales of this book are being placed in a fund to help students attend SIAM meetings and other SIAM related activities. This fund is administered by SIAM and qualified individuals are encouraged to write directly to SIAM for guidelines.

This page intentionally left blank

Preface to the Second Edition

Since its initial public release in February 1992, LAPACK has expanded in both depth and breadth. LAPACK is now available in both Fortran and C. The publication of this second edition of the Users' Guide coincides with the release of version 2.0 of the LAPACK software.

This release of LAPACK introduces new routines and extends the functionality of existing routines. Prominent among the new routines are driver and computational routines for the generalized nonsymmetric eigenproblem, generalized linear least squares problems, the generalized singular value decomposition, a generalized banded symmetric-definite eigenproblem, and divide-and-conquer methods for symmetric eigenproblems. Additional computational routines include the generalized QR and RQ factorizations and reduction of a band matrix to bidiagonal form.

Added functionality has been incorporated into the expert driver routines that involve equilibration (xGESVX, xGBSVX, xPOSVX, xPPSVX, and xPBSVX). The option FACT = 'F' now permits the user to input a prefactored, pre-equilibrated matrix. The expert drivers xGESVX and xGBSVX now return the reciprocal of the pivot growth from Gaussian elimination. xBDSQR has been modified to compute singular values of bidiagonal matrices much more quickly than before, provided singular vectors are not also wanted. The least squares driver routines xGELS, xGELSS, and xGELSX now make available the residual root-sum-squares for each right hand side.

All LAPACK routines reflect the current version number with the date on the routine indicating when it was last modified. For more information on revisions to the LAPACK software or this Users' Guide please refer to the LAPACK `release_notes` file on netlib. Instructions for obtaining this file can be found in Chapter 1.

On-line manpages (troff files) for LAPACK routines, as well as for most of the BLAS routines, are available on netlib. Refer to Section 1.9 for further details.

We hope that future releases of LAPACK will include routines for reordering eigenvalues in the generalized Schur factorization; solving the generalized Sylvester equation; computing condition numbers for the generalized eigenproblem (for eigenvalues, eigenvectors, clusters of eigenvalues, and deflating subspaces); fast algorithms for the singular value decomposition based on divide and conquer; high accuracy methods for symmetric eigenproblems and the SVD based on Jacobi's algorithm; updating and/or downdating for linear least squares problems; computing singular values by bidiagonal bisection; and computing singular vectors by bidiagonal inverse iteration.

The following additions/modifications have been made to this second edition of the Users' Guide:

Chapter 1 (Essentials) now includes information on accessing LAPACK via the World Wide Web.

Chapter 2 (Contents of LAPACK) has been expanded to discuss new routines.

Chapter 3 (Performance of LAPACK) has been updated with performance results from version 2.0 of LAPACK. In addition, a new section entitled "LAPACK Benchmark" has been introduced to present timings for several driver routines.

Chapter 4 (Accuracy and Stability) has been simplified and rewritten. Much of the theory and other details have been separated into "Further Details" sections. Example Fortran code segments are included to demonstrate the calculation of error bounds using LAPACK.

Appendices A, B, and D have been expanded to cover the new routines.

Appendix E (LAPACK Working Notes) lists a number of new Working Notes, written during the LAPACK 2 and ScaLAPACK projects (see below) and published by the University of Tennessee. The Bibliography has been updated to give the most recent published references.

The Specifications of Routines have been extended and updated to cover the new routines and revisions to existing routines.

The Bibliography and Index have been moved to the end of the book. The Index has been expanded into two indexes: Index by Keyword and Index by Routine Name. Occurrences of LAPACK, LINPACK, and EISPACK routine names have been cited in the latter index.

The original LAPACK project was funded by the NSF. Since its completion, two follow-up projects, LAPACK 2 and ScaLAPACK, have been funded in the U.S. by the NSF and ARPA in 1990–1994 and 1991–1995, respectively. In addition to making possible the additions and extensions in this release, these grants have supported the following closely related activities.

A major effort is underway to implement LAPACK-type algorithms for distributed memory machines. As a result of these efforts, several new software items are now available on netlib. The new items that have been introduced are distributed memory versions of the core routines from LAPACK; a fully parallel package to solve a symmetric positive definite sparse linear system on a message passing multiprocessor using Cholesky factorization; a package based on Arnoldi's method for solving large-scale nonsymmetric, symmetric, and generalized algebraic eigenvalue problems; and templates for sparse iterative methods for solving $Ax = b$. For more information on the availability of each of these packages, consult the `scalapack` and `linalg` indexes on netlib via `netlib@www.netlib.org`.

We have also explored the advantages of IEEE floating point arithmetic [4] in implementing linear algebra routines. The accurate rounding properties and "friendly" exception handling capabilities of IEEE arithmetic permit us to write faster, more robust versions of several algorithms in LAPACK. Since all machines do not yet implement IEEE arithmetic, these algorithms are not currently part of the library [33], although we expect them to be in the future. For more information, please refer to Section 1.12.

LAPACK has been translated from Fortran into C and, in addition, a subset of the LAPACK routines has been implemented in C++. For more information on obtaining the C or C++ versions of LAPACK, consult Section 1.12 or the `clapack` or `c++` indexes on netlib via `netlib@www.netlib.org`.

We deeply appreciate the careful scrutiny of those individuals who reported mistakes, typographical errors, or shortcomings in the first edition.

We acknowledge with gratitude the support which we have received from the following organizations and the help of individual members of their staff: Cray Research Inc.; NAG Ltd.

We would additionally like to thank the following people, who were not acknowledged in the first edition, for their contributions:

Françoise Chatelin, Inderjit Dhillon, Stan Eisenstat, Vince Fernando, Ming Gu, Rencang Li, Xiaoye Li, George Ostrouchov, Antoine Petitet, Chris Puscasiu, Huan Ren, Jeff Rutter, Ken Stanley, Steve Timson, and Clint Whaley.

As before, the royalties from the sales of this book are being placed in a fund to help students attend SIAM meetings and other SIAM related activities. This fund is administered by SIAM and qualified individuals are encouraged to write directly to SIAM for guidelines.

This page intentionally left blank

Part 1

Guide

This page intentionally left blank

Chapter 1

Essentials

1.1 LAPACK

LAPACK is a library of Fortran 77 subroutines for solving the most commonly occurring problems in numerical linear algebra. It has been designed to be efficient on a wide range of modern high-performance computers. The name LAPACK is an acronym for Linear Algebra PACKage.

`http://www.netlib.org/lapack/`

A list of LAPACK Frequently Asked Questions (FAQ) is maintained on this webpage.

1.2 Problems that LAPACK can Solve

LAPACK can solve systems of linear equations, linear least squares problems, eigenvalue problems and singular value problems. LAPACK can also handle many associated computations such as matrix factorizations or estimating condition numbers.

LAPACK contains **driver routines** for solving standard types of problems, **computational routines** to perform a distinct computational task, and **auxiliary routines** to perform a certain subtask or common low-level computation. Each driver routine typically calls a sequence of computational routines. Taken as a whole, the computational routines can perform a wider range of tasks than are covered by the driver routines. Many of the auxiliary routines may be of use to numerical analysts or software developers, so we have documented the Fortran source for these routines with the same level of detail used for the LAPACK routines and driver routines.

Dense and band matrices are provided for, but not general sparse matrices. In all areas, similar functionality is provided for real and complex matrices. See Chapter 2 for a complete summary of the contents.

1.3 Computers for which LAPACK is Suitable

LAPACK is designed to give high efficiency on vector processors, high-performance "super-scalar" workstations, and shared memory multiprocessors. It can also be used satisfactorily on all types of scalar machines (PC's, workstations, mainframes). A distributed-memory version of LAPACK, ScaLAPACK[17], has been developed for other types of parallel architectures (for example, massively parallel SIMD machines, or distributed memory machines). See Chapter 3 for some examples of the performance achieved by LAPACK routines.

1.4 LAPACK Compared with LINPACK and EISPACK

LAPACK has been designed to supersede LINPACK [38] and EISPACK [92, 54], principally by restructuring the software to achieve much greater efficiency, where possible, on modern high-performance computers; also by adding extra functionality, by using some new or improved algorithms, and by integrating the two sets of algorithms into a unified package.

Appendix D lists the LAPACK counterparts of LINPACK and EISPACK routines. Not all the facilities of LINPACK and EISPACK are covered by Release 3.0 of LAPACK.

1.5 LAPACK and the BLAS

LAPACK routines are written so that as much as possible of the computation is performed by calls to the Basic Linear Algebra Subprograms (BLAS) [78, 42, 40]. Highly efficient machine-specific implementations of the BLAS are available for many modern high-performance computers. The BLAS enable LAPACK routines to achieve high performance with portable code. The methodology for constructing LAPACK routines in terms of calls to the BLAS is described in Chapter 3.

The BLAS are not strictly speaking part of LAPACK, but Fortran 77 code for the BLAS is distributed with LAPACK, or can be obtained separately from *netlib*. This code constitutes the "model implementation" [41, 39].

`http://www.netlib.org/blas/blas.tgz`

The model implementation is not expected to perform as well as a specially tuned implementation on most high-performance computers — on some machines it may give *much* worse performance — but it allows users to run LAPACK codes on machines that do not offer any other implementation of the BLAS.

For information on available optimized BLAS libraries, as well as other BLAS-related questions, please refer to the BLAS FAQ:

`http://www.netlib.org/blas/faq.html`

1.6 Availability of LAPACK

The complete LAPACK package or individual routines from LAPACK are freely available on *netlib* [44] and can be obtained via the World Wide Web or anonymous ftp.

The LAPACK homepage can be accessed on the World Wide Web via the URL address:

```
http://www.netlib.org/lapack/
```

Prebuilt LAPACK libraries are available on *netlib* for a variety of architectures.

```
http://www.netlib.org/lapack/archives/
```

The main *netlib* servers are:

Tennessee, U.S.A.	`http://www.netlib.org/`
New Jersey, U.S.A.	`http://netlib.bell-labs.com/`
Kent, UK	`http://www.mirror.ac.uk/sites/netlib.bell-labs.com/netlib/master/`
Bergen, Norway	`http://www.netlib.no/`

Each of these sites is the master location for some significant part of the netlib collection of software; a distributed replication facility keeps them synchronized nightly.

There are also a number of mirror repositories located around the world, and a list of these sites is maintained on *netlib.*

```
http://www.netlib.org/bib/mirrors.html
```

Most of the sites provide both ftp and http access. If ftp and http are difficult, a user may wish to send e-mail. E.g.,

```
echo ``send index from lapack'' | mail netlib@www.netlib.org
```

General information about LAPACK can be obtained by contacting any of the URLs listed above. If additional information is desired, feel free to contact the authors at `lapack@cs.utk.edu`.

The complete package, including test code and timing programs in four different Fortran data types, constitutes some 805,000 lines of Fortran source and comments.

Alternatively, if a user does not have internet access, netlib software, as well as other mathematical and statistical freeware, is available on CD-ROM from the following two companies.

Prime Time Freeware	Walnut Creek CDROM
370 Altair Way, Suite 150	4041 Pike Lane, Ste D-902
Sunnyvale, CA 94086, USA	Concord CA 94520, USA
`http://www.ptf.com/`	`http://www.cdrom.com/`
`info@ptf.com`	`orders@cdrom.com`
Tel: +1 408-433-9662	Tel: +1 800-786-9907
Fax: +1 408-433-0727	Fax: +1 510-674-0821

1.7 Commercial Use of LAPACK

LAPACK is a freely available software package provided on the World Wide Web via *netlib,* anonymous ftp, and http access. Thus it can be included in commercial packages (and has been). We ask only that proper credit be given to the authors by citing this users' guide as the official reference for LAPACK.

Like all software, this package is copyrighted. It is not trademarked; however, if modifications are made that affect the interface, functionality, or accuracy of the resulting software, the name of the routine should be changed. Any modification to our software should be noted in the modifier's documentation.

We will gladly answer questions regarding our software. If modifications are made to the software, however, it is the responsibility of the individuals/company who modified the routine to provide support.

1.8 Installation of LAPACK

To ease the installation process, prebuilt LAPACK libraries are available on *netlib* for a variety of architectures.

`http://www.netlib.org/lapack/archives/`

Included with each prebuilt library archive is the make include file `make.inc` detailing the compiler options, and so on, used to compile the library. If a prebuilt library is not available for the specific architecture, the user will need to download the source code from *netlib*

```
http://www.netlib.org/lapack/lapack.tgz
http://www.netlib.org/lapack/lapack-pc.zip
```

and build the library as instructed in the LAPACK Installation Guide [3, 37]. Note that separate distribution tar/zip files are provided for Unix/Linux and Windows 98/NT installations. Sample `make.inc` files for various architectures are included in the distribution tar/zip file and will require only limited modifications to customize for a specific architecture.

Machine-specific installation hints are contained in the `release_notes` file on *netlib.*

`http://www.netlib.org/lapack/release_notes`

A comprehensive test suite for the BLAS is provided in the LAPACK distribution and on *netlib*, and it is highly recommended that this test suite, as well as the LAPACK test suite, be run to ensure proper installation of the package.

Two installations guides are available for LAPACK. A Quick Installation Guide (LAPACK Working Note 81) [37] is distributed with the complete package. This Quick Installation Guide provides installation instructions for Unix/Linux systems. A comprehensive Installation Guide [3] (LAPACK Working Note 41), which contains descriptions of the testing and timings programs, as well as detailed non-Unix installation instructions, is also available. See also Chapter 6.

1.9 Documentation for LAPACK

This **Users' Guide** gives an informal introduction to the design of the package, and a detailed description of its contents. Chapter 5 explains the conventions used in the software and documentation. Part 2 contains complete specifications of all the driver routines and computational routines. These specifications have been derived from the leading comments in the source text.

On-line manpages (troff files) for LAPACK routines, as well as for most of the BLAS routines, are available on netlib. These files are automatically generated at the time of each release. For more information, see the `manpages.tgz` entry on the `lapack` index on netlib.

1.10 Support for LAPACK

LAPACK has been thoroughly tested before release, on many different types of computers. The LAPACK project supports the package in the sense that reports of errors or poor performance will gain immediate attention from the developers. Such reports — and also descriptions of interesting applications and other comments — should be sent to:

LAPACK Project
c/o J. J. Dongarra
Computer Science Department
University of Tennessee
Knoxville, TN 37996-1301
USA
Email: lapack@cs.utk.edu

1.11 Errata in LAPACK

A list of known problems, bugs, and compiler errors for LAPACK, as well as an errata list for this guide, is maintained on *netlib*.

`http://www.netlib.org/lapack/release_notes`

This errata file, as well as an FAQ (Frequently Asked Questions) file, can be accessed via the LAPACK homepage.

1.12 Other Related Software

As previously mentioned in the Preface, many LAPACK-related software projects are currently available on netlib. Alternative language interfaces to LAPACK (or translations/conversions of LAPACK) are available in Fortran 90, C, and Java.

```
http://www.netlib.org/lapack90/
http://www.netlib.org/clapack/
http://www.netlib.org/java/f2j/
```

The ScaLAPACK (or Scalable LAPACK) library includes a subset of LAPACK routines redesigned for distributed memory message-passing MIMD computers and networks of workstations supporting MPI and/or PVM. For more detailed information please refer to the ScaLAPACK Users' Guide [17] or the ScaLAPACK homepage:

`http://www.netlib.org/scalapack/`

Chapter 2

Contents of LAPACK

2.1 What's new in version 3.0?

Version 3.0 of LAPACK introduces new routines, as well as extending the functionality of existing routines. The most significant new routines and functions are:

1. a faster singular value decomposition (SVD), computed by divide-and-conquer (xGESDD)
2. faster routines for solving rank-deficient least squares problems:
   - using QR with column pivoting (xGELSY, based on xGEQP3)
   - using the SVD based on divide-and-conquer (xGELSD)
3. new routines for the generalized symmetric eigenproblem:
   - xHEGVD/xSYGVD, xHPGVD/xSPGVD, xHBGVD/xSBGVD: faster routines based on divide-and-conquer
   - xHEGVX/xSYGVX, xHPGVX/xSPGVX, xHBGVX/xSBGVX: routines based on bisection/inverse iteration to more efficiently compute a subset of the eigenvalues and/or eigenvectors
4. faster routines for the symmetric eigenproblem using the "relative robust representation" algorithm (xSYEVR/xHEEVR, xSTEVR, xSTEGR)
5. new simple and expert drivers for the generalized nonsymmetric eigenproblem (xGGES, xGGEV, xGGESX, xGGEVX), including error bounds
6. a solver for the generalized Sylvester equation (xTGSYL), used in 5)
7. computational routines (xTGEXC, xTGSEN, xTGSNA) used in 5)
8. a blocked version of xTZRQF (xTZRZF), and associated xORMRZ/xUNMRZ

One of the primary design features of the LAPACK library is that all releases are backward compatible. A user's program calling LAPACK will never fail because of a new release of the library. As a result, however, the calling sequences (or amount of workspace required) to existing routines cannot be altered. Therefore, if a performance enhancement requires a modification of this type, a new routine must be created. There are several routines included in LAPACK, version 3.0, that fall into this category. Specifically,

- xGEGS is deprecated and replaced by routine xGGES
- xGEGV is deprecated and replaced by routine xGGEV
- xGELSX is deprecated and replaced by routine xGELSY
- xGEQPF is deprecated and replaced by routine xGEQP3
- xTZRQF is deprecated and replaced by routine xTZRZF
- xLATZM is deprecated and replaced by routines xORMRZ/xUNMRZ

The "old" version of the routine is still included in the library but the user is advised to upgrade to the "new" faster version. References to the "old" versions are removed from this users' guide.

In addition to replacing the above list of routines, there are a number of other significantly faster new driver routines that we recommend in place of their older counterparts listed below. We continue to include the older drivers in this users' guide because the old drivers may use less workspace than the new drivers, and because the old drivers may be faster in certain special cases (we will continue to improve the new drivers in a future release until they completely replace their older counterparts):

- xSYEV/xHEEV and xSYEVD/xHEEVD should be replaced by xSYEVR/xHEEVR
- xSTEV and xSTEVD should be replaced by xSTEVR
- xSPEV/xHPEV should be replaced by xSPEVD/xHPEVD
- xSBEV/xHBEV should be replaced by xSBEVD/xHBEVD
- xGESVD should be replaced by xGESDD
- xSYGV/xHEGV should be replaced by xSYGVD/xHEGVD
- xSPGV/xHPGV should be replaced by xSPGVD/xHPGVD
- xSBGV/xHBGV should be replaced by xSBGVD/xHBGVD

This release of LAPACK introduces routines that exploit IEEE arithmetic. We have a prototype running of a new algorithm (xSTEGR), which may be the ultimate solution for the symmetric eigenproblem on both parallel and serial machines. This algorithm has been incorporated into the drivers xSYEVR, xHEEVR and xSTEVR for the symmetric eigenproblem, and will be propagated

into the generalized symmetric definite eigenvalue problems, the SVD, the generalized SVD and the SVD-based least squares solver. Refer to section 2.4.4 for further information. We expect to also propagate this algorithm into ScaLAPACK.

We have also incorporated the $LWORK = -1$ query capability into this release of LAPACK, whereby a user can request the amount of workspace required for a routine. For complete details, refer to section 5.1.8.

2.2 Structure of LAPACK

2.2.1 Levels of Routines

The subroutines in LAPACK are classified as follows:

- **driver** routines, each of which solves a complete problem, for example solving a system of linear equations, or computing the eigenvalues of a real symmetric matrix. Users are recommended to use a driver routine if there is one that meets their requirements. They are listed in Section 2.3.
- **computational** routines, each of which performs a distinct computational task, for example an LU factorization, or the reduction of a real symmetric matrix to tridiagonal form. Each driver routine calls a sequence of computational routines. Users (especially software developers) may need to call computational routines directly to perform tasks, or sequences of tasks, that cannot conveniently be performed by the driver routines. They are listed in Section 2.4.
- **auxiliary** routines, which in turn can be classified as follows:
  - routines that perform subtasks of block algorithms — in particular, routines that implement unblocked versions of the algorithms;
  - routines that perform some commonly required low-level computations, for example scaling a matrix, computing a matrix-norm, or generating an elementary Householder matrix; some of these may be of interest to numerical analysts or software developers and could be considered for future additions to the BLAS;
  - a few extensions to the BLAS, such as routines for applying complex plane rotations, or matrix-vector operations involving complex symmetric matrices (the BLAS themselves are not strictly speaking part of LAPACK).

Both driver routines and computational routines are fully described in this Users' Guide, but not the auxiliary routines. A list of the auxiliary routines, with brief descriptions of their functions, is given in Appendix B.

2.2.2 Data Types and Precision

LAPACK provides the same range of functionality for **real** and **complex** data.

For most computations there are matching routines, one for real and one for complex data, but there are a few exceptions. For example, corresponding to the routines for real symmetric indefinite systems of linear equations, there are routines for complex Hermitian and complex symmetric systems, because both types of complex systems occur in practical applications. However, there is no complex analogue of the routine for finding selected eigenvalues of a real symmetric tridiagonal matrix, because a complex Hermitian matrix can always be reduced to a real symmetric tridiagonal matrix.

Matching routines for real and complex data have been coded to maintain a close correspondence between the two, wherever possible. However, in some areas (especially the nonsymmetric eigenproblem) the correspondence is necessarily weaker.

All routines in LAPACK are provided in both **single** and **double** precision versions. The double precision versions have been generated automatically, using Toolpack/1 [88].

Double precision routines for complex matrices require the non-standard Fortran data type COMPLEX*16, which is available on most machines where double precision computation is usual.

2.2.3 Naming Scheme

The name of each LAPACK routine is a coded specification of its function (within the very tight limits of standard Fortran 77 6-character names).

All driver and computational routines have names of the form **XYYZZZ**, where for some driver routines the 6th character is blank.

The first letter, **X**, indicates the data type as follows:

S REAL
D DOUBLE PRECISION
C COMPLEX
Z COMPLEX*16 or DOUBLE COMPLEX

When we wish to refer to an LAPACK routine generically, regardless of data type, we replace the first letter by "x". Thus xGESV refers to any or all of the routines SGESV, CGESV, DGESV and ZGESV.

The next two letters, **YY**, indicate the type of matrix (or of the most significant matrix). Most of these two-letter codes apply to both real and complex matrices; a few apply specifically to one or the other, as indicated in Table 2.1.

When we wish to refer to a class of routines that performs the same function on different types of matrices, we replace the first three letters by "xyy". Thus xyySVX refers to all the expert driver routines for systems of linear equations that are listed in Table 2.2.

The last three letters **ZZZ** indicate the computation performed. Their meanings will be explained in Section 2.4. For example, SGEBRD is a single precision routine that performs a bidiagonal reduction (BRD) of a real general matrix.

Table 2.1: Matrix types in the LAPACK naming scheme

BD	bidiagonal
DI	diagonal
GB	general band
GE	general (i.e., unsymmetric, in some cases rectangular)
GG	general matrices, generalized problem (i.e., a pair of general matrices)
GT	general tridiagonal
HB	(complex) Hermitian band
HE	(complex) Hermitian
HG	upper Hessenberg matrix, generalized problem (i.e a Hessenberg and a triangular matrix)
HP	(complex) Hermitian, packed storage
HS	upper Hessenberg
OP	(real) orthogonal, packed storage
OR	(real) orthogonal
PB	symmetric or Hermitian positive definite band
PO	symmetric or Hermitian positive definite
PP	symmetric or Hermitian positive definite, packed storage
PT	symmetric or Hermitian positive definite tridiagonal
SB	(real) symmetric band
SP	symmetric, packed storage
ST	(real) symmetric tridiagonal
SY	symmetric
TB	triangular band
TG	triangular matrices, generalized problem (i.e., a pair of triangular matrices)
TP	triangular, packed storage
TR	triangular (or in some cases quasi-triangular)
TZ	trapezoidal
UN	(complex) unitary
UP	(complex) unitary, packed storage

The names of auxiliary routines follow a similar scheme except that the 2nd and 3rd characters YY are usually LA (for example, SLASCL or CLARFG). There are two kinds of exception. Auxiliary routines that implement an unblocked version of a block algorithm have similar names to the routines that perform the block algorithm, with the sixth character being "2" (for example, SGETF2 is the unblocked version of SGETRF). A few routines that may be regarded as extensions to the BLAS are named according to the BLAS naming schemes (for example, CROT, CSYR).

2.3 Driver Routines

This section describes the driver routines in LAPACK. Further details on the terminology and the numerical operations they perform are given in Section 2.4, which describes the computational routines.

2.3.1 Linear Equations

Two types of driver routines are provided for solving systems of linear equations:

- a **simple** driver (name ending -SV), which solves the system $AX = B$ by factorizing A and overwriting B with the solution X;
- an **expert** driver (name ending -SVX), which can also perform the following functions (some of them optionally):
  - solve $A^T X = B$ or $A^H X = B$ (unless A is symmetric or Hermitian);
  - estimate the condition number of A, check for near-singularity, and check for pivot growth;
  - refine the solution and compute forward and backward error bounds;
  - equilibrate the system if A is poorly scaled.

  The expert driver requires roughly twice as much storage as the simple driver in order to perform these extra functions.

Both types of driver routines can handle multiple right hand sides (the columns of B).

Different driver routines are provided to take advantage of special properties or storage schemes of the matrix A, as shown in Table 2.2.

These driver routines cover all the functionality of the computational routines for linear systems, except matrix inversion. It is seldom necessary to compute the inverse of a matrix explicitly, and it is certainly not recommended as a means of solving linear systems.

Table 2.2: Driver routines for linear equations

Type of matrix and storage scheme	Operation	Single precision		Double precision	
		real	complex	real	complex
general	simple driver	SGESV	CGESV	DGESV	ZGESV
	expert driver	SGESVX	CGESVX	DGESVX	ZGESVX
general band	simple driver	SGBSV	CGBSV	DGBSV	ZGBSV
	expert driver	SGBSVX	CGBSVX	DGBSVX	ZGBSVX
general tridiagonal	simple driver	SGTSV	CGTSV	DGTSV	ZGTSV
	expert driver	SGTSVX	CGTSVX	DGTSVX	ZGTSVX
symmetric/Hermitian positive definite	simple driver	SPOSV	CPOSV	DPOSV	ZPOSV
	expert driver	SPOSVX	CPOSVX	DPOSVX	ZPOSVX
symmetric/Hermitian positive definite (packed storage)	simple driver	SPPSV	CPPSV	DPPSV	ZPPSV
	expert driver	SPPSVX	CPPSVX	DPPSVX	ZPPSVX
symmetric/Hermitian positive definite band	simple driver	SPBSV	CPBSV	DPBSV	ZPBSV
	expert driver	SPBSVX	CPBSVX	DPBSVX	ZPBSVX
symmetric/Hermitian positive definite tridiagonal	simple driver	SPTSV	CPTSV	DPTSV	ZPTSV
	expert driver	SPTSVX	CPTSVX	DPTSVX	ZPTSVX
symmetric/Hermitian indefinite	simple driver	SSYSV	CHESV	DSYSV	ZHESV
	expert driver	SSYSVX	CHESVX	DSYSVX	ZHESVX
complex symmetric	simple driver		CSYSV		ZSYSV
	expert driver		CSYSVX		ZSYSVX
symmetric/Hermitian indefinite (packed storage)	simple driver	SSPSV	CHPSV	DSPSV	ZHPSV
	expert driver	SSPSVX	CHPSVX	DSPSVX	ZHPSVX
complex symmetric (packed storage)	simple driver		CSPSV		ZSPSV
	expert driver		CSPSVX		ZSPSVX

2.3.2 Linear Least Squares (LLS) Problems

The **linear least squares problem** is:

$$\underset{x}{\text{minimize}} \ \|b - Ax\|_2 \tag{2.1}$$

where A is an m-by-n matrix, b is a given m element vector and x is the n element solution vector.

In the most usual case $m \geq n$ and $\text{rank}(A) = n$, and in this case the solution to problem (2.1) is unique, and the problem is also referred to as finding a **least squares solution** to an **overdetermined** system of linear equations.

When $m < n$ and $\text{rank}(A) = m$, there are an infinite number of solutions x which exactly satisfy $b - Ax = 0$. In this case it is often useful to find the unique solution x which minimizes $\|x\|_2$, and the problem is referred to as finding a **minimum norm solution** to an **underdetermined** system of linear equations.

The driver routine xGELS solves problem (2.1) on the assumption that $\text{rank}(A) = \min(m, n)$ — in other words, A has **full rank** — finding a least squares solution of an overdetermined system when $m > n$, and a minimum norm solution of an underdetermined system when $m < n$. xGELS uses a QR or LQ factorization of A, and also allows A to be replaced by A^T in the statement of the problem (or by A^H if A is complex).

In the general case when we may have $\text{rank}(A) < \min(m, n)$ — in other words, A may be **rank-deficient** — we seek the **minimum norm least squares** solution x which minimizes both $\|x\|_2$ and $\|b - Ax\|_2$.

The driver routines xGELSX, xGELSY, xGELSS, and xGELSD, solve this general formulation of problem 2.1, allowing for the possibility that A is rank-deficient; xGELSX and xGELSY use a **complete orthogonal factorization** of A, while xGELSS uses the **singular value decomposition** of A, and xGELSD uses the **singular value decomposition** of A with an algorithm based on divide and conquer.

The subroutine xGELSY is a faster version of xGELSX, but requires more workspace since it calls blocked algorithms to perform the complete orthogonal factorization. xGELSX has been retained for compatibility with Release 2.0 of LAPACK, but we omit references to this routine in the remainder of this users' guide.

The subroutine xGELSD is significantly faster than its older counterpart xGELSS, especially for large problems, but may require somewhat more workspace depending on the matrix dimensions.

The LLS driver routines are listed in Table 2.3.

All four routines allow several right hand side vectors b and corresponding solutions x to be handled in a single call, storing these vectors as columns of matrices B and X, respectively. Note however that problem 2.1 is solved for each right hand side vector independently; this is *not* the same as finding a matrix X which minimizes $\|B - AX\|_2$.

Table 2.3: Driver routines for linear least squares problems

Operation	Single precision		Double precision	
	real	complex	real	complex
solve LLS using QR or LQ factorization	SGELS	CGELS	DGELS	ZGELS
solve LLS using complete orthogonal factorization	SGELSY	CGELSY	DGELSY	ZGELSY
solve LLS using SVD	SGELSS	CGELSS	DGELSS	ZGELSS
solve LLS using divide-and-conquer SVD	SGELSD	CGELSD	DGELSD	ZGELSD

2.3.3 Generalized Linear Least Squares (LSE and GLM) Problems

Driver routines are provided for two types of generalized linear least squares problems.

The first is

$$\min_x \|c - Ax\|_2 \quad \text{subject to} \quad Bx = d \tag{2.2}$$

where A is an m-by-n matrix and B is a p-by-n matrix, c is a given m-vector, and d is a given p-vector, with $p \leq n \leq m+p$. This is called a **linear equality-constrained least squares problem (LSE)**. The routine xGGLSE solves this problem using the generalized RQ (GRQ) factorization, on the assumptions that B has full row rank p and the matrix $\begin{pmatrix} A \\ B \end{pmatrix}$ has full column rank n. Under these assumptions, the problem LSE has a unique solution.

The second generalized linear least squares problem is

$$\min_x \|y\|_2 \quad \text{subject to} \quad d = Ax + By \tag{2.3}$$

where A is an n-by-m matrix, B is an n-by-p matrix, and d is a given n-vector, with $m \leq n \leq m+p$. This is sometimes called a **general** (Gauss-Markov) **linear model problem (GLM)**. When $B = I$, the problem reduces to an ordinary linear least squares problem. When B is square and nonsingular, the GLM problem is equivalent to the **weighted linear least squares problem**:

$$\min_x \|B^{-1}(d - Ax)\|_2$$

The routine xGGGLM solves this problem using the generalized QR (GQR) factorization, on the assumptions that A has full column rank m, and the matrix (A, B) has full row rank n. Under these assumptions, the problem is always consistent, and there are unique solutions x and y. The driver routines for generalized linear least squares problems are listed in Table 2.4.

Table 2.4: Driver routines for generalized linear least squares problems

Operation	Single precision		Double precision	
	real	complex	real	complex
solve LSE problem using GRQ	SGGLSE	CGGLSE	DGGLSE	ZGGLSE
solve GLM problem using GQR	SGGGLM	CGGGLM	DGGGLM	ZGGGLM

2.3.4 Standard Eigenvalue and Singular Value Problems

2.3.4.1 Symmetric Eigenproblems (SEP)

The **symmetric eigenvalue problem** is to find the **eigenvalues**, λ, and corresponding **eigenvectors**, $z \neq 0$, such that

$$Az = \lambda z, \quad A = A^T, \text{ where } A \text{ is real.}$$

For the **Hermitian eigenvalue problem** we have

$$Az = \lambda z, \quad A = A^H.$$

For both problems the eigenvalues λ are real.

When all eigenvalues and eigenvectors have been computed, we write:

$$A = Z\Lambda Z^T$$

where Λ is a diagonal matrix whose diagonal elements are the eigenvalues, and Z is an orthogonal (or unitary) matrix whose columns are the eigenvectors. This is the classical **spectral factorization** of A.

There are four types of driver routines for symmetric and Hermitian eigenproblems. Originally LAPACK had just the simple and expert drivers described below, and the other two were added after improved algorithms were discovered. Ultimately we expect the algorithm in the most recent driver (called RRR below) to supersede all the others, but in LAPACK 3.0 the other drivers may still be faster on some problems, so we retain them.

- A **simple** driver (name ending -EV) computes all the eigenvalues and (optionally) eigenvectors.
- An **expert** driver (name ending -EVX) computes all or a selected subset of the eigenvalues and (optionally) eigenvectors. If few enough eigenvalues or eigenvectors are desired, the expert driver is faster than the simple driver.
- A **divide-and-conquer** driver (name ending -EVD) solves the same problem as the simple driver. It is much faster than the simple driver for large matrices, but uses more workspace. The name divide-and-conquer refers to the underlying algorithm (see sections 2.4.4 and 3.4.3).
- A **relatively robust representation** (RRR) driver (name ending -EVR) computes all or (in a later release) a subset of the eigenvalues, and (optionally) eigenvectors. It is the fastest algorithm of all (except for a few cases), and uses the least workspace. The name RRR refers to the underlying algorithm (see sections 2.4.4 and 3.4.3).

Different driver routines are provided to take advantage of special structure or storage of the matrix A, as shown in Table 2.5.

2.3.4.2 Nonsymmetric Eigenproblems (NEP)

The **nonsymmetric eigenvalue problem** is to find the **eigenvalues**, λ, and corresponding **eigenvectors**, $v \neq 0$, such that

$$Av = \lambda v.$$

A real matrix A may have complex eigenvalues, occurring as complex conjugate pairs. More precisely, the vector v is called a **right eigenvector** of A, and a vector $u \neq 0$ satisfying

$$u^H A = \lambda u^H$$

is called a **left eigenvector** of A.

This problem can be solved via the **Schur factorization** of A, defined in the real case as

$$A = ZTZ^T,$$

where Z is an orthogonal matrix and T is an upper quasi-triangular matrix with 1-by-1 and 2-by-2 diagonal blocks, the 2-by-2 blocks corresponding to complex conjugate pairs of eigenvalues of A. In the complex case the Schur factorization is

$$A = ZTZ^H,$$

where Z is unitary and T is a complex upper triangular matrix.

The columns of Z are called the **Schur vectors**. For each k ($1 \leq k \leq n$), the first k columns of Z form an orthonormal basis for the **invariant subspace** corresponding to the first k eigenvalues on the diagonal of T. Because this basis is orthonormal, it is preferable in many applications to compute Schur vectors rather than eigenvectors. It is possible to order the Schur factorization so that any desired set of k eigenvalues occupy the k leading positions on the diagonal of T.

Two pairs of drivers are provided, one pair focusing on the Schur factorization, and the other pair on the eigenvalues and eigenvectors as shown in Table 2.5:

- xGEES: a simple driver that computes all or part of the Schur factorization of A, with optional ordering of the eigenvalues;
- xGEESX: an expert driver that can additionally compute condition numbers for the average of a selected subset of the eigenvalues, and for the corresponding right invariant subspace;
- xGEEV: a simple driver that computes all the eigenvalues of A, and (optionally) the right or left eigenvectors (or both);
- xGEEVX: an expert driver that can additionally balance the matrix to improve the conditioning of the eigenvalues and eigenvectors, and compute condition numbers for the eigenvalues or right eigenvectors (or both).

2.3.4.3 Singular Value Decomposition (SVD)

The **singular value decomposition** of an m-by-n matrix A is given by

$$A = U\Sigma V^T, \quad (A = U\Sigma V^H \quad \text{in the complex case})$$

where U and V are orthogonal (unitary) and Σ is an m-by-n diagonal matrix with real diagonal elements, σ_i, such that

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{\min(m,n)} \geq 0.$$

The σ_i are the **singular values** of A and the first $\min(m, n)$ columns of U and V are the **left** and **right singular vectors** of A.

The singular values and singular vectors satisfy:

$$Av_i = \sigma_i u_i \quad \text{and} \quad A^T u_i = \sigma_i v_i \quad (\text{or} \quad A^H u_i = \sigma_i v_i \quad)$$

where u_i and v_i are the ith columns of U and V respectively.

There are two types of driver routines for the SVD. Originally LAPACK had just the simple driver described below, and the other one was added after an improved algorithm was discovered.

- a **simple** driver xGESVD computes all the singular values and (optionally) left and/or right singular vectors.
- a **divide and conquer** driver xGESDD solves the same problem as the simple driver. It is much faster than the simple driver for large matrices, but uses more workspace. The name divide-and-conquer refers to the underlying algorithm (see sections 2.4.4 and 3.4.3).

Table 2.5: Driver routines for standard eigenvalue and singular value problems

Type of problem	Function and storage scheme	Single precision		Double precision	
		real	complex	real	complex
SEP	simple driver	SSYEV	CHEEV	DSYEV	ZHEEV
	divide and conquer driver	SSYEVD	CHEEVD	DSYEVD	ZHEEVD
	expert driver	SSYEVX	CHEEVX	DSYEVX	ZHEEVX
	RRR driver	SSYEVR	CHEEVR	DSYEVR	ZHEEVR
	simple driver (packed storage)	SSPEV	CHPEV	DSPEV	ZHPEV
	divide and conquer driver (packed storage)	SSPEVD	CHPEVD	DSPEVD	ZHPEVD
	expert driver (packed storage)	SSPEVX	CHPEVX	DSPEVX	ZHPEVX
	simple driver (band matrix)	SSBEV	CHBEV	DSBEV	ZHBEV
	divide and conquer driver (band matrix)	SSBEVD	CHBEVD	DSBEVD	ZHBEVD
	expert driver (band matrix)	SSBEVX	CHBEVX	DSBEVX	ZHBEVX
	simple driver (tridiagonal matrix)	SSTEV		DSTEV	
	divide and conquer driver (tridiagonal matrix)	SSTEVD		DSTEVD	
	expert driver (tridiagonal matrix)	SSTEVX		DSTEVX	
	RRR driver (tridiagonal matrix)	SSTEVR		DSTEVR	
NEP	simple driver for Schur factorization	SGEES	CGEES	DGEES	ZGEES
	expert driver for Schur factorization	SGEESX	CGEESX	DGEESX	ZGEESX
	simple driver for eigenvalues/vectors	SGEEV	CGEEV	DGEEV	ZGEEV
	expert driver for eigenvalues/vectors	SGEEVX	CGEEVX	DGEEVX	ZGEEVX
SVD	simple driver	SGESVD	CGESVD	DGESVD	ZGESVD
	divide and conquer driver	SGESDD	CGESDD	DGESDD	ZGESDD

2.3.5 Generalized Eigenvalue and Singular Value Problems

2.3.5.1 Generalized Symmetric Definite Eigenproblems (GSEP)

Drivers are provided to compute all the eigenvalues and (optionally) the eigenvectors of the following types of problems:

1. $Az = \lambda Bz$
2. $ABz = \lambda z$
3. $BAz = \lambda z$

where A and B are symmetric or Hermitian and B is positive definite. For all these problems the eigenvalues λ are real. The matrices Z of computed eigenvectors satisfy $Z^T AZ = \Lambda$ (problem types 1 and 3) or $Z^{-1}AZ^{-T} = I$ (problem type 2), where Λ is a diagonal matrix with the eigenvalues on the diagonal. Z also satisfies $Z^T BZ = I$ (problem types 1 and 2) or $Z^T B^{-1}Z = I$ (problem type 3).

There are three types of driver routines for generalized symmetric and Hermitian eigenproblems. Originally LAPACK had just the simple and expert drivers described below, and the other one was added after an improved algorithm was discovered.

- a **simple** driver (name ending -GV) computes all the eigenvalues and (optionally) eigenvectors.
- an **expert** driver (name ending -GVX) computes all or a selected subset of the eigenvalues and (optionally) eigenvectors. If few enough eigenvalues or eigenvectors are desired, the expert driver is faster than the simple driver.
- a **divide-and-conquer** driver (name ending -GVD) solves the same problem as the simple driver. It is much faster than the simple driver for large matrices, but uses more workspace. The name divide-and-conquer refers to the underlying algorithm (see sections 2.4.4 and 3.4.3).

Different driver routines are provided to take advantage of special structure or storage of the matrices A and B, as shown in Table 2.6.

2.3.5.2 Generalized Nonsymmetric Eigenproblems (GNEP)

Given a matrix pair (A, B), where A and B are square $n \times n$ matrices, the **generalized nonsymmetric eigenvalue problem** is to find the **eigenvalues** λ and corresponding **eigenvectors** $x \neq 0$ such that

$$Ax = \lambda Bx,$$

or to find the eigenvalues μ and corresponding eigenvectors $y \neq 0$ such that

$$\mu Ay = By.$$

Note that these problems are equivalent with $\mu = 1/\lambda$ and $x = y$ if neither λ nor μ is zero. In order to deal with the case that λ or μ is zero, or nearly so, the LAPACK routines return two values, α and β, for each eigenvalue, such that $\lambda = \alpha/\beta$ and $\mu = \beta/\alpha$.

More precisely, x and y are called **right eigenvectors**. Vectors $u \neq 0$ or $v \neq 0$ satisfying

$$u^H A = \lambda u^H B \quad \text{or} \quad \mu v^H A = v^H B$$

are called **left eigenvectors**.

Sometimes the following, equivalent notation is used to refer to the generalized eigenproblem for the pair (A, B): The object $A - \lambda B$, where λ is an indeterminate, is called a **matrix pencil**, or just **pencil**. So one can also refer to the generalized eigenvalues and eigenvectors of the pencil $A - \lambda B$.

If the determinant of $A - \lambda B$ is identically zero for all values of λ, the eigenvalue problem is called **singular**; otherwise it is **regular**. Singularity of (A, B) is signaled by some $\alpha = \beta = 0$ (in the presence of roundoff, α and β may be very small). In this case, the eigenvalue problem is very ill-conditioned, and in fact some of the other nonzero values of α and β may be indeterminate (see section 4.11.1.4 for further discussion) [93, 105, 29]. The current routines in LAPACK are intended only for regular matrix pencils.

The generalized nonsymmetric eigenvalue problem can be solved via the **generalized Schur decomposition** of the matrix pair (A, B), defined in the *real case* as

$$A = QSZ^T, \quad B = QTZ^T$$

where Q and Z are orthogonal matrices, T is upper triangular, and S is an upper quasi-triangular matrix with 1-by-1 and 2-by-2 diagonal blocks, the 2-by-2 blocks corresponding to complex conjugate pairs of eigenvalues of (A, B). In the *complex case*, the generalized Schur decomposition is

$$A = QSZ^H, \quad B = QTZ^H$$

where Q and Z are unitary and S and T are both upper triangular.

The columns of Q and Z are called **left and right generalized Schur vectors** and span pairs of **deflating subspaces** of A and B [93]. Deflating subspaces are a generalization of invariant subspaces: For each k $(1 \leq k \leq n)$, the first k columns of Z span a right deflating subspace mapped by both A and B into a left deflating subspace spanned by the first k columns of Q.

More formally, let $Q = (Q_1, Q_2)$ and $Z = (Z_1, Z_2)$ be a conformal partitioning with respect to the cluster of k eigenvalues in the (1,1)-block of (S, T), i.e. where Q_1 and Z_1 both have k columns, and S_{11} and T_{11} below are both k-by-k,

$$\begin{pmatrix} Q_1^H \\ Q_2^H \end{pmatrix} (A - \lambda B) \begin{pmatrix} Z_1 & Z_2 \end{pmatrix} = S - \lambda T \equiv \begin{pmatrix} S_{11} & S_{12} \\ 0 & S_{22} \end{pmatrix} - \lambda \begin{pmatrix} T_{11} & T_{12} \\ 0 & T_{22} \end{pmatrix}.$$

Then subspaces $\mathcal{L} = \text{span}(Q_1)$ and $\mathcal{R} = \text{span}(Z_1)$ form a pair of (left and right) deflating subspaces associated with the cluster of (S_{11}, T_{11}), satisfying $\mathcal{L} = A\mathcal{R} + B\mathcal{R}$ and $\dim(\mathcal{L}) = \dim(\mathcal{R})$ [94, 95]. It is possible to order the generalized Schur form so that (S_{11}, T_{11}) has any desired subset of k eigenvalues, taken from the set of n eigenvalues of (S, T).

As for the standard nonsymmetric eigenproblem, two pairs of drivers are provided, one pair focusing on the generalized Schur decomposition, and the other pair on the eigenvalues and eigenvectors as shown in Table 2.6:

- xGGES: a simple driver that computes all or part of the generalized Schur decomposition of (A, B), with optional ordering of the eigenvalues;
- xGGESX: an expert driver that can additionally compute condition numbers for the average of a selected subset of eigenvalues, and for the corresponding pair of deflating subspaces;
- xGGEV: a simple driver that computes all the generalized eigenvalues of (A, B), and optionally the left or right eigenvectors (or both);
- xGGEVX: an expert driver that can additionally balance the matrix pair to improve the conditioning of the eigenvalues and eigenvectors, and compute condition numbers for the eigenvalues and/or left and right eigenvectors (or both).

To save space in Table 2.6, the word "generalized" is omitted before Schur decomposition, eigenvalues/vectors and singular values/vectors.

The subroutines xGGES and xGGEV are improved versions of the drivers, xGEGS and xGEGV, respectively. xGEGS and xGEGV have been retained for compatibility with Release 2.0 of LAPACK, but we omit references to these routines in the remainder of this users' guide.

2.3.5.3 Generalized Singular Value Decomposition (GSVD)

The **generalized (or quotient) singular value decomposition** of an m-by-n matrix A and a p-by-n matrix B is given by the pair of factorizations

$$A = U\Sigma_1[0,\ R]Q^T \quad \text{and} \quad B = V\Sigma_2[0,\ R]Q^T\ .$$

The matrices in these factorizations have the following properties:

- U is m-by-m, V is p-by-p, Q is n-by-n, and all three matrices are orthogonal. If A and B are complex, these matrices are unitary instead of orthogonal, and Q^T should be replaced by Q^H in the pair of factorizations.
- R is r-by-r, upper triangular and nonsingular. $[0, R]$ is r-by-n (in other words, the 0 is an r-by-$n-r$ zero matrix). The integer r is the rank of $\begin{pmatrix} A \\ B \end{pmatrix}$, and satisfies $r \leq n$.
- Σ_1 is m-by-r, Σ_2 is p-by-r, both are real, nonnegative and diagonal, and $\Sigma_1^T\Sigma_1 + \Sigma_2^T\Sigma_2 = I$. Write $\Sigma_1^T\Sigma_1 = \mathrm{diag}(\alpha_1^2, \ldots, \alpha_r^2)$ and $\Sigma_2^T\Sigma_2 = \mathrm{diag}(\beta_1^2, \ldots, \beta_r^2)$, where α_i and β_i lie in the interval from 0 to 1. The ratios $\alpha_1/\beta_1, \ldots, \alpha_r/\beta_r$ are called the **generalized singular values** of the pair A, B. If $\beta_i = 0$, then the generalized singular value α_i/β_i is **infinite.**

Σ_1 and Σ_2 have the following detailed structures, depending on whether $m - r \geq 0$ or $m - r < 0$. In the first case, $m - r \geq 0$, then

$$\Sigma_1 = \begin{array}{r} \\ k \\ l \\ m-k-l \end{array} \begin{array}{c} \begin{array}{cc} k & l \end{array} \\ \begin{pmatrix} I & 0 \\ 0 & C \\ 0 & 0 \end{pmatrix} \end{array} \quad \text{and} \quad \Sigma_2 = \begin{array}{r} \\ l \\ p-l \end{array} \begin{array}{c} \begin{array}{cc} k & l \end{array} \\ \begin{pmatrix} 0 & S \\ 0 & 0 \end{pmatrix} \end{array}.$$

Here l is the rank of B, $k = r - l$, C and S are diagonal matrices satisfying $C^2 + S^2 = I$, and S is nonsingular. We may also identify $\alpha_1 = \cdots = \alpha_k = 1$, $\alpha_{k+i} = c_{ii}$ for $i = 1, \ldots, l$, $\beta_1 = \cdots = \beta_k = 0$, and $\beta_{k+i} = s_{ii}$ for $i = 1, \ldots, l$. Thus, the first k generalized singular values $\alpha_1/\beta_1, \ldots, \alpha_k/\beta_k$ are infinite, and the remaining l generalized singular values are finite.

In the second case, when $m - r < 0$,

$$\Sigma_1 = \begin{array}{r} \\ k \\ m-k \end{array} \begin{array}{c} \begin{array}{ccc} k & m-k & k+l-m \end{array} \\ \begin{pmatrix} I & 0 & 0 \\ 0 & C & 0 \end{pmatrix} \end{array}$$

and

$$\Sigma_2 = \begin{array}{r} \\ m-k \\ k+l-m \\ p-l \end{array} \begin{array}{c} \begin{array}{ccc} k & m-k & k+l-m \end{array} \\ \begin{pmatrix} 0 & S & 0 \\ 0 & 0 & I \\ 0 & 0 & 0 \end{pmatrix} \end{array} .$$

Again, l is the rank of B, $k = r - l$, C and S are diagonal matrices satisfying $C^2 + S^2 = I$, S is nonsingular, and we may identify $\alpha_1 = \cdots = \alpha_k = 1$, $\alpha_{k+i} = c_{ii}$ for $i = 1, \ldots, m-k$, $\alpha_{m+1} = \cdots = \alpha_r = 0$, $\beta_1 = \cdots = \beta_k = 0$, $\beta_{k+i} = s_{ii}$ for $i = 1, \ldots, m-k$, and $\beta_{m+1} = \cdots = \beta_r = 1$. Thus, the first k generalized singular values $\alpha_1/\beta_1, \ldots, \alpha_k/\beta_k$ are infinite, and the remaining l generalized singular values are finite.

Here are some important special cases of the generalized singular value decomposition. First, if B is square and nonsingular, then $r = n$ and the generalized singular value decomposition of A and B is equivalent to the singular value decomposition of AB^{-1}, where the singular values of AB^{-1} are equal to the generalized singular values of the pair A, B:

$$AB^{-1} = (U\Sigma_1 RQ^T)(V\Sigma_2 RQ^T)^{-1} = U(\Sigma_1\Sigma_2^{-1})V^T \ .$$

Second, if the columns of $(A^T \ B^T)^T$ are orthonormal, then $r = n$, $R = I$ and the generalized singular value decomposition of A and B is equivalent to the CS (Cosine-Sine) decomposition of $(A^T \ B^T)^T$ [55]:

$$\begin{pmatrix} A \\ B \end{pmatrix} = \begin{pmatrix} U & 0 \\ 0 & V \end{pmatrix} \begin{pmatrix} \Sigma_1 \\ \Sigma_2 \end{pmatrix} Q^T \ .$$

Third, the generalized eigenvalues and eigenvectors of $A^TA - \lambda B^TB$ can be expressed in terms of the generalized singular value decomposition: Let

$$X = Q \begin{pmatrix} I & 0 \\ 0 & R^{-1} \end{pmatrix} \ .$$

Then

$$X^TA^TAX = \begin{pmatrix} 0 & 0 \\ 0 & \Sigma_1^T\Sigma_1 \end{pmatrix} \text{ and } X^TB^TBX = \begin{pmatrix} 0 & 0 \\ 0 & \Sigma_2^T\Sigma_2 \end{pmatrix} .$$

Therefore, the columns of X are the eigenvectors of $A^TA - \lambda B^TB$, and the "nontrivial" eigenvalues are the squares of the generalized singular values (see also section 2.3.5.1). "Trivial" eigenvalues

are those corresponding to the leading $n - r$ columns of X, which span the common null space of $A^T A$ and $B^T B$. The "trivial eigenvalues" are not well defined[1].

A single driver routine xGGSVD computes the generalized singular value decomposition of A and B (see Table 2.6). The method is based on the method described in [83, 10, 8].

Table 2.6: Driver routines for generalized eigenvalue and singular value problems

Type of problem	Function and storage scheme	Single precision		Double precision	
		real	complex	real	complex
GSEP	simple driver	SSYGV	CHEGV	DSYGV	ZHEGV
	divide and conquer driver	SSYGVD	CHEGVD	DSYGVD	ZHEGVD
	expert driver	SSYGVX	CHEGVX	DSYGVX	ZHEGVX
	simple driver (packed storage)	SSPGV	CHPGV	DSPGV	ZHPGV
	divide and conquer driver	SSPGVD	CHPGVD	DSPGVD	ZHPGVD
	expert driver	SSPGVX	CHPGVX	DSPGVX	ZHPGVX
	simple driver (band matrices)	SSBGV	CHBGV	DSBGV	ZHBGV
	divide and conquer driver	SSBGVD	CHBGVD	DSBGV	ZHBGVD
	expert driver	SSBGVX	CHBGVX	DSBGVX	ZHBGVX
GNEP	simple driver for Schur factorization	SGGES	CGGES	DGGES	ZGGES
	expert driver for Schur factorization	SGGESX	CGGESX	DGGESX	ZGGESX
	simple driver for eigenvalues/vectors	SGGEV	CGGEV	DGGEV	ZGGEV
	expert driver for eigenvalues/vectors	SGGEVX	CGGEVX	DGGEVX	ZGGEVX
GSVD	singular values/vectors	SGGSVD	CGGSVD	DGGSVD	ZGGSVD

2.4 Computational Routines

2.4.1 Linear Equations

We use the standard notation for a system of simultaneous linear equations:

$$Ax = b \tag{2.4}$$

where A is the **coefficient matrix**, b is the **right hand side**, and x is the **solution**. In (2.4) A is assumed to be a square matrix of order n, but some of the individual routines allow A to be rectangular. If there are several right hand sides, we write

$$AX = B \tag{2.5}$$

[1] If we tried to compute the trivial eigenvalues in the same way as the nontrivial ones, that is by taking ratios of the leading $n - r$ diagonal entries of $X^T A^T AX$ and $X^T B^T BX$, we would get 0/0. For a detailed mathematical discussion of this decomposition, see the discussion of the Kronecker Canonical Form in [53].

where the columns of B are the individual right hand sides, and the columns of X are the corresponding solutions. The basic task is to compute X, given A and B.

If A is upper or lower triangular, (2.4) can be solved by a straightforward process of backward or forward substitution. Otherwise, the solution is obtained after first factorizing A as a product of triangular matrices (and possibly also a diagonal matrix or permutation matrix).

The form of the factorization depends on the properties of the matrix A. LAPACK provides routines for the following types of matrices, based on the stated factorizations:

- **general** matrices (LU factorization with partial pivoting):

  $$A = PLU$$

  where P is a permutation matrix, L is lower triangular with unit diagonal elements (lower trapezoidal if $m > n$), and U is upper triangular (upper trapezoidal if $m < n$).

- **general band** matrices including **tridiagonal** matrices (LU factorization with partial pivoting): If A is m-by-n with kl subdiagonals and ku superdiagonals, the factorization is

  $$A = LU$$

  where L is a product of permutation and unit lower triangular matrices with kl subdiagonals, and U is upper triangular with $kl + ku$ superdiagonals.

- **symmetric and Hermitian positive definite** matrices including **band** matrices (Cholesky factorization):

  $$A = U^T U \text{ or } A = LL^T \text{(in the symmetric case)}$$

  $$A = U^H U \text{ or } A = LL^H \text{(in the Hermitian case)}$$

  where U is an upper triangular matrix and L is lower triangular.

- **symmetric and Hermitian positive definite tridiagonal** matrices (LDL^T factorization):

  $$A = UDU^T \text{ or } A = LDL^T \text{(in the symmetric case)}$$

  $$A = UDU^H \text{ or } A = LDL^H \text{(in the Hermitian case)}$$

  where U is a unit upper bidiagonal matrix, L is unit lower bidiagonal, and D is diagonal.

- **symmetric and Hermitian indefinite** matrices (symmetric indefinite factorization):

  $$A = UDU^T \text{ or } A = LDL^T \text{(in the symmetric case)}$$

  $$A = UDU^H \text{ or } A = LDL^H \text{(in the Hermitian case)}$$

  where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and D is symmetric and block diagonal with diagonal blocks of order 1 or 2.

The factorization for a general tridiagonal matrix is like that for a general band matrix with $kl = 1$ and $ku = 1$. The factorization for a symmetric positive definite band matrix with k superdiagonals (or subdiagonals) has the same form as for a symmetric positive definite matrix, but the factor U (or L) is a band matrix with k superdiagonals (subdiagonals). Band matrices use a compact band storage scheme described in section 5.3.3. LAPACK routines are also provided for symmetric matrices (whether positive definite or indefinite) using **packed** storage, as described in section 5.3.2.

While the primary use of a matrix factorization is to solve a system of equations, other related tasks are provided as well. Wherever possible, LAPACK provides routines to perform each of these tasks for each type of matrix and storage scheme (see Tables 2.7 and 2.8). The following list relates the tasks to the last 3 characters of the name of the corresponding computational routine:

xyyTRF: factorize (obviously not needed for triangular matrices);

xyyTRS: use the factorization (or the matrix A itself if it is triangular) to solve (2.5) by forward or backward substitution;

xyyCON: estimate the reciprocal of the condition number $\kappa(A) = \|A\|.\|A^{-1}\|$; Higham's modification [63] of Hager's method [59] is used to estimate $\|A^{-1}\|$, except for symmetric positive definite tridiagonal matrices for which it is computed directly with comparable efficiency [61];

xyyRFS: compute bounds on the error in the computed solution (returned by the xyyTRS routine), and refine the solution to reduce the backward error (see below);

xyyTRI: use the factorization (or the matrix A itself if it is triangular) to compute A^{-1} (not provided for band matrices, because the inverse does not in general preserve bandedness);

xyyEQU: compute scaling factors to equilibrate A (not provided for tridiagonal, symmetric indefinite, or triangular matrices). These routines do not actually scale the matrices: auxiliary routines xLAQyy may be used for that purpose — see the code of the driver routines xyySVX for sample usage.

Note that some of the above routines depend on the output of others:

xyyTRF: may work on an equilibrated matrix produced by xyyEQU and xLAQyy, if yy is one of {GE, GB, PO, PP, PB};

xyyTRS: requires the factorization returned by xyyTRF;

xyyCON: requires the norm of the original matrix A, and the factorization returned by xyyTRF;

xyyRFS: requires the original matrices A and B, the factorization returned by xyyTRF, and the solution X returned by xyyTRS;

xyyTRI: requires the factorization returned by xyyTRF.

The RFS ("refine solution") routines perform iterative refinement and compute backward and forward error bounds for the solution. Iterative refinement is done in the same precision as the input data. In particular, the residual is *not* computed with extra precision, as has been traditionally done. The benefit of this procedure is discussed in Section 4.4.

Table 2.7: Computational routines for linear equations

Type of matrix and storage scheme	Operation	Single precision		Double precision	
		real	complex	real	complex
general	factorize	SGETRF	CGETRF	DGETRF	ZGETRF
	solve using factorization	SGETRS	CGETRS	DGETRS	ZGETRS
	estimate condition number	SGECON	CGECON	DGECON	ZGECON
	error bounds for solution	SGERFS	CGERFS	DGERFS	ZGERFS
	invert using factorization	SGETRI	CGETRI	DGETRI	ZGETRI
	equilibrate	SGEEQU	CGEEQU	DGEEQU	ZGEEQU
general band	factorize	SGBTRF	CGBTRF	DGBTRF	ZGBTRF
	solve using factorization	SGBTRS	CGBTRS	DGBTRS	ZGBTRS
	estimate condition number	SGBCON	CGBCON	DGBCON	ZGBCON
	error bounds for solution	SGBRFS	CGBRFS	DGBRFS	ZGBRFS
	equilibrate	SGBEQU	CGBEQU	DGBEQU	ZGBEQU
general tridiagonal	factorize	SGTTRF	CGTTRF	DGTTRF	ZGTTRF
	solve using factorization	SGTTRS	CGTTRS	DGTTRS	ZGTTRS
	estimate condition number	SGTCON	CGTCON	DGTCON	ZGTCON
	error bounds for solution	SGTRFS	CGTRFS	DGTRFS	ZGTRFS
symmetric/Hermitian positive definite	factorize	SPOTRF	CPOTRF	DPOTRF	ZPOTRF
	solve using factorization	SPOTRS	CPOTRS	DPOTRS	ZPOTRS
	estimate condition number	SPOCON	CPOCON	DPOCON	ZPOCON
	error bounds for solution	SPORFS	CPORFS	DPORFS	ZPORFS
	invert using factorization	SPOTRI	CPOTRI	DPOTRI	ZPOTRI
	equilibrate	SPOEQU	CPOEQU	DPOEQU	ZPOEQU
symmetric/Hermitian positive definite (packed storage)	factorize	SPPTRF	CPPTRF	DPPTRF	ZPPTRF
	solve using factorization	SPPTRS	CPPTRS	DPPTRS	ZPPTRS
	estimate condition number	SPPCON	CPPCON	DPPCON	ZPPCON
	error bounds for solution	SPPRFS	CPPRFS	DPPRFS	ZPPRFS
	invert using factorization	SPPTRI	CPPTRI	DPPTRI	ZPPTRI
	equilibrate	SPPEQU	CPPEQU	DPPEQU	ZPPEQU
symmetric/Hermitian positive definite band	factorize	SPBTRF	CPBTRF	DPBTRF	ZPBTRF
	solve using factorization	SPBTRS	CPBTRS	DPBTRS	ZPBTRS
	estimate condition number	SPBCON	CPBCON	DPBCON	ZPBCON
	error bounds for solution	SPBRFS	CPBRFS	DPBRFS	ZPBRFS
	equilibrate	SPBEQU	CPBEQU	DPBEQU	ZPBEQU
symmetric/Hermitian positive definite tridiagonal	factorize	SPTTRF	CPTTRF	DPTTRF	ZPTTRF
	solve using factorization	SPTTRS	CPTTRS	DPTTRS	ZPTTRS
	estimate condition number	SPTCON	CPTCON	DPTCON	ZPTCON
	error bounds for solution	SPTRFS	CPTRFS	DPTRFS	ZPTRFS

Table 2.8: Computational routines for linear equations (continued)

Type of matrix and storage scheme	Operation	Single precision		Double precision	
		real	complex	real	complex
symmetric/Hermitian indefinite	factorize	SSYTRF	CHETRF	DSYTRF	ZHETRF
	solve using factorization	SSYTRS	CHETRS	DSYTRS	ZHETRS
	estimate condition number	SSYCON	CHECON	DSYCON	ZHECON
	error bounds for solution	SSYRFS	CHERFS	DSYRFS	ZHERFS
	invert using factorization	SSYTRI	CHETRI	DSYTRI	ZHETRI
complex symmetric	factorize		CSYTRF		ZSYTRF
	solve using factorization		CSYTRS		ZSYTRS
	estimate condition number		CSYCON		ZSYCON
	error bounds for solution		CSYRFS		ZSYRFS
	invert using factorization		CSYTRI		ZSYTRI
symmetric/Hermitian indefinite (packed storage)	factorize	SSPTRF	CHPTRF	DSPTRF	ZHPTRF
	solve using factorization	SSPTRS	CHPTRS	DSPTRS	ZHPTRS
	estimate condition number	SSPCON	CHPCON	DSPCON	ZHPCON
	error bounds for solution	SSPRFS	CHPRFS	DSPRFS	ZHPRFS
	invert using factorization	SSPTRI	CHPTRI	DSPTRI	ZHPTRI
complex symmetric (packed storage)	factorize		CSPTRF		ZSPTRF
	solve using factorization		CSPTRS		ZSPTRS
	estimate condition number		CSPCON		ZSPCON
	error bounds for solution		CSPRFS		ZSPRFS
	invert using factorization		CSPTRI		ZSPTRI
triangular	solve	STRTRS	CTRTRS	DTRTRS	ZTRTRS
	estimate condition number	STRCON	CTRCON	DTRCON	ZTRCON
	error bounds for solution	STRRFS	CTRRFS	DTRRFS	ZTRRFS
	invert	STRTRI	CTRTRI	DTRTRI	ZTRTRI
triangular (packed storage)	solve	STPTRS	CTPTRS	DTPTRS	ZTPTRS
	estimate condition number	STPCON	CTPCON	DTPCON	ZTPCON
	error bounds for solution	STPRFS	CTPRFS	DTPRFS	ZTPRFS
	invert	STPTRI	CTPTRI	DTPTRI	ZTPTRI
triangular band	solve	STBTRS	CTBTRS	DTBTRS	ZTBTRS
	estimate condition number	STBCON	CTBCON	DTBCON	ZTBCON
	error bounds for solution	STBRFS	CTBRFS	DTBRFS	ZTBRFS

2.4.2 Orthogonal Factorizations and Linear Least Squares Problems

LAPACK provides a number of routines for factorizing a general rectangular m-by-n matrix A, as the product of an **orthogonal** matrix (**unitary** if complex) and a **triangular** (or possibly trapezoidal) matrix.

A real matrix Q is **orthogonal** if $Q^TQ = I$; a complex matrix Q is **unitary** if $Q^HQ = I$. Orthogonal or unitary matrices have the important property that they leave the two-norm of a vector invariant:

$$||x||_2 = ||Qx||_2, \quad \text{if } Q \text{ is orthogonal or unitary.}$$

As a result, they help to maintain numerical stability because they do not amplify rounding errors.

Orthogonal factorizations are used in the solution of linear least squares problems. They may also be used to perform preliminary steps in the solution of eigenvalue or singular value problems.

2.4.2.1 QR Factorization

The most common, and best known, of the factorizations is the ***QR* factorization** given by

$$A = Q \begin{pmatrix} R \\ 0 \end{pmatrix}, \quad \text{if } m \geq n,$$

where R is an n-by-n upper triangular matrix and Q is an m-by-m orthogonal (or unitary) matrix. If A is of full rank n, then R is non-singular. It is sometimes convenient to write the factorization as

$$A = \begin{pmatrix} Q_1 & Q_2 \end{pmatrix} \begin{pmatrix} R \\ 0 \end{pmatrix}$$

which reduces to

$$A = Q_1 R,$$

where Q_1 consists of the first n columns of Q, and Q_2 the remaining $m - n$ columns.

If $m < n$, R is trapezoidal, and the factorization can be written

$$A = Q \begin{pmatrix} R_1 & R_2 \end{pmatrix}, \quad \text{if } m < n,$$

where R_1 is upper triangular and R_2 is rectangular.

The routine xGEQRF computes the QR factorization. The matrix Q is not formed explicitly, but is represented as a product of elementary reflectors, as described in section 5.4. Users need not be aware of the details of this representation, because associated routines are provided to work with Q: xORGQR (or xUNGQR in the complex case) can generate all or part of Q, while xORMQR (or xUNMQR) can pre- or post-multiply a given matrix by Q or Q^T (Q^H if complex).

The QR factorization can be used to solve the linear least squares problem (2.1) when $m \geq n$ and A is of full rank, since

$$||b - Ax||_2 = ||Q^Tb - Q^TAx||_2 = \left\| \begin{matrix} c_1 - Rx \\ c_2 \end{matrix} \right\|_2, \quad \text{where } c \equiv \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} Q_1^Tb \\ Q_2^Tb \end{pmatrix} = Q^Tb;$$

c can be computed by xORMQR (or xUNMQR), and c_1 consists of its first n elements. Then x is the solution of the upper triangular system

$$Rx = c_1$$

which can be computed by xTRTRS. The residual vector r is given by

$$r = b - Ax = Q \begin{pmatrix} 0 \\ c_2 \end{pmatrix},$$

and may be computed using xORMQR (or xUNMQR). The residual sum of squares $\|r\|_2^2$ may be computed without forming r explicitly, since

$$\|r\|_2 = \|b - Ax\|_2 = \|c_2\|_2.$$

2.4.2.2 LQ Factorization

The ***LQ* factorization** is given by

$$A = \begin{pmatrix} L & 0 \end{pmatrix} Q = \begin{pmatrix} L & 0 \end{pmatrix} \begin{pmatrix} Q_1 \\ Q_2 \end{pmatrix} = LQ_1, \quad \text{if } m \leq n,$$

where L is m-by-m lower triangular, Q is n-by-n orthogonal (or unitary), Q_1 consists of the first m rows of Q, and Q_2 the remaining $n - m$ rows.

This factorization is computed by the routine xGELQF, and again Q is represented as a product of elementary reflectors; xORGLQ (or xUNGLQ in the complex case) can generate all or part of Q, and xORMLQ (or xUNMLQ) can pre- or post-multiply a given matrix by Q or Q^T (Q^H if Q is complex).

The LQ factorization of A is essentially the same as the QR factorization of A^T (A^H if A is complex), since

$$A = \begin{pmatrix} L & 0 \end{pmatrix} Q \quad \Longleftrightarrow \quad A^T = Q^T \begin{pmatrix} L^T \\ 0 \end{pmatrix}.$$

The LQ factorization may be used to find a minimum norm solution of an underdetermined system of linear equations $Ax = b$ where A is m-by-n with $m < n$ and has rank m. The solution is given by

$$x = Q^T \begin{pmatrix} L^{-1}b \\ 0 \end{pmatrix}$$

and may be computed by calls to xTRTRS and xORMLQ.

2.4.2.3 QR Factorization with Column Pivoting

To solve a linear least squares problem (2.1) when A is not of full rank, or the rank of A is in doubt, we can perform either a QR factorization with column pivoting or a singular value decomposition (see subsection 2.4.6).

The **QR factorization with column pivoting** is given by

$$A = Q \begin{pmatrix} R \\ 0 \end{pmatrix} P^T, \quad m \geq n,$$

where Q and R are as before and P is a permutation matrix, chosen (in general) so that

$$|r_{11}| \geq |r_{22}| \geq \cdots \geq |r_{nn}|$$

and moreover, for each k,

$$|r_{kk}| \geq \|R_{k:j,j}\|_2 \quad \text{for } j = k+1, \ldots, n.$$

In exact arithmetic, if rank$(A) = k$, then the whole of the submatrix R_{22} in rows and columns $k+1$ to n would be zero. In numerical computation, the aim must be to determine an index k, such that the leading submatrix R_{11} in the first k rows and columns is well-conditioned, and R_{22} is negligible:

$$R = \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix} \simeq \begin{pmatrix} R_{11} & R_{12} \\ 0 & 0 \end{pmatrix}.$$

Then k is the effective rank of A. See Golub and Van Loan [55] for a further discussion of numerical rank determination.

The so-called basic solution to the linear least squares problem (2.1) can be obtained from this factorization as

$$x = P \begin{pmatrix} R_{11}^{-1} \hat{c}_1 \\ 0 \end{pmatrix},$$

where $\hat{c}_1$ consists of just the first k elements of $c = Q^T b$.

The QR factorization with column pivoting can be computed either by subroutine xGEQPF or by subroutine xGEQP3. Both subroutines compute the factorization but do not attempt to determine the rank of A. xGEQP3 is a Level 3 BLAS version of QR with column pivoting and is considerably faster than xGEQPF, while maintaining the same numerical behavior. The difference between the two routines can best be described as follows. For each column, the subroutine xGEQPF selects one column, permutes it, computes the reflector that zeroes some of its components, and applies it to the rest of the matrix via Level 2 BLAS operations. The subroutine xGEQP3, however, only updates one column and one row of the rest of the matrix (information necessary for the next pivoting phase) and delays the update of the rest of the matrix until a block of columns has been processed. This resulting block of reflectors is then applied to the rest of the matrix as a Level 3 BLAS operation. xGEQPF has been retained for compatibility with Release 2.0 of LAPACK, but we omit references to this routine in the remainder of this users' guide.

For both subroutines, the matrix Q is represented in exactly the same way as after a call of xGEQRF, and so the routines xORGQR and xORMQR can be used to work with Q (xUNGQR and xUNMQR if Q is complex).

2.4.2.4 Complete Orthogonal Factorization

The QR factorization with column pivoting does not enable us to compute a *minimum norm* solution to a rank-deficient linear least squares problem, unless $R_{12} = 0$. However, by applying further orthogonal (or unitary) transformations from the right to the upper trapezoidal matrix $\begin{pmatrix} R_{11} & R_{12} \end{pmatrix}$, using the routine xTZRQF (or xTZRZF), R_{12} can be eliminated:

$$\begin{pmatrix} R_{11} & R_{12} \end{pmatrix} Z = \begin{pmatrix} T_{11} & 0 \end{pmatrix}.$$

This gives the **complete orthogonal factorization**

$$AP = Q \begin{pmatrix} T_{11} & 0 \\ 0 & 0 \end{pmatrix} Z^T$$

from which the minimum norm solution can be obtained as

$$x = PZ \begin{pmatrix} T_{11}^{-1} \hat{c}_1 \\ 0 \end{pmatrix}.$$

The matrix Z is not formed explicitly, but is represented as a product of elementary reflectors, as described in section 5.4. Users need not be aware of the details of this representation, because associated routines are provided to work with Z: xORMRZ (or xUNMRZ) can pre- or post-multiply a given matrix by Z or Z^T (Z^H if complex).

The subroutine xTZRZF is a faster and blocked version of xTZRQF. xTZRQF has been retained for compatibility with Release 2.0 of LAPACK, but we omit references to this routine in the remainder of this users' guide.

2.4.2.5 Other Factorizations

The ***QL*** and ***RQ*** **factorizations** are given by

$$A = Q \begin{pmatrix} 0 \\ L \end{pmatrix}, \quad \text{if } m \geq n,$$

and

$$A = \begin{pmatrix} 0 & R \end{pmatrix} Q, \quad \text{if } m \leq n.$$

These factorizations are computed by xGEQLF and xGERQF, respectively; they are less commonly used than either the QR or LQ factorizations described above, but have applications in, for example, the computation of generalized QR factorizations [2].

All the factorization routines discussed here (except xTZRQF and xTZRZF) allow arbitrary m and n, so that in some cases the matrices R or L are trapezoidal rather than triangular. A routine that performs pivoting is provided only for the QR factorization.

Table 2.9: Computational routines for orthogonal factorizations

Type of factorization and matrix	Operation	Single precision		Double precision	
		real	complex	real	complex
QR, general	factorize with pivoting	SGEQP3	CGEQP3	DGEQP3	ZGEQP3
	factorize, no pivoting	SGEQRF	CGEQRF	DGEQRF	ZGEQRF
	generate Q	SORGQR	CUNGQR	DORGQR	ZUNGQR
	multiply matrix by Q	SORMQR	CUNMQR	DORMQR	ZUNMQR
LQ, general	factorize, no pivoting	SGELQF	CGELQF	DGELQF	ZGELQF
	generate Q	SORGLQ	CUNGLQ	DORGLQ	ZUNGLQ
	multiply matrix by Q	SORMLQ	CUNMLQ	DORMLQ	ZUNMLQ
QL, general	factorize, no pivoting	SGEQLF	CGEQLF	DGEQLF	ZGEQLF
	generate Q	SORGQL	CUNGQL	DORGQL	ZUNGQL
	multiply matrix by Q	SORMQL	CUNMQL	DORMQL	ZUNMQL
RQ, general	factorize, no pivoting	SGERQF	CGERQF	DGERQF	ZGERQF
	generate Q	SORGRQ	CUNGRQ	DORGRQ	ZUNGRQ
	multiply matrix by Q	SORMRQ	CUNMRQ	DORMRQ	ZUNMRQ
RZ, trapezoidal	factorize, no pivoting (blocked algorithm)	STZRZF	CTZRZF	DTZRZF	ZTZRZF
	multiply matrix by Q	SORMRZ	CUNMRZ	DORMRZ	ZUNMRZ

2.4.3 Generalized Orthogonal Factorizations and Linear Least Squares Problems

2.4.3.1 Generalized QR Factorization

The **generalized QR (GQR) factorization** of an n-by-m matrix A and an n-by-p matrix B is given by the pair of factorizations

$$A = QR \quad \text{and} \quad B = QTZ$$

where Q and Z are respectively n-by-n and p-by-p orthogonal matrices (or unitary matrices if A and B are complex). R has the form:

$$R = \begin{array}{c} \\ m \\ n-m \end{array} \begin{array}{c} m \\ \begin{pmatrix} R_{11} \\ 0 \end{pmatrix} \end{array}, \quad \text{if} \quad n \geq m$$

or

$$R = \begin{array}{c} \\ n \end{array} \begin{array}{cc} n & m-n \\ \begin{pmatrix} R_{11} & R_{12} \end{pmatrix} \end{array}, \quad \text{if} \quad n < m$$

where R_{11} is upper triangular. T has the form

$$T = \begin{array}{c} \\ n \end{array}\begin{array}{c} \begin{array}{cc} p-n & n \end{array} \\ \begin{pmatrix} 0 & T_{12} \end{pmatrix} \end{array}, \quad \text{if} \quad n \le p$$

or

$$T = \begin{array}{c} \\ n-p \\ p \end{array}\begin{array}{c} p \\ \begin{pmatrix} T_{11} \\ T_{21} \end{pmatrix} \end{array}, \quad \text{if} \quad n > p$$

where T_{12} or T_{21} is upper triangular.

Note that if B is square and nonsingular, the GQR factorization of A and B implicitly gives the QR factorization of the matrix $B^{-1}A$:

$$B^{-1}A = Z^T(T^{-1}R)$$

without explicitly computing the matrix inverse B^{-1} or the product $B^{-1}A$.

The routine xGGQRF computes the GQR factorization by first computing the QR factorization of A and then the RQ factorization of Q^TB. The orthogonal (or unitary) matrices Q and Z can either be formed explicitly or just used to multiply another given matrix in the same way as the orthogonal (or unitary) matrix in the QR factorization (see section 2.4.2).

The GQR factorization was introduced in [60, 84]. The implementation of the GQR factorization here follows [2]. Further generalizations of the GQR factorization can be found in [22].

The GQR factorization can be used to solve the general (Gauss-Markov) linear model problem (GLM) (see (2.3) and [81][55, page 252]). Using the GQR factorization of A and B, we rewrite the equation $d = Ax + By$ from (2.3) as

$$\begin{aligned} Q^Td &= Q^TAx + Q^TBy \\ &= Rx + TZy. \end{aligned}$$

We partition this as

$$\begin{pmatrix} d_1 \\ d_2 \end{pmatrix} = \begin{array}{c} \\ m \\ n-m \end{array}\begin{array}{c} m \\ \begin{pmatrix} R_{11} \\ 0 \end{pmatrix} \end{array} x + \begin{array}{c} \\ m \\ n-m \end{array}\begin{array}{c} \begin{array}{cc} p-n+m & n-m \end{array} \\ \begin{pmatrix} T_{11} & T_{12} \\ 0 & T_{22} \end{pmatrix} \end{array} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$$

where

$$\begin{pmatrix} d_1 \\ d_2 \end{pmatrix} \equiv Q^Td, \quad \text{and} \quad \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \equiv Zy\,;$$

$\begin{pmatrix} d_1 \\ d_2 \end{pmatrix}$ can be computed by xORMQR (or xUNMQR).

The GLM problem is solved by setting

$$y_1 = 0 \quad \text{and} \quad y_2 = T_{22}^{-1}d_2$$

from which we obtain the desired solutions

$$x = R_{11}^{-1}(d_1 - T_{12}y_2) \quad \text{and} \quad y = Z^T \begin{pmatrix} 0 \\ y_2 \end{pmatrix},$$

which can be computed by xTRSV, xGEMV and xORMRQ (or xUNMRQ).

2.4.3.2 Generalized RQ Factorization

The **generalized RQ (GRQ) factorization** of an m-by-n matrix A and a p-by-n matrix B is given by the pair of factorizations

$$A = RQ \quad \text{and} \quad B = ZTQ$$

where Q and Z are respectively n-by-n and p-by-p orthogonal matrices (or unitary matrices if A and B are complex). R has the form

$$R = \begin{array}{c} \\ m \end{array} \begin{array}{c} \begin{array}{cc} n-m & m \end{array} \\ \begin{pmatrix} 0 & R_{12} \end{pmatrix} \end{array}, \quad \text{if} \quad m \le n$$

or

$$R = \begin{array}{c} m-n \\ n \end{array} \begin{array}{c} n \\ \begin{pmatrix} R_{11} \\ R_{21} \end{pmatrix} \end{array}, \quad \text{if} \quad m > n,$$

where R_{12} or R_{21} is upper triangular. T has the form

$$T = \begin{array}{c} n \\ p-n \end{array} \begin{array}{c} n \\ \begin{pmatrix} T_{11} \\ 0 \end{pmatrix} \end{array}, \quad \text{if} \quad p \ge n$$

or

$$T = \begin{array}{c} \\ p \end{array} \begin{array}{c} \begin{array}{cc} p & n-p \end{array} \\ \begin{pmatrix} T_{11} & T_{12} \end{pmatrix} \end{array}, \quad \text{if} \quad p < n$$

where T_{11} is upper triangular.

Note that if B is square and nonsingular, the GRQ factorization of A and B implicitly gives the RQ factorization of the matrix AB^{-1}:

$$AB^{-1} = (RT^{-1})Z^T$$

without explicitly computing the matrix inverse B^{-1} or the product AB^{-1}.

The routine xGGRQF computes the GRQ factorization by first computing the RQ factorization of A and then the QR factorization of BQ^T. The orthogonal (or unitary) matrices Q and Z can either be formed explicitly or just used to multiply another given matrix in the same way as the orthogonal (or unitary) matrix in the RQ factorization (see section 2.4.2).

The GRQ factorization can be used to solve the linear equality-constrained least squares problem (LSE) (see (2.2) and [55, page 567]). We use the GRQ factorization of B and A (note that B and A have swapped roles), written as

$$B = TQ \quad \text{and} \quad A = ZRQ.$$

We write the linear equality constraints $Bx = d$ as:

$$TQx = d$$

which we partition as:

$$\begin{array}{cc} & \begin{array}{cc} n-p & p \end{array} \\ p & \begin{pmatrix} 0 & T_{12} \end{pmatrix} \end{array} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = d \quad \text{where} \quad \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \equiv Qx.$$

Therefore x_2 is the solution of the upper triangular system

$$T_{12}x_2 = d$$

Furthermore,

$$\begin{aligned} \|Ax - c\|_2 &= \|Z^T Ax - Z^T c\|_2 \\ &= \|RQx - Z^T c\|_2. \end{aligned}$$

We partition this expression as:

$$\begin{array}{cc} & \begin{array}{cc} n-p & p \end{array} \\ \begin{array}{r} n-p \\ p+m-n \end{array} & \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix} \end{array} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} - \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}$$

where $\begin{pmatrix} c_1 \\ c_2 \end{pmatrix} \equiv Z^T c$, which can be computed by xORMQR (or xUNMQR).

To solve the LSE problem, we set

$$R_{11}x_1 + R_{12}x_2 - c_1 = 0$$

which gives x_1 as the solution of the upper triangular system

$$R_{11}x_1 = c_1 - R_{12}x_2.$$

Finally, the desired solution is given by

$$x = Q^T \begin{pmatrix} x_1 \\ x_2 \end{pmatrix},$$

which can be computed by xORMRQ (or xUNMRQ).

2.4.4 Symmetric Eigenproblems

Let A be a real symmetric or complex Hermitian n-by-n matrix. A scalar λ is called an **eigenvalue** and a nonzero column vector z the corresponding **eigenvector** if $Az = \lambda z$. λ is always real when A is real symmetric or complex Hermitian.

The basic task of the symmetric eigenproblem routines is to compute values of λ and, optionally, corresponding vectors z for a given matrix A.

This computation proceeds in the following stages:

1. The real symmetric or complex Hermitian matrix A is reduced to **real tridiagonal form** T. If A is real symmetric this decomposition is $A = QTQ^T$ with Q orthogonal and T symmetric tridiagonal. If A is complex Hermitian, the decomposition is $A = QTQ^H$ with Q unitary and T, as before, *real* symmetric tridiagonal.

2. Eigenvalues and eigenvectors of the real symmetric tridiagonal matrix T are computed. If all eigenvalues and eigenvectors are computed, this is equivalent to factorizing T as $T = S\Lambda S^T$, where S is orthogonal and Λ is diagonal. The diagonal entries of Λ are the eigenvalues of T, which are also the eigenvalues of A, and the columns of S are the eigenvectors of T; the eigenvectors of A are the columns of $Z = QS$, so that $A = Z\Lambda Z^T$ ($Z\Lambda Z^H$ when A is complex Hermitian).

In the real case, the decomposition $A = QTQ^T$ is computed by one of the routines xSYTRD, xSPTRD, or xSBTRD, depending on how the matrix is stored (see Table 2.10). The complex analogues of these routines are called xHETRD, xHPTRD, and xHBTRD. The routine xSYTRD (or xHETRD) represents the matrix Q as a product of elementary reflectors, as described in section 5.4. The routine xORGTR (or in the complex case xUNMTR) is provided to form Q explicitly; this is needed in particular before calling xSTEQR to compute all the eigenvectors of A by the QR algorithm. The routine xORMTR (or in the complex case xUNMTR) is provided to multiply another matrix by Q without forming Q explicitly; this can be used to transform eigenvectors of T computed by xSTEIN, back to eigenvectors of A.

When packed storage is used, the corresponding routines for forming Q or multiplying another matrix by Q are xOPGTR and xOPMTR (in the complex case, xUPGTR and xUPMTR).

When A is banded and xSBTRD (or xHBTRD) is used to reduce it to tridiagonal form, Q is determined as a product of Givens rotations, not as a product of elementary reflectors; if Q is required, it must be formed explicitly by the reduction routine. xSBTRD is based on the vectorizable algorithm due to Kaufman [77].

There are several routines for computing eigenvalues and eigenvectors of T, to cover the cases of computing some or all of the eigenvalues, and some or all of the eigenvectors. In addition, some routines run faster in some computing environments or for some matrices than for others. Also, some routines are more accurate than other routines.

See section 2.3.4.1 for a discussion.

xSTEQR This routine uses the implicitly shifted QR algorithm. It switches between the QR and QL variants in order to handle graded matrices more effectively than the simple QL variant that is provided by the EISPACK routines IMTQL1 and IMTQL2. See [56] for details. This routine is used by drivers with names ending in -EV and -EVX to compute all the eigenvalues and eigenvectors (see section 2.3.4.1).

xSTERF This routine uses a square-root free version of the QR algorithm, also switching between QR and QL variants, and can only compute all the eigenvalues. See [56] for details. This routine is used by drivers with names ending in -EV and -EVX to compute all the eigenvalues and no eigenvectors (see section 2.3.4.1).

xSTEDC This routine uses Cuppen's divide and conquer algorithm to find the eigenvalues and the eigenvectors (if only eigenvalues are desired, xSTEDC calls xSTERF). xSTEDC can be many times faster than xSTEQR for large matrices but needs more work space ($2n^2$ or $3n^2$). See [20, 57, 89] and section 3.4.3 for details. This routine is used by drivers with names ending in -EVD to compute all the eigenvalues and eigenvectors (see section 2.3.4.1).

xSTEGR This routine uses the relatively robust representation (RRR) algorithm to find eigenvalues and eigenvectors. This routine uses an LDL^T factorization of a number of translates $T - sI$ of T, for one shift s near each cluster of eigenvalues. For each translate the algorithm computes very accurate eigenpairs for the tiny eigenvalues. xSTEGR is faster than all the other routines except in a few cases, and uses the least workspace. See [35] and section 3.4.3 for details.

xPTEQR This routine applies to symmetric *positive definite* tridiagonal matrices only. It uses a combination of Cholesky factorization and bidiagonal QR iteration (see xBDSQR) and may be significantly more accurate than the other routines except xSTEGR. See [14, 32, 23, 51] for details.

xSTEBZ This routine uses bisection to compute some or all of the eigenvalues. Options provide for computing all the eigenvalues in a real interval or all the eigenvalues from the ith to the jth largest. It can be highly accurate, but may be adjusted to run faster if lower accuracy is acceptable. This routine is used by drivers with names ending in -EVX.

xSTEIN Given accurate eigenvalues, this routine uses inverse iteration to compute some or all of the eigenvectors. This routine is used by drivers with names ending in -EVX.

See Table 2.10.

Table 2.10: Computational routines for the symmetric eigenproblem

Type of matrix and storage scheme	Operation	Single precision		Double precision	
		real	complex	real	complex
dense symmetric (or Hermitian)	tridiagonal reduction	SSYTRD	CHETRD	DSYTRD	ZHETRD
packed symmetric (or Hermitian)	tridiagonal reduction	SSPTRD	CHPTRD	DSPTRD	ZHPTRD
band symmetric (or Hermitian)	tridiagonal reduction	SSBTRD	CHBTRD	DSBTRD	ZHBTRD
orthogonal/unitary	generate matrix after reduction by xSYTRD	SORGTR	CUNGTR	DORGTR	ZUNGTR
	multiply matrix after reduction by xSYTRD	SORMTR	CUNMTR	DORMTR	ZUNMTR
orthogonal/unitary (packed storage)	generate matrix after reduction by xSPTRD	SOPGTR	CUPGTR	DOPGTR	ZUPGTR
	multiply matrix after reduction by xSPTRD	SOPMTR	CUPMTR	DOPMTR	ZUPMTR
symmetric tridiagonal	eigenvalues/ eigenvectors via QR	SSTEQR	CSTEQR	DSTEQR	ZSTEQR
	eigenvalues only via root-free QR	SSTERF		DSTERF	
	eigenvalues/ eigenvectors via divide and conquer	SSTEDC	CSTEDC	DSTEDC	ZSTEDC
	eigenvalues/ eigenvectors via RRR	SSTEGR	CSTEGR	DSTEGR	ZSTEGR
	eigenvalues only via bisection	SSTEBZ		DSTEBZ	
	eigenvectors by inverse iteration	SSTEIN	CSTEIN	DSTEIN	ZSTEIN
symmetric tridiagonal positive definite	eigenvalues/ eigenvectors	SPTEQR	CPTEQR	DPTEQR	ZPTEQR

2.4.5 Nonsymmetric Eigenproblems

2.4.5.1 Eigenvalues, Eigenvectors and Schur Factorization

Let A be a square n-by-n matrix. A scalar λ is called an **eigenvalue** and a non-zero column vector v the corresponding **right eigenvector** if $Av = \lambda v$. A nonzero column vector u satisfying $u^H A = \lambda u^H$ is called the **left eigenvector**. The first basic task of the routines described in this section is to compute, for a given matrix A, all n values of λ and, if desired, their associated right eigenvectors v and/or left eigenvectors u.

A second basic task is to compute the **Schur factorization** of a matrix A. If A is complex, then its Schur factorization is $A = ZTZ^H$, where Z is unitary and T is upper triangular. If A is real, its Schur factorization is $A = ZTZ^T$, where Z is orthogonal. and T is upper quasi-triangular (1-by-1 and 2-by-2 blocks on its diagonal). The columns of Z are called the **Schur vectors** of A. The eigenvalues of A appear on the diagonal of T; complex conjugate eigenvalues of a real A correspond to 2-by-2 blocks on the diagonal of T.

These two basic tasks can be performed in the following stages:

1. A general matrix A is reduced to **upper Hessenberg form** H which is zero below the first subdiagonal. The reduction may be written $A = QHQ^T$ with Q orthogonal if A is real, or $A = QHQ^H$ with Q unitary if A is complex. The reduction is performed by subroutine xGEHRD, which represents Q in a factored form, as described in section 5.4. The routine xORGHR (or in the complex case xUNGHR) is provided to form Q explicitly. The routine xORMHR (or in the complex case xUNMHR) is provided to multiply another matrix by Q without forming Q explicitly.

2. The upper Hessenberg matrix H is reduced to Schur form T, giving the Schur factorization $H = STS^T$ (for H real) or $H = STS^H$ (for H complex). The matrix S (the Schur vectors of H) may optionally be computed as well. Alternatively S may be postmultiplied into the matrix Q determined in stage 1, to give the matrix $Z = QS$, the Schur vectors of A. The eigenvalues are obtained from the diagonal of T. All this is done by subroutine xHSEQR.

3. Given the eigenvalues, the eigenvectors may be computed in two different ways. xHSEIN performs inverse iteration on H to compute the eigenvectors of H; xORMHR can then be used to multiply the eigenvectors by the matrix Q in order to transform them to eigenvectors of A. xTREVC computes the eigenvectors of T, and optionally transforms them to those of H or A if the matrix S or Z is supplied. Both xHSEIN and xTREVC allow selected left and/or right eigenvectors to be computed.

Other subsidiary tasks may be performed before or after those just described.

2.4.5.2 Balancing

The routine xGEBAL may be used to **balance** the matrix A prior to reduction to Hessenberg form. Balancing involves two steps, either of which is optional:

- first, xGEBAL attempts to permute A by a similarity transformation to block upper triangular form:

$$PAP^T = A' = \begin{pmatrix} A'_{11} & A'_{12} & A'_{13} \\ 0 & A'_{22} & A'_{23} \\ 0 & 0 & A'_{33} \end{pmatrix}$$

  where P is a permutation matrix and A'_{11} and A'_{33} are *upper triangular*. Thus the matrix is already in Schur form outside the central diagonal block A'_{22} in rows and columns ILO to IHI. Subsequent operations by xGEBAL, xGEHRD or xHSEQR need only be applied to these rows and columns; therefore ILO and IHI are passed as arguments to xGEHRD and xHSEQR. This can save a significant amount of work if ILO > 1 or IHI $< n$. If no suitable permutation can be found (as is very often the case), xGEBAL sets ILO $= 1$ and IHI $= n$, and A'_{22} is the whole of A.

- secondly, xGEBAL applies a diagonal similarity transformation to A' to make the rows and columns of A'_{22} as close in norm in possible:

$$A'' = DA'D^{-1} = \begin{pmatrix} I & 0 & 0 \\ 0 & D_{22} & 0 \\ 0 & 0 & I \end{pmatrix} \begin{pmatrix} A'_{11} & A'_{12} & A'_{13} \\ 0 & A'_{22} & A'_{23} \\ 0 & 0 & A'_{33} \end{pmatrix} \begin{pmatrix} I & 0 & 0 \\ 0 & D_{22}^{-1} & 0 \\ 0 & 0 & I \end{pmatrix}$$

  This can improve the accuracy of later processing in some cases; see subsection 4.8.1.2.

If A was balanced by xGEBAL, then eigenvectors computed by subsequent operations are eigenvectors of the balanced matrix A''; xGEBAK must then be called to transform them back to eigenvectors of the original matrix A.

2.4.5.3 Invariant Subspaces and Condition Numbers

The Schur form depends on the order of the eigenvalues on the diagonal of T and this may optionally be chosen by the user. Suppose the user chooses that $\lambda_1, \ldots, \lambda_j$, $1 \le j \le n$, appear in the upper left corner of T. Then the first j columns of Z span the **right invariant subspace** of A corresponding to $\lambda_1, \ldots, \lambda_j$.

The following routines perform this re-ordering and also compute condition numbers for eigenvalues, eigenvectors, and invariant subspaces:

1. xTREXC will move an eigenvalue (or 2-by-2 block) on the diagonal of the Schur form from its original position to any other position. It may be used to choose the order in which eigenvalues appear in the Schur form.

2. xTRSYL solves the Sylvester matrix equation $AX \pm XB = C$ for X, given matrices A, B and C, with A and B (quasi) triangular. It is used in the routines xTRSNA and xTRSEN, but it is also of independent interest.

3. xTRSNA computes the condition numbers of the eigenvalues and/or right eigenvectors of a matrix T in Schur form. These are the same as the condition numbers of the eigenvalues and

right eigenvectors of the original matrix A from which T is derived. The user may compute these condition numbers for all eigenvalue/eigenvector pairs, or for any selected subset. For more details, see section 4.8 and [12].

4. xTRSEN moves a selected subset of the eigenvalues of a matrix T in Schur form to the upper left corner of T, and optionally computes the condition numbers of their average value and of their right invariant subspace. These are the same as the condition numbers of the average eigenvalue and right invariant subspace of the original matrix A from which T is derived. For more details, see section 4.8 and [12]

See Table 2.11 for a complete list of the routines.

Table 2.11: Computational routines for the nonsymmetric eigenproblem

Type of matrix and storage scheme	Operation	Single precision		Double precision	
		real	complex	real	complex
general	Hessenberg reduction	SGEHRD	CGEHRD	DGEHRD	ZGEHRD
	balancing	SGEBAL	CGEBAL	DGEBAL	ZGEBAL
	backtransforming	SGEBAK	CGEBAK	DGEBAK	ZGEBAK
orthogonal/unitary	generate matrix after Hessenberg reduction	SORGHR	CUNGHR	DORGHR	ZUNGHR
	multiply matrix after Hessenberg reduction	SORMHR	CUNMHR	DORMHR	ZUNMHR
Hessenberg	Schur factorization	SHSEQR	CHSEQR	DHSEQR	ZHSEQR
	eigenvectors by inverse iteration	SHSEIN	CHSEIN	DHSEIN	ZHSEIN
(quasi)triangular	eigenvectors	STREVC	CTREVC	DTREVC	ZTREVC
	reordering Schur factorization	STREXC	CTREXC	DTREXC	ZTREXC
	Sylvester equation	STRSYL	CTRSYL	DTRSYL	ZTRSYL
	condition numbers of eigenvalues/vectors	STRSNA	CTRSNA	DTRSNA	ZTRSNA
	condition numbers of eigenvalue cluster/ invariant subspace	STRSEN	CTRSEN	DTRSEN	ZTRSEN

2.4.6 Singular Value Decomposition

Let A be a general real m-by-n matrix. The **singular value decomposition (SVD)** of A is the factorization $A = U\Sigma V^T$, where U and V are orthogonal, and $\Sigma = \mathrm{diag}(\sigma_1, \ldots, \sigma_r)$, $r = \min(m, n)$, with $\sigma_1 \geq \cdots \geq \sigma_r \geq 0$. If A is complex, then its SVD is $A = U\Sigma V^H$ where U and V are unitary,

and Σ is as before with real diagonal elements. The σ_i are called the **singular values**, the first r columns of V the **right singular vectors** and the first r columns of U the **left singular vectors.**

The routines described in this section, and listed in Table 2.12, are used to compute this decomposition. The computation proceeds in the following stages:

1. The matrix A is reduced to bidiagonal form: $A = U_1 B V_1^T$ if A is real ($A = U_1 B V_1^H$ if A is complex), where U_1 and V_1 are orthogonal (unitary if A is complex), and B is real and upper-bidiagonal when $m \geq n$ and lower bidiagonal when $m < n$, so that B is nonzero only on the main diagonal and either on the first superdiagonal (if $m \geq n$) or the first subdiagonal (if $m < n$).
2. The SVD of the bidiagonal matrix B is computed: $B = U_2 \Sigma V_2^T$, where U_2 and V_2 are orthogonal and Σ is diagonal as described above. The singular vectors of A are then $U = U_1 U_2$ and $V = V_1 V_2$.

The reduction to bidiagonal form is performed by the subroutine xGEBRD, or by xGBBRD for a band matrix.

The routine xGEBRD represents U_1 and V_1 in factored form as products of elementary reflectors, as described in section 5.4. If A is real, the matrices U_1 and V_1 may be computed explicitly using routine xORGBR, or multiplied by other matrices without forming U_1 and V_1 using routine xORMBR. If A is complex, one instead uses xUNGBR and xUNMBR, respectively.

If A is banded and xGBBRD is used to reduce it to bidiagonal form, U_1 and V_1 are determined as products of Givens rotations , rather than as products of elementary reflectors. If U_1 or V_1 is required, it must be formed explicitly by xGBBRD. xGBBRD uses a vectorizable algorithm, similar to that used by xSBTRD (see Kaufman [77]). xGBBRD may be much faster than xGEBRD when the bandwidth is narrow.

The SVD of the bidiagonal matrix is computed either by subroutine xBDSQR or by subroutine xBDSDC. xBDSQR uses QR iteration when singular vectors are desired [32, 23], and otherwise uses the dqds algorithm [51]. xBDSQR is more accurate than its counterparts in LINPACK and EISPACK: barring underflow and overflow, it computes all the singular values of B to nearly full relative precision, independent of their magnitudes. It also computes the singular vectors much more accurately. See section 4.9 and [32, 23, 51] for details.

xBDSDC uses a variation of Cuppen's divide and conquer algorithm to find singular values and singular vectors [69, 58]. It is much faster than xBDSQR if singular vectors of large matrices are desired. When singular values only are desired, it uses the same dqds algorithm as xBDSQR [51]. Divide-and-conquer is not guaranteed to compute singular values to nearly full relative precision, but in practice xBDSDC is often at least as accurate as xBDSQR. xBDSDC represents the singular vector matrices U_2 and V_2 in a compressed format requiring only $O(n \log n)$ space instead of n^2. U_2 and V_2 may subsequently be generated explicitly using routine xLASDQ, or multiplied by vectors without forming them explicitly using routine xLASD0.

If $m \gg n$, it may be more efficient to first perform a QR factorization of A, using the routine xGEQRF , and then to compute the SVD of the n-by-n matrix R, since if $A = QR$ and $R = U\Sigma V^T$,

then the SVD of A is given by $A = (QU)\Sigma V^T$. Similarly, if $m \ll n$, it may be more efficient to first perform an LQ factorization of A, using xGELQF. These preliminary QR and LQ factorizations are performed by the drivers xGESVD and xGESDD.

The SVD may be used to find a minimum norm solution to a (possibly) rank-deficient linear least squares problem (2.1). The effective rank, k, of A can be determined as the number of singular values which exceed a suitable threshold. Let $\hat{\Sigma}$ be the leading k-by-k submatrix of Σ, and $\hat{V}$ be the matrix consisting of the first k columns of V. Then the solution is given by:

$$x = \hat{V}\hat{\Sigma}^{-1}\hat{c}_1$$

where $\hat{c}_1$ consists of the first k elements of $c = U^T b = U_2^T U_1^T b$. $U_1^T b$ can be computed using xORMBR, and xBDSQR has an option to multiply a vector by U_2^T.

Table 2.12: Computational routines for the singular value decomposition

Type of matrix and storage scheme	Operation	Single precision		Double precision	
		real	complex	real	complex
general	bidiagonal reduction	SGEBRD	CGEBRD	DGEBRD	ZGEBRD
general band	bidiagonal reduction	SGBBRD	CGBBRD	DGBBRD	ZGBBRD
orthogonal/unitary	generate matrix after bidiagonal reduction	SORGBR	CUNGBR	DORGBR	ZUNGBR
	multiply matrix after bidiagonal reduction	SORMBR	CUNMBR	DORMBR	ZUNMBR
bidiagonal	SVD using QR or dqds	SBDSQR	CBDSQR	DBDSQR	ZBDSQR
	SVD using divide-and-conquer	SBDSDC		DBDSDC	

2.4.7 Generalized Symmetric Definite Eigenproblems

This section is concerned with the solution of the generalized eigenvalue problems $Az = \lambda Bz$, $ABz = \lambda z$, and $BAz = \lambda z$, where A and B are real symmetric or complex Hermitian and B is positive definite. Each of these problems can be reduced to a standard symmetric eigenvalue problem, using a Cholesky factorization of B as either $B = LL^T$ or $B = U^T U$ (LL^H or $U^H U$ in the Hermitian case). In the case $Ax = \lambda Bz$, if A and B are banded then this may also be exploited to get a faster algorithm.

With $B = LL^T$, we have

$$Az = \lambda Bz \quad \Rightarrow \quad (L^{-1}AL^{-T})(L^T z) = \lambda(L^T z).$$

Hence the eigenvalues of $Az = \lambda Bz$ are those of $Cy = \lambda y$, where C is the symmetric matrix $C = L^{-1}AL^{-T}$ and $y = L^T z$. In the complex case C is Hermitian with $C = L^{-1}AL^{-H}$ and $y = L^H z$.

Table 2.13: Reduction of generalized symmetric definite eigenproblems to standard problems

	Type of problem	Factorization of B	Reduction	Recovery of eigenvectors
1.	$Az = \lambda Bz$	$B = LL^T$ $B = U^TU$	$C = L^{-1}AL^{-T}$ $C = U^{-T}AU^{-1}$	$z = L^{-T}y$ $z = U^{-1}y$
2.	$ABz = \lambda z$	$B = LL^T$ $B = U^TU$	$C = L^TAL$ $C = UAU^T$	$z = L^{-T}y$ $z = U^{-1}y$
3.	$BAz = \lambda z$	$B = LL^T$ $B = U^TU$	$C = L^TAL$ $C = UAU^T$	$z = Ly$ $z = U^Ty$

Table 2.13 summarizes how each of the three types of problem may be reduced to standard form $Cy = \lambda y$, and how the eigenvectors z of the original problem may be recovered from the eigenvectors y of the reduced problem. The table applies to real problems; for complex problems, transposed matrices must be replaced by conjugate-transposes.

Given A and a Cholesky factorization of B, the routines xyyGST overwrite A with the matrix C of the corresponding standard problem $Cy = \lambda y$ (see Table 2.14). This may then be solved using the routines described in subsection 2.4.4. No special routines are needed to recover the eigenvectors z of the generalized problem from the eigenvectors y of the standard problem, because these computations are simple applications of Level 2 or Level 3 BLAS.

If the problem is $Az = \lambda Bz$ and the matrices A and B are banded, the matrix C as defined above is, in general, full. We can reduce the problem to a banded standard problem by modifying the definition of C thus:

$$C = X^TAX, \quad \text{where } X = U^{-1}Q \quad \text{or} \quad L^{-T}Q,$$

where Q is an orthogonal matrix chosen to ensure that C has bandwidth no greater than that of A. Q is determined as a product of Givens rotations. This is known as Crawford's algorithm (see Crawford [19]). If X is required, it must be formed explicitly by the reduction routine.

A further refinement is possible when A and B are banded, which halves the amount of work required to form C (see Wilkinson [104]). Instead of the standard Cholesky factorization of B as U^TU or LL^T, we use a "split Cholesky" factorization $B = S^TS$ (S^HS if B is complex), where:

$$S = \begin{pmatrix} U_{11} & \\ M_{21} & L_{22} \end{pmatrix}$$

with U_{11} upper triangular and L_{22} lower triangular of order approximately $n/2$; S has the same bandwidth as B. After B has been factorized in this way by the routine xPBSTF, the reduction of the banded generalized problem $Az = \lambda Bz$ to a banded standard problem $Cy = \lambda y$ is performed by the routine xSBGST (or xHBGST for complex matrices). This routine implements a vectorizable form of the algorithm, suggested by Kaufman [77].

2.4.8 Generalized Nonsymmetric Eigenproblems

2.4.8.1 Eigenvalues, Eigenvectors and Generalized Schur Decomposition

Let A and B be n-by-n matrices. A scalar λ is called a **generalized eigenvalue** and a non-zero column vector x the corresponding **right generalized eigenvector** of the pair (A, B), if $Ax = \lambda Bx$. A non-zero column vector y satisfying $y^H A = \lambda y^H B$ is called the **left generalized eigenvector** corresponding to λ. (For simplicity, we will usually omit the word "generalized" when no confusion is likely to arise.) If B is singular, we can have the **infinite eigenvalue** $\lambda = \infty$, by which we mean $Bx = 0$. Note that if A is non-singular, then the equivalent problem $\mu Ax = Bx$ is perfectly well-defined, and the infinite eigenvalue corresponds to $\mu = 0$. The generalized symmetric definite eigenproblem in section 2.3.7 has only finite real eigenvalues. The generalized nonsymmetric eigenvalue problem can have real, complex or infinite eigenvalues. To deal with both finite (including zero) and infinite eigenvalues, the LAPACK routines return two values, α and β. If β is non-zero then $\lambda = \alpha/\beta$ is an eigenvalue. If β is zero then $\lambda = \infty$ is an eigenvalue of (A, B). (Round off may change an exactly zero β to a small nonzero value, changing the eigenvalue $\lambda = \infty$ to some very large value; see section 4.11 for details.) A basic task of these routines is to compute all n pairs (α, β) and x and/or y for a given pair of matrices (A, B).

If the determinant of $A - \lambda B$ is identically zero for all values of λ, the eigenvalue problem is called **singular**; otherwise it is **regular**. Singularity of (A, B) is signaled by some $\alpha = \beta = 0$ (in the presence of roundoff, α and β may be very small). In this case, the eigenvalue problem is very ill-conditioned, and in fact some of the other nonzero values of α and β may be indeterminate (see section 4.11.1.4 for further discussion) [93, 105, 29, 53].

Another basic task is to compute the **generalized Schur decomposition** of the pair (A, B). If A and B are complex, then their generalized Schur decomposition is $A = QSZ^H$ and $B = QTZ^H$, where Q and Z are unitary and S and T are upper triangular. The LAPACK routines normalize T to have real non-negative diagonal entries. Note that in this form, the eigenvalues can be easily computed from the diagonals: $\lambda_i = s_{ii}/t_{ii}$ (if $t_{ii} \neq 0$) and $\lambda_i = \infty$ (if $t_{ii} = 0$), and so the LAPACK routines return $\alpha_i = s_{ii}$ and $\beta_i = t_{ii}$.

The generalized Schur form depends on the order of the eigenvalues on the diagonal of (S, T). This

Table 2.14: Computational routines for the generalized symmetric definite eigenproblem

Type of matrix and storage scheme	Operation	Single precision		Double precision	
		real	complex	real	complex
symmetric/Hermitian	reduction	SSYGST	CHEGST	DSYGST	ZHEGST
symmetric/Hermitian (packed storage)	reduction	SSPGST	CHPGST	DSPGST	ZHPGST
symmetric/Hermitian banded	split Cholesky factorization	SPBSTF	CPBSTF	DPBSTF	ZPBSTF
	reduction	SSBGST	DSBGST	CHBGST	ZHBGST

order may optionally be chosen by the user.

If A and B are real, then their generalized Schur decomposition is $A = QSZ^T$ and $B = QTZ^T$, where Q and Z are orthogonal, S is quasi-upper triangular with 1-by-1 and 2-by-2 blocks on the diagonal, and T is upper triangular with non-negative diagonal entries. The structure of a typical pair of (S, T) is illustrated below for $n = 6$:

$$S = \begin{pmatrix} \times & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \\ 0 & 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & 0 & \times & \times \end{pmatrix}, \qquad T = \begin{pmatrix} \times & \times & \times & \times & \times & \times \\ 0 & \times & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times & \times \\ 0 & 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & 0 & \times & 0 \\ 0 & 0 & 0 & 0 & 0 & \times \end{pmatrix}$$

The 1×1 diagonal blocks of (S, T) (those in the (1,1) and (4,4) positions) contain the real eigenvalues of (A, B) and the 2×2 diagonal blocks of (S, T) (those in the (2:3,2:3) and (5:6,5:6) positions) contain conjugate pairs of complex eigenvalues of (A, B). The 2×2 diagonal blocks of T corresponding to 2-by-2 blocks of S are made diagonal. This arrangement enables us to work entirely with real numbers, even when some of the eigenvalues of (A, B) are complex. Note that for real eigenvalues, as for all eigenvalues in the complex case, the α_i and β_i values corresponding to real eigenvalues may be easily computed from the diagonals of S and T. The α_i and β_i values corresponding to complex eigenvalues of a 2-by-2 diagonal block of (S, T) are computed by first computing the complex conjugate eigenvalues λ and $\bar{\lambda}$ of the block, then computing the values of β_i and β_{i+1} that would result if the block were put into *complex* generalized Schur form, and finally multiplying to get $\alpha_i = \lambda\beta_i$ and $\alpha_{i+1} = \bar{\lambda}\beta_{i+1}$.

The columns of Q and Z are called **generalized Schur vectors** and span pairs of **deflating subspaces** of A and B [94]. Deflating subspaces are a generalization of invariant subspaces: the first k columns of Z span a right deflating subspace mapped by both A and B into a left deflating subspace spanned by the first k columns of Q. This pair of deflating subspaces corresponds to the first k eigenvalues appearing at the top left corner of S and T as explained in section 2.3.5.2.

The computations proceed in the following stages:

1. The pair (A, B) is reduced to **generalized upper Hessenberg form**. If A and B are real, this decomposition is $A = UHV^T$ and $B = URV^T$ where H is upper Hessenberg (zero below the first subdiagonal), R is upper triangular, and U and V are orthogonal. If A and B are complex, the decomposition is $A = UHV^H$ and $B = URV^H$ with U and V unitary, and H and R as before. This decomposition is performed by the subroutine xGGHRD, which computes H and R, and optionally U and/or V. Note that in contrast to xGEHRD (for the standard nonsymmetric eigenvalue problem), xGGHRD does not compute U and V in a factored form.

2. The pair (H, R) is reduced to generalized Schur form $H = QSZ^T$ and $R = QTZ^T$ (for H and R real) or $H = QSZ^H$ and $R = QTZ^H$ (for H and R complex) by subroutine xHGEQZ.

The values α_i and β_i are also computed, where $\lambda_i = \alpha_i/\beta_i$ are the eigenvalues. The matrices Z and Q are optionally computed.

3. The left and/or right eigenvectors of the pair (S, T) are computed by xTGEVC. One may optionally transform the right eigenvectors of (S, T) to the right eigenvectors of (A, B) (or of (H, R)) by passing (UQ, VZ) (or (Q, Z)) to xTGEVC.

Other subsidiary tasks may be performed before or after those described.

2.4.8.2 Balancing

The routine xGGBAL may be used to **balance** the matrix pair (A, B) prior to reduction to generalized Hessenberg form. Balancing involves two steps, either of which is optional:

1. First, xGGBAL attempts to permute (A, B) by an equivalence transformation to block upper triangular form:

$$P_1(A,B)P_2 = (A', B') = \left(\begin{bmatrix} A'_{11} & A'_{12} & A'_{13} \\ 0 & A'_{22} & A'_{23} \\ 0 & 0 & A'_{33} \end{bmatrix}, \begin{bmatrix} B'_{11} & B'_{12} & B'_{13} \\ 0 & B'_{22} & B'_{23} \\ 0 & 0 & B'_{33} \end{bmatrix} \right)$$

   where P_1 and P_2 are permutation matrices and A'_{11}, A'_{33}, B'_{11} and B'_{33} are *upper triangular.* Thus the matrix pair is already in generalized Schur form outside the central diagonal blocks A'_{22} and B'_{22} in rows and columns ILO to IHI. Subsequent operations by xGGBAL, xGGHRD or xHGEQZ need only be applied to these rows and columns; therefore ILO and IHI are passed as arguments to xGGHRD and xHGEQZ. This can save a significant amount of work if ILO > 1 or IHI $< n$. If no suitable permutations can be found (as is very often the case), xGGBAL sets ILO $= 1$ and IHI $= n$, and A'_{22} is the whole of A and B'_{22} is the whole of B.

2. Secondly, xGGBAL applies diagonal equivalence transformations to (A', B') to attempt to make the matrix norm smaller with respect to the eigenvalues and tries to reduce the inaccuracy contributed by roundoff [100]:

$$\begin{aligned} (A'', B'') &= D_1(A', B')D_2 \\ &= \begin{bmatrix} I & 0 & 0 \\ 0 & D'_1 & 0 \\ 0 & 0 & I \end{bmatrix} \left(\begin{bmatrix} A'_{11} & A'_{12} & A'_{13} \\ 0 & A'_{22} & A'_{23} \\ 0 & 0 & A'_{33} \end{bmatrix}, \begin{bmatrix} B'_{11} & B'_{12} & B'_{13} \\ 0 & B'_{22} & B'_{23} \\ 0 & 0 & B'_{33} \end{bmatrix} \right) \begin{bmatrix} I & 0 & 0 \\ 0 & D'_2 & 0 \\ 0 & 0 & I \end{bmatrix} \end{aligned}$$

   This can improve the accuracy of later processing in some cases; see subsection 4.11.1.2.

If the matrix pair (A, B) was balanced by xGGBAL, then eigenvectors computed by subsequent operations are eigenvectors of the balanced matrix pair (A'', B''). xGGBAK must then be called to transform them back to eigenvectors of the original matrix pair (A, B). Note that these transformations can improve speed and accuracy of later processing in some cases; however, the diagonal transformation step can occasionally make the norm of the pencil (A', B') larger and hence degrade the accuracy.

2.4.8.3 Deflating Subspaces and Condition Numbers

The generalized Schur form depends on the order of the eigenvalues on the diagonal of (S,T) and this may optionally be chosen by the user. Suppose the user chooses that $(\alpha_1,\beta_1),\ldots,(\alpha_j,\beta_j), 1 \leq j \leq n$, appear in the upper left corner of (S,T). Then the first j columns of UQ and VZ span the **left and right deflating subspaces** of (A,B) corresponding to $(\alpha_1,\beta_1),\ldots,(\alpha_j,\beta_j)$.

The following routines perform this reordering and also compute condition numbers for eigenvalues, eigenvectors and deflating subspaces:

1. xTGEXC will move an eigenvalue pair (or a pair of 2-by-2 blocks) on the diagonal of the generalized Schur form (S,T) from its original position to any other position. It may be used to choose the order in which eigenvalues appear in the generalized Schur form. The reordering is performed with orthogonal (unitary) transformation matrices. For more details see [70, 73].

2. xTGSYL solves the generalized Sylvester equations $AR - LB = sC$ and $DR - LE = sF$ for L and R, given A and B upper (quasi-)triangular and D and E upper triangular. It is also possible to solve a transposed system (conjugate transposed system in the complex case) $A^TX + D^TY = sC$ and $-XB^T - YE^T = sF$ for X and Y. The scaling factor s is set during the computations to avoid overflow. Optionally, xTGSYL computes a Frobenius norm-based estimate of the "separation" between the two matrix pairs (A,B) and (D,E). xTGSYL is used by the routines xTGSNA and xTGSEN, but it is also of independent interest. For more details see [71, 74, 75].

3. xTGSNA computes condition numbers of the eigenvalues and/or left and right eigenvectors of a matrix pair (S,T) in generalized Schur form. These are the same as the condition numbers of the eigenvalues and eigenvectors of the original matrix pair (A,B), from which (S,T) is derived. The user may compute these condition numbers for all eigenvalues and associated eigenvectors, or for any selected subset. For more details see section 4.11 and [73].

4. xTGSEN moves a selected subset of the eigenvalues of a matrix pair (S,T) in generalized Schur form to the upper left corner of (S,T), and optionally computes condition numbers of their average value and their associated pair of (left and right) deflating subspaces. These are the same as the condition numbers of the average eigenvalue and the deflating subspace pair of the original matrix pair (A,B), from which (S,T) is derived. For more details see section 4.11 and [73].

See Table 2.15 for a complete list of the routines, where, to save space, the word "generalized" is omitted.

2.4.9 Generalized (or Quotient) Singular Value Decomposition

The **generalized (or quotient) singular value decomposition** of an m-by-n matrix A and a p-by-n matrix B is described in section 2.3.5. The routines described in this section, are used to compute the decomposition. The computation proceeds in the following two stages:

Table 2.15: Computational routines for the generalized nonsymmetric eigenproblem

Type of matrix and storage scheme	Operation	Single precision		Double precision	
		real	complex	real	complex
general	Hessenberg reduction	SGGHRD	CGGHRD	DGGHRD	ZGGHRD
	balancing	SGGBAL	CGGBAL	DGGBAL	ZGGBAL
	back transforming	SGGBAK	CGGBAK	DGGBAK	ZGGBAK
Hessenberg	Schur factorization	SHGEQZ	CHGEQZ	DHGEQZ	ZHGEQZ
(quasi)triangular	eigenvectors	STGEVC	CTGEVC	DTGEVC	ZTGEVC
	reordering Schur decomposition	STGEXC	CTGEXC	DTGEXC	ZTGEXC
	Sylvester equation	STGSYL	CTGSYL	DTGSYL	ZTGSYL
	condition numbers of eigenvalues/vectors	STGSNA	CTGSNA	DTGSNA	ZTGSNA
	condition numbers of eigenvalue cluster/ deflating subspaces	STGSEN	CTGSEN	DTGSEN	ZTGSEN

1. xGGSVP is used to reduce the matrices A and B to triangular form:

$$U_1^T A Q_1 = \begin{array}{r} \\ k \\ l \\ m-k-l \end{array} \begin{array}{c} \begin{array}{ccc} n-k-l & k & l \end{array} \\ \begin{pmatrix} 0 & A_{12} & A_{13} \\ 0 & 0 & A_{23} \\ 0 & 0 & 0 \end{pmatrix} \end{array}$$

$$V_1^T B Q_1 = \begin{array}{r} \\ l \\ p-l \end{array} \begin{array}{c} \begin{array}{ccc} n-k-l & k & l \end{array} \\ \begin{pmatrix} 0 & 0 & B_{13} \\ 0 & 0 & 0 \end{pmatrix} \end{array}$$

where A_{12} and B_{13} are nonsingular upper triangular, and A_{23} is upper triangular. If $m-k-l < 0$, the bottom zero block of $U_1^T A Q_1$ does not appear, and A_{23} is upper trapezoidal. U_1, V_1 and Q_1 are orthogonal matrices (or unitary matrices if A and B are complex). l is the rank of B, and $k+l$ is the rank of $\begin{pmatrix} A \\ B \end{pmatrix}$.

2. The generalized singular value decomposition of two l-by-l upper triangular matrices A_{23} and B_{13} is computed using xTGSJA[2]:

$$A_{23} = U_2 C R Q_2^T \quad \text{and} \quad B_{13} = V_2 S R Q_2^T \ .$$

Here U_2, V_2 and Q_2 are orthogonal (or unitary) matrices, C and S are both real nonnegative diagonal matrices satisfying $C^2 + S^2 = I$, S is nonsingular, and R is upper triangular and nonsingular.

[2] If $m-k-l < 0$, we may add some zero rows to A_{23} to make it upper triangular.

Table 2.16: Computational routines for the generalized singular value decomposition

Operation	Single precision		Double precision	
	real	complex	real	complex
triangular reduction of A and B	SGGSVP	CGGSVP	DGGSVP	ZGGSVP
GSVD of a pair of triangular matrices	STGSJA	CTGSJA	DTGSJA	ZTGSJA

The reduction to triangular form, performed by xGGSVP, uses QR decomposition with column pivoting for numerical rank determination. See [8] for details.

The generalized singular value decomposition of two triangular matrices, performed by xTGSJA, is done using a Jacobi-like method as described in [83, 10].

Chapter 3

Performance of LAPACK

Note: this chapter presents some performance figures for LAPACK routines. The figures are provided for illustration only, and should not be regarded as a definitive up-to-date statement of performance. They have been selected from performance figures obtained in 1999 during the development of version 3.0 of LAPACK. All reported timings were obtained using the optimized version of the BLAS available on each machine. Performance is affected by many factors that may change from time to time, such as details of hardware (cycle time, cache size), compiler, and BLAS. We attempted to obtain reliable timings by making multiple runs, but we do not guarantee the reproducibility of these numbers. To obtain up-to-date performance figures, use the timing programs provided with LAPACK.

3.1 Factors that Affect Performance

Can we provide **portable** software for computations in dense linear algebra that is **efficient** on a wide range of modern high-performance computers? If so, how? Answering these questions — and providing the desired software — has been the goal of the LAPACK project.

LINPACK [38] and EISPACK [92, 54] have for many years provided high-quality portable software for linear algebra; but on modern high-performance computers they often achieve only a small fraction of the peak performance of the machines. Therefore, LAPACK has been designed to supersede LINPACK and EISPACK, principally by achieving much greater efficiency — but at the same time also adding extra functionality, using some new or improved algorithms, and integrating the two sets of algorithms into a single package.

LAPACK was originally targeted to achieve good performance on single-processor vector machines and on shared memory multiprocessor machines with a modest number of powerful processors. Since the start of the project, another class of machines has emerged for which LAPACK software is equally well-suited—the high-performance "super-scalar" workstations. (LAPACK is intended to be used across the whole spectrum of modern computers, but when considering performance, the emphasis is on machines at the more powerful end of the spectrum.)

Here we discuss the main factors that affect the performance of linear algebra software on these classes of machines.

3.1.1 Vectorization

Designing vectorizable algorithms in linear algebra is usually straightforward. Indeed, for many computations there are several variants, all vectorizable, but with different characteristics in performance (see, for example, [45]). Linear algebra algorithms can come close to the peak performance of many machines — principally because peak performance depends on some form of chaining of vector addition and multiplication operations, and this is just what the algorithms require.

However, when the algorithms are realized in straightforward Fortran 77 code, the performance may fall well short of the expected level, usually because vectorizing Fortran compilers fail to minimize the number of memory references — that is, the number of vector load and store operations. This brings us to the next factor.

3.1.2 Data Movement

What often limits the actual performance of a vector—or scalar— floating-point unit is the rate of transfer of data between different levels of memory in the machine. Examples include: the transfer of vector operands in and out of vector registers, the transfer of scalar operands in and out of a high-speed scalar processor, the movement of data between main memory and a high-speed cache or local memory, and paging between actual memory and disk storage in a virtual memory system.

It is desirable to maximize the ratio of floating-point operations to memory references, and to reuse data as much as possible while it is stored in the higher levels of the memory hierarchy (for example, vector registers or high-speed cache).

A Fortran programmer has no explicit control over these types of data movement, although one can often influence them by imposing a suitable structure on an algorithm.

3.1.3 Parallelism

The nested loop structure of most linear algebra algorithms offers considerable scope for loop-based parallelism on shared memory machines. This is the principal type of parallelism that LAPACK at present aims to exploit. It can sometimes be generated automatically by a compiler, but often requires the insertion of compiler directives.

3.2 The BLAS as the Key to Portability

How then can we hope to be able to achieve sufficient control over vectorization, data movement, and parallelism in portable Fortran code, to obtain the levels of performance that machines can offer?

The LAPACK strategy for combining efficiency with portability is to construct the software as much as possible out of calls to the BLAS (Basic Linear Algebra Subprograms); the BLAS are used as building blocks.

The efficiency of LAPACK software depends on efficient implementations of the BLAS being provided by computer vendors (or others) for their machines. Thus the BLAS form a low-level interface between LAPACK software and different machine architectures. Above this level, almost all of the LAPACK software is truly portable.

There are now three levels of BLAS:

Level 1 BLAS [78]: for vector operations, such as $y \leftarrow \alpha x + y$

Level 2 BLAS [42]: for matrix-vector operations, such as $y \leftarrow \alpha Ax + \beta y$

Level 3 BLAS [40]: 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 LAPACK, but 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 Y-MP, CRAY C90, or CONVEX C4 machine. However on other vector processors, such as a CRAY 2, or a RISC workstation or PC with one more levels of cache, their performance is limited by the rate of data movement between different levels of memory.

This limitation is overcome by the Level 3 BLAS, which perform $O(n^3)$ 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 BLAS also allow us to exploit parallelism in a way that is transparent to the software that calls them. Even the Level 2 BLAS offer some scope for exploiting parallelism, but greater scope is provided by the Level 3 BLAS, as Table 3.1 illustrates.

3.3 Block Algorithms and their Derivation

It is comparatively straightforward to recode many of the algorithms in LINPACK and EISPACK so that they call Level 2 BLAS. Indeed, in the simplest cases the same floating-point operations are performed, possibly even in the same order: it is just a matter of reorganizing the software. To illustrate this point we derive the Cholesky factorization algorithm that is used in the LINPACK routine SPOFA, which factorizes a symmetric positive definite matrix as $A = U^T U$. Writing these equations as:

$$\begin{pmatrix} A_{11} & a_j & A_{13} \\ \cdot & a_{jj} & \alpha_j^T \\ \cdot & \cdot & A_{33} \end{pmatrix} = \begin{pmatrix} U_{11}^T & 0 & 0 \\ u_j^T & u_{jj} & 0 \\ U_{13}^T & \mu_j & U_{33}^T \end{pmatrix} \begin{pmatrix} U_{11} & u_j & U_{13} \\ 0 & u_{jj} & \mu_j^T \\ 0 & 0 & U_{33} \end{pmatrix}$$

Table 3.1: Speed in megaflops of Level 2 and Level 3 BLAS operations on an SGI Origin 2000

(all matrices are of order 1000; U is upper triangular)

Number of processors:	1	2	4	8	16
Level 2: $y \leftarrow \alpha Ax + \beta y$	210	154	385	493	163
Level 3: $C \leftarrow \alpha AB + \beta C$	555	1021	1864	3758	4200
Level 2: $x \leftarrow Ux$	162	152	148	153	137
Level 3: $B \leftarrow UB$	518	910	1842	3519	5171
Level 2: $x \leftarrow U^{-1}x$	172	80	140	168	17
Level 3: $B \leftarrow U^{-1}B$	519	967	1859	3507	5078

and equating coefficients of the jth column, we obtain:

$$\begin{aligned} a_j &= U_{11}^T u_j \\ a_{jj} &= u_j^T u_j + u_{jj}^2. \end{aligned}$$

Hence, if U_{11} has already been computed, we can compute u_j and u_{jj} from the equations:

$$\begin{aligned} U_{11}^T u_j &= a_j \\ u_{jj}^2 &= a_{jj} - u_j^T u_j. \end{aligned}$$

Here is the body of the code of the LINPACK routine SPOFA, which implements the above method:

```
      DO 30 J = 1, N
         INFO = J
         S = 0.0E0
         JM1 = J - 1
         IF (JM1 .LT. 1) GO TO 20
         DO 10 K = 1, JM1
            T = A(K,J) - SDOT(K-1,A(1,K),1,A(1,J),1)
            T = T/A(K,K)
            A(K,J) = T
            S = S + T*T
   10    CONTINUE
   20    CONTINUE
         S = A(J,J) - S
C     ......EXIT
         IF (S .LE. 0.0E0) GO TO 40
         A(J,J) = SQRT(S)
   30 CONTINUE
```

And here is the same computation recoded in "LAPACK-style" to use the Level 2 BLAS routine STRSV (which solves a triangular system of equations). The call to STRSV has replaced the loop over K which made several calls to the Level 1 BLAS routine SDOT. (For reasons given below, this is not the actual code used in LAPACK — hence the term "LAPACK-style".)

```
      DO 10 J = 1, N
         CALL STRSV( 'Upper', 'Transpose', 'Non-unit', J-1, A, LDA,
     $               A(1,J), 1 )
         S = A(J,J) - SDOT( J-1, A(1,J), 1, A(1,J), 1 )
         IF( S.LE.ZERO ) GO TO 20
         A(J,J) = SQRT( S )
   10 CONTINUE
```

This change by itself is sufficient to make big gains in performance on machines like the CRAY C-90.

But on many machines such as an IBM RISC Sys/6000-550 (using double precision) there is virtually no difference in performance between the LINPACK-style and the LAPACK Level 2 BLAS style code. Both styles run at a megaflop rate far below its peak performance for matrix-matrix multiplication. To exploit the faster speed of Level 3 BLAS, the algorithms must undergo a deeper level of restructuring, and be re-cast as a **block algorithm** — that is, an algorithm that operates on **blocks** or submatrices of the original matrix.

To derive a block form of Cholesky factorization, we write the defining equation in partitioned form thus:

$$\begin{pmatrix} A_{11} & A_{12} & A_{13} \\ . & A_{22} & A_{23} \\ . & . & A_{33} \end{pmatrix} = \begin{pmatrix} U_{11}^T & 0 & 0 \\ U_{12}^T & U_{22}^T & 0 \\ U_{13}^T & U_{23}^T & U_{33}^T \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} & U_{13} \\ 0 & U_{22} & U_{23} \\ 0 & 0 & U_{33} \end{pmatrix}.$$

Equating submatrices in the second block of columns, we obtain:

$$\begin{aligned} A_{12} &= U_{11}^T U_{12} \\ A_{22} &= U_{12}^T U_{12} + U_{22}^T U_{22}. \end{aligned}$$

Hence, if U_{11} has already been computed, we can compute U_{12} as the solution to the equation

$$U_{11}^T U_{12} = A_{12}$$

by a call to the Level 3 BLAS routine STRSM; and then we can compute U_{22} from

$$U_{22}^T U_{22} = A_{22} - U_{12}^T U_{12}.$$

This involves first updating the symmetric submatrix A_{22} by a call to the Level 3 BLAS routine SSYRK, and then computing its Cholesky factorization. Since Fortran does not allow recursion, a separate routine must be called (using Level 2 BLAS rather than Level 3), named SPOTF2 in the code below. In this way successive blocks of columns of U are computed. Here is LAPACK-style code for the block algorithm. In this code-fragment NB denotes the width of the blocks.

```
      DO 10 J = 1, N, NB
         JB = MIN( NB, N-J+1 )
         CALL STRSM( 'Left', 'Upper', 'Transpose', 'Non-unit', J-1, JB,
     $               ONE, A, LDA, A( 1, J ), LDA )
         CALL SSYRK( 'Upper', 'Transpose', JB, J-1, -ONE, A( 1, J ), LDA,
     $               ONE, A( J, J ), LDA )
         CALL SPOTF2( 'Upper', JB, A( J, J ), LDA, INFO )
         IF( INFO.NE.0 ) GO TO 20
   10 CONTINUE
```

But that is not the end of the story, and the code given above is not the code that is actually used in the LAPACK routine SPOTRF. We mentioned in subsection 3.1.1 that for many linear algebra computations there are several vectorizable variants, often referred to as i-, j- and k-variants, according to a convention introduced in [45] and used in [55]. The same is true of the corresponding block algorithms.

It turns out that the j-variant that was chosen for LINPACK, and used in the above examples, is not the fastest on many machines, because it is based on solving triangular systems of equations, which can be significantly slower than matrix-matrix multiplication. The variant actually used in LAPACK is the i-variant, which does rely on matrix-matrix multiplication.

3.4 Examples of Block Algorithms in LAPACK

Having discussed in detail the derivation of one particular block algorithm, we now describe examples of the performance that has been achieved with a variety of block algorithms. Tables 3.2, 3.3, 3.4, 3.5, and 3.6 describe the hardware and software characteristics of the machines.

Table 3.2: Characteristics of the Compaq/Digital computers timed

	Dec Alpha Miata	Compaq AlphaServer DS-20
Model	LX164	DS-20 (21264)
Processor	EV56	EV6
Clock speed (MHz)	533	500
Processors per node	1	1
Operating system	Linux 2.2.7	OSF1 V4.0 1091
BLAS	ATLAS (version 1.0)	DXML (version 3.5)
Fortran compiler	g77 (egcs 2.91.60)	f77 (version 5.2)
Fortran flags	-funroll-all-loops -fno-f2c -O3	-O4 -fpe1
Precision	double (64-bit)	double (64-bit)

See Gallivan *et al.* [52] and Dongarra *et al.* [43] for an alternative survey of algorithms for dense linear algebra on high-performance computers.

Table 3.3: Characteristics of the IBM computers timed

	IBM Power 3	IBM PowerPC
Model	Winterhawk	
Processor	630	604e
Clock speed (MHz)	200	190
Processors per node	1	1
Operating system	AIX 4.3	Linux 2.2.7
BLAS	ESSL (3.1.1.0)	ATLAS (version 1.0)
Fortran compiler	xlf (6.1.0.0)	g77 (egcs 2.91.66)
Fortran flags	-O4 -qmaxmem=-1	-funroll-all-loops -fno-f2c -O3
Precision	double (64-bit)	double (64-bit)

Table 3.4: Characteristics of the Intel computers timed

	Intel Pentium II	Intel Pentium III
Model		
Processor	Pentium II	Pentium III
Clock speed (MHz)	450	550
Processors per node	1	1
Operating system	Linux 2.2.7	Linux 2.2.5-15
BLAS	ATLAS (version 1.0)	ATLAS (version 1.0)
Fortran compiler	g77 (egcs 2.91.60)	g77 (egcs 2.91.66)
Fortran flags	-funroll-all-loops -fno-f2c -O3	-funroll-all-loops -fno-f2c -O3
Precision	double (64-bit)	double (64-bit)

3.4.1 Factorizations for Solving Linear Equations

The well-known LU and Cholesky factorizations are the simplest block algorithms to derive. No extra floating-point operations nor extra working storage are required.

Table 3.7 illustrates the speed of the LAPACK routine for LU factorization of a real matrix, DGETRF in double precision. This corresponds to 64-bit floating-point arithmetic. A block size of 1 means that the unblocked algorithm is used, since it is faster than — or at least as fast as — a blocked algorithm. These numbers may be compared to those for DGEMM in Table 3.12, which should be upper bounds.

Table 3.8 gives similar results for Cholesky factorization.

LAPACK, like LINPACK, provides a factorization for symmetric indefinite matrices, so that A is factorized as $PUDU^TP^T$, where P is a permutation matrix, and D is block diagonal with blocks of order 1 or 2. A block form of this algorithm has been derived, and is implemented in the LAPACK routine SSYTRF/DSYTRF. It has to duplicate a little of the computation in order to "look ahead"

Table 3.5: Characteristics of the SGI computer timed

	SGI Origin 2000
Model	IP27
Processor	MIPS R12000
Clock speed (MHz)	300
Processors per node	64
Operating system	IRIX 6.5
BLAS	SGI BLAS
Fortran compiler	f77 (7.2.1.2m)
Fortran flags	-O3 -64 -mips4 -r10000 -OPT:IEEE_NaN_inf=ON
Precision	double (64-bit)

Table 3.6: Characteristics of the Sun computers timed

	Sun Ultra 2	Sun Enterprise 450
Model	Ultra 2 Model 2200	Model 1300
Processor	Sun UltraSPARC	Sun UltraSPARC-II
Clock speed (MHz)	200	300
Processors per node	1	1
Operating system	SunOS 5.5.1	SunOS 5.5.7
BLAS	Sun Performance Library	Sun Performance Library
Fortran compiler	f77 (SC5.0)	f77 (SC5.0)
Fortran flags	-f -dalign -native -xO5 -xarch=v8plusa	-f -dalign -native -xO5 -xarch=v8plusa
Precision	double (64-bit)	double (64-bit)

to determine the necessary row and column interchanges, but the extra work can be more than compensated for by the greater speed of updating the matrix by blocks as is illustrated in Table 3.9, provided that n is large enough.

LAPACK, like LINPACK, provides LU and Cholesky factorizations of band matrices. The LINPACK algorithms can easily be restructured to use Level 2 BLAS, though that has little effect on performance for matrices of very narrow bandwidth. It is also possible to use Level 3 BLAS, at the price of doing some extra work with zero elements outside the band [48]. This becomes worthwhile for matrices of large order and semi-bandwidth greater than 100 or so.

Table 3.7: Speed in megaflops of DGETRF for square matrices of order n

	No. of processors	Block size	Values of n	
			100	1000
Dec Alpha Miata	1	28	172	370
Compaq AlphaServer DS-20	1	28	353	440
IBM Power 3	1	32	278	551
IBM PowerPC	1	52	77	148
Intel Pentium II	1	40	132	250
Intel Pentium III	1	40	143	297
SGI Origin 2000	1	64	228	452
SGI Origin 2000	4	64	190	699
Sun Ultra 2	1	64	121	240
Sun Enterprise 450	1	64	163	334

Table 3.8: Speed in megaflops of DPOTRF for matrices of order n with UPLO = 'U'

	No. of processors	Block size	Values of n	
			100	1000
Dec Alpha Miata	1	28	197	399
Compaq AlphaServer DS-20	1	28	306	464
IBM Power 3	1	32	299	586
IBM PowerPC	1	52	79	125
Intel Pentium II	1	40	118	253
Intel Pentium III	1	40	142	306
SGI Origin 2000	1	64	222	520
SGI Origin 2000	4	64	137	1056
Sun Ultra 2	1	64	131	276
Sun Enterprise 450	1	64	178	391

Table 3.9: Speed in megaflops of DSYTRF for matrices of order n with UPLO = 'U' on an IBM Power 3

Block size	Values of n	
	100	1000
1	186	215
32	130	412

Table 3.10: Speed in megaflops of DGEQRF for square matrices of order n

	No. of processors	Block size	Values of n	
			100	1000
Dec Alpha Miata	1	28	141	363
Compaq AlphaServer DS-20	1	28	326	444
IBM Power 3	1	32	244	559
IBM PowerPC	1	52	45	127
Intel Pentium II	1	40	113	250
Intel Pentium III	1	40	135	297
SGI Origin 2000	1	64	173	451
SGI Origin 2000	4	64	55	766
Sun Ultra 2	1	64	20	230
Sun Enterprise 450	1	64	48	329

3.4.2 QR Factorization

The traditional algorithm for QR factorization is based on the use of elementary Householder matrices of the general form

$$H = I - \tau v v^T$$

where v is a column vector and τ is a scalar. This leads to an algorithm with very good vector performance, especially if coded to use Level 2 BLAS.

The key to developing a block form of this algorithm is to represent a product of b elementary Householder matrices of order n as a block form of a Householder matrix. This can be done in various ways. LAPACK uses the following form [90]:

$$H_1 H_2 \ldots H_b = I - VTV^T$$

where V is an n-by-b matrix whose columns are the individual vectors $v_1, v_2, \ldots, v_b$ associated with the Householder matrices $H_1, H_2, \ldots, H_b$, and T is an upper triangular matrix of order b. Extra work is required to compute the elements of T, but once again this is compensated for by the greater speed of applying the block form. Table 3.10 summarizes results obtained with the LAPACK routine DGEQRF.

3.4.3 Eigenvalue Problems

Eigenvalue problems have also provided a fertile ground for the development of higher performance algorithms. These algorithms generally all consist of three phases: (1) reduction of the original dense matrix to a condensed form by orthogonal transformations, (2) solution of condensed form, and (3) optional backtransformation of the solution of the condensed form to the solution of the original matrix. In addition to block versions of algorithms for phases 1 and 3, a number of entirely new algorithms for phase 2 have recently been discovered. In particular, Version 3.0 of LAPACK

includes new block algorithms for the singular value decomposition (SVD) and SVD-based least squares solver, as well as a prototype of a new algorithm xSTEGR[35, 87, 86, 36], which may be the ultimate solution for the symmetric eigenproblem and SVD on both parallel and serial machines.

The first step in solving many types of eigenvalue problems is to reduce the original matrix to a condensed form by orthogonal transformations. In the reduction to condensed forms, the unblocked algorithms all use elementary Householder matrices and have good vector performance. Block forms of these algorithms have been developed [46], but all require additional operations, and a significant proportion of the work must still be performed by Level 2 BLAS, so there is less possibility of compensating for the extra operations.

The algorithms concerned are:

- reduction of a symmetric matrix to tridiagonal form to solve a symmetric eigenvalue problem: LAPACK routine xSYTRD applies a symmetric block update of the form

  $$A \leftarrow A - UX^T - XU^T$$

  using the Level 3 BLAS routine xSYR2K; Level 3 BLAS account for at most half the work.

- reduction of a rectangular matrix to bidiagonal form to compute a singular value decomposition: LAPACK routine xGEBRD applies a block update of the form

  $$A \leftarrow A - UX^T - YV^T$$

  using two calls to the Level 3 BLAS routine xGEMM; Level 3 BLAS account for at most half the work.

- reduction of a nonsymmetric matrix to Hessenberg form to solve a nonsymmetric eigenvalue problem: LAPACK routine xGEHRD applies a block update of the form

  $$A \leftarrow (I - VT^TV^T)(A - XV^T).$$

  Level 3 BLAS account for at most three-quarters of the work.

Note that only in the reduction to Hessenberg form is it possible to use the block Householder representation described in subsection 3.4.2. Extra work must be performed to compute the n-by-b matrices X and Y that are required for the block updates (b is the block size) — and extra workspace is needed to store them.

Nevertheless, the performance gains can be worthwhile on some machines for large enough matrices, for example, on an IBM Power 3, as shown in Table 3.11.

Following the reduction of a dense (or band) symmetric matrix to tridiagonal form T, we must compute the eigenvalues and (optionally) eigenvectors of T. Computing the eigenvalues of T alone (using LAPACK routine xSTERF) requires $O(n^2)$ flops, whereas the reduction routine xSYTRD does $\frac{4}{3}n^3 + O(n^2)$ flops. So eventually the cost of finding eigenvalues alone becomes small compared to the cost of reduction. However, xSTERF does only scalar floating point operations, without scope for the BLAS, so n may have to be large before xSYTRD is slower than xSTERF.

Table 3.11: Speed in megaflops of reductions to condensed forms on an IBM Power 3

(all matrices are square of order n)

	Block size	Values of n 100	1000
DSYTRD	1	356	418
	32	192	499
DGEBRD	1	269	241
	32	179	342
DGEHRD	1	277	250
	32	277	471

Version 2.0 of LAPACK introduced a new algorithm, xSTEDC, for finding all eigenvalues and eigenvectors of T. The new algorithm can exploit Level 2 and 3 BLAS, whereas the previous algorithm, xSTEQR, could not. Furthermore, xSTEDC usually does many fewer flops than xSTEQR, so the speedup is compounded. Briefly, xSTEDC works as follows (for details, see [57, 89]). The tridiagonal matrix T is written as

$$T = \begin{pmatrix} T_1 & 0 \\ 0 & T_2 \end{pmatrix} + H$$

where T_1 and T_2 are tridiagonal, and H is a very simple rank-one matrix. Then the eigenvalues and eigenvectors of T_1 and T_2 are found by applying the algorithm recursively; this yields $T_1 = Q_1\Lambda_1 Q_1^T$ and $T_2 = Q_2\Lambda_2 Q_2^T$, where Λ_i is a diagonal matrix of eigenvalues, and the columns of Q_i are orthonormal eigenvectors. Thus

$$T = \begin{pmatrix} Q_1\Lambda_1 Q_1^T & 0 \\ 0 & Q_2\Lambda_2 Q_2^T \end{pmatrix} + H = \begin{pmatrix} Q_1 & 0 \\ 0 & Q_2 \end{pmatrix} \cdot \left(\begin{pmatrix} \Lambda_1 & 0 \\ 0 & \Lambda_2 \end{pmatrix} + H' \right) \cdot \begin{pmatrix} Q_1^T & 0 \\ 0 & Q_2^T \end{pmatrix}$$

where H' is again a simple rank-one matrix. The eigenvalues and eigenvectors of $\begin{pmatrix} \Lambda_1 & 0 \\ 0 & \Lambda_2 \end{pmatrix} + H'$ may be found using $O(n^2)$ scalar operations, yielding $\begin{pmatrix} \Lambda_1 & 0 \\ 0 & \Lambda_2 \end{pmatrix} + H' = \hat{Q}\Lambda\hat{Q}^T$. Substituting this into the last displayed expression yields

$$T = \begin{pmatrix} Q_1 & 0 \\ 0 & Q_2 \end{pmatrix} \left(\hat{Q}\Lambda\hat{Q}^T\right) \begin{pmatrix} Q_1^T & 0 \\ 0 & Q_2^T \end{pmatrix} = \left(\begin{pmatrix} Q_1 & 0 \\ 0 & Q_2 \end{pmatrix} \hat{Q} \right) \Lambda \left(\hat{Q}^T \begin{pmatrix} Q_1^T & 0 \\ 0 & Q_2^T \end{pmatrix} \right) = Q\Lambda Q^T ,$$

where the diagonals of Λ are the desired eigenvalues of T, and the columns of $Q = \begin{pmatrix} Q_1 & 0 \\ 0 & Q_2 \end{pmatrix} \hat{Q}$ are the eigenvectors. Almost all the work is done in the two matrix multiplies of Q_1 and Q_2 times $\hat{Q}$, which is done using the Level 3 BLAS.

The same recursive algorithm has been developed for the singular value decomposition of the bidiagonal matrix resulting from reducing a dense matrix with xGEBRD. The SVD driver using this algorithm is called xGESDD. This recursive algorithm is also used for the SVD-based linear least squares solver xGELSD; see Figure 3.3 to see how much faster DGELSD is than its older routine DGELSS. SBDSQR, Comparison timings of DGESVD and DGESDD can be found in Tables 3.18 and 3.19.

Version 3.0 of LAPACK introduced another new algorithm, xSTEGR, for finding all the eigenvalues and eigenvectors of a symmetric tridiagonal matrix. It is usually even faster than xSTEDC above, and we expect it to ultimately replace all other LAPACK algorithm for the symmetric eigenvalue problem and SVD. Here is a rough description of how it works; for details see [35, 87, 86, 36].

It is easiest to think of xSTEGR as a variation on xSTEIN, inverse iteration. If all the eigenvalues were well separated, xSTEIN would run in $O(n^2)$ time. But it is difficult for xSTEIN to compute accurate eigenvectors belonging to close eigenvalues, those that have four or more decimals in common with their neighbors. Indeed, xSTEIN slows down because it reorthogonalizes the corresponding eigenvectors. xSTEGR escapes this difficulty by exploiting the invariance of eigenvectors under translation.

For each cluster $\mathcal{C}$ of close eigenvalues the algorithm chooses a shift s at one end of $\mathcal{C}$, or just outside, with the property that $T - sI$ permits triangular factorization $LDL^T = T - sI$ such that the small shifted eigenvalues ($\lambda - s$ for $\lambda \in \mathcal{C}$) are determined to high relative accuracy by the entries in L and D. Note that the small shifted eigenvalues will have fewer digits in common than those in $\mathcal{C}$. The algorithm computes these small shifted eigenvalues to high relative accuracy, either by xLASQ2 or by refining earlier approximations using bisection. This means that each computed $\hat{\lambda}$ approximating a $\lambda - s$ has error bounded by $O(\varepsilon)|\hat{\lambda}|$.

The next task is to compute an eigenvector for $\lambda - s$. For each $\hat{\lambda}$ the algorithm computes, with care, an optimal *twisted factorization*

$$\begin{aligned} LDL^T - \hat{\lambda}I &= N_r \Delta_r N_r^T \\ \Delta_r &= \mathrm{diag}(\delta_1, \delta_2, ..., \delta_n) \end{aligned}$$

obtained by implementing triangular factorization both from top down and bottom up and joining them at a well chosen index r. An approximate eigenvector z is obtained by solving $N_r^T z = e_r$ where e_r is column r of I. It turns out that $N_r \Delta_r N_r^T = e_r \delta_r$ and

$$||LDL^T - \hat{\lambda}I|| = |\text{error in } \hat{\lambda}|/||u||_\infty + ...$$

where ... indicates smaller terms and $Tu = \lambda u$, $u^T u = 1$. ¿From the basic gap theorem [85]

$$|\sin\theta(u, z)| <= O(\epsilon))|\hat{\lambda}|/(||u||_\infty |\hat{\lambda} - \hat{\mu}|) + ...$$

where $\hat{\mu}$ is $\hat{\lambda}$'s neighbor in the shifted spectrum. Given this nice bound the algorithm computes z for all λ such that the relative gap $|\lambda - \mu|/|\lambda - s| > 10^{-3}$. These z's have an error that is $O(\epsilon)$.

The procedure described above is repeated for any eigenvalues that remain without eigenvectors. It can be shown that all the computed z's are very close to eigenvectors of small relative perturbations of one global Cholesky factorization GG^T of a translate $T - \sigma I$ of T. A key component of the

algorithm is the use of recently discovered differential qd algorithms to ensure that the twisted factorizations described above are computed without ever forming the indicated matrix products such as LDL^T [51].

For computing the eigenvalues and eigenvectors of a Hessenberg matrix—or rather for computing its Schur factorization— yet another flavor of block algorithm has been developed: a **multishift** QR iteration [9]. Whereas the traditional EISPACK routine HQR uses a double shift (and the corresponding complex routine COMQR uses a single shift), the multishift algorithm uses block shifts of higher order. It has been found that often the total number of operations *decreases* as the order of shift is increased until a minimum is reached typically between 4 and 8; for higher orders the number of operations increases quite rapidly. On many machines the speed of applying the shift increases steadily with the order, and the optimum order of shift is typically in the range 8–16. Note however that the performance can be very sensitive to the choice of the order of shift; it also depends on the numerical properties of the matrix. Dubrulle [49] has studied the practical performance of the algorithm, while Watkins and Elsner [101] discuss its theoretical asymptotic convergence rate.

Finally, we note that research into block algorithms for symmetric and nonsymmetric eigenproblems continues [11, 68], and future versions of LAPACK will be updated to contain the best algorithms available.

3.5 LAPACK Benchmark

This section contains performance numbers for selected LAPACK driver routines. These routines provide complete solutions for the most common problems of numerical linear algebra, and are the routines users are most likely to call:

- Solve an n-by-n system of linear equations with 1 right hand side using DGESV. side
- Find only the eigenvalues of an n-by-n nonsymmetric matrix using DGEEV.
- Find the eigenvalues and right eigenvectors of an n-by-n nonsymmetric matrix using DGEEV. DSYEVD. matrix using DSYEVD.
- Find only the singular values of an n-by-n matrix using DGESDD.
- Find the singular values and right and left singular vectors of an n-by-n matrix using DGESVD.
- Find the singular values and right and left singular vectors of an n-by-n matrix using DGESDD.

We only present data on DGESDD for singular values only, and not DGESVD, because both use the same algorithm. We include both DGESVD and DGESDD for computing all the singular values and singular vectors to illustrate the speedup of the new algorithm DGESDD over its predecessor

DGESVD: For 1000-by-1000 matrices DGESDD is between 6 and 7 times faster than DGESVD on most machines.

The above drivers are timed on a variety of computers. In addition, we present data on fewer machines to compare the performance of the five different routines for solving linear least squares problems, and several different routines for the symmetric eigenvalue problem. Again, the purpose is to illustrate the performance improvements in LAPACK 3.0.

Data is provided for PCs, shared memory parallel computers, and high performance workstations. All timings were obtained by using the machine-specific optimized BLAS available on each machine. For machines running the Linux operating system, the ATLAS[102] BLAS were used. In all cases the data consisted of 64-bit floating point numbers (double precision). For each machine and each driver, a small problem ($N = 100$ with $LDA = 101$) and a large problem ($N = 1000$ with $LDA = 1001$) were run. Block sizes NB = 1, 16, 32 and 64 were tried, with data only for the fastest run reported in the tables below. For DGEEV, $ILO = 1$ and $IHI = N$. The test matrices were generated with randomly distributed entries. All run times are reported in seconds, and block size is denoted by nb. The value of nb was chosen to make $N = 1000$ optimal. It is not necessarily the best choice for $N = 100$. See Section 6.2 for details.

The performance data is reported using three or four statistics. First, the run-time in seconds is given. The second statistic measures how well our performance compares to the speed of the BLAS, specifically DGEMM. This "equivalent matrix multiplies" statistic is calculated as

$$\frac{\text{run-time(LAPACK Driver on an } n\text{-by-}n \text{ matrix)}}{\text{run-time(DGEMM used to multiply two } n\text{-by-}n \text{ matrices)}}$$

and labeled as $\frac{\text{Time}}{\text{T(MM)}}$ in the tables. The performance information for the BLAS routines DGEMV (TRANS='N') and DGEMM (TRANSA='N', TRANSB='N') is provided in Table 3.12, along with the clock speed for each machine in Tables 3.2, 3.3, 3.4, 3.5, and 3.6. The third statistic is the true megaflop rating. For the eigenvalue and singular value drivers, a fourth "synthetic megaflop" statistic is also presented. We provide this statistic because the number of floating point operations needed to find eigenvalues and singular values depends on the input data, unlike linear equation solving or linear least squares solving with DGELS, and on the algorithm. The synthetic megaflop rating is defined to be a "standard" number of flops required to solve the problem, divided by the run-time in microseconds. This "standard" number of flops is taken to be the average for a standard algorithm over a variety of problems, as given in Table 3.13. In all cases we ignore terms of $O(N^2)$. The flop count in this table for the nonsymmetric eigenvalue problem assumes the unblocked algorithm is used, and that two QR sweeps are needed to deflate each eigenvalue [55]. The flop count for the symmetric eigenproblem and SVD assumes that all the work done on the tridiagonal and bidiagonal matrices is $O(N^2)$, as is the case with xSTEGR and will be the case with the SVD in the future.

We also include several figures comparing the speed of several routines for the symmetric eigenvalue problem and several least squares drivers to highlight the performance improvements in LAPACK 3.0.

First consider Figure 3.1, which compares the performance of three routines, DSTEQR, DSTEDC and DSTEGR, for computing all the eigenvalues and eigenvectors of a symmetric tridiagonal ma-

trix. The times are shown on a Compaq AlphaServer DS-20 for matrix dimensions from 100 to 1000. The symmetric tridiagonal matrix was obtained by taking a random dense symmetric matrix and reducing it to tridiagonal form (the performance can vary depending on the distribution of the eigenvalues of the matrix, but the data shown here is typical). DSTEQR (used in driver DSYEV) was the only algorithm available in LAPACK 1.0, DSTEDC (used in driver DSYEVD) was introduced in LAPACK 2.0, and DSTEGR (used in driver DSYEVR) was introduced in LAPACK 3.0. As can be seen, for large matrices DSTEGR is about 14 times faster than DSTEDC and nearly 50 times faster than DSTEQR.

Next consider Figure 3.2, which compares the performance of four driver routines, DSYEV, DSYEVX, DSYEVD and DSYEVR, for computing all the eigenvalues and eigenvectors of a dense symmetric matrix. The times are shown on an IBM Power 3 for matrix dimensions from 100 to 2000. The symmetric matrix was chosen randomly. The cost of these drivers is essentially the cost of phases 1 and 3 (reduction to tridiagonal form and backtransformation) plus the cost of phase 2 (the symmetric tridiagonal eigenproblem) discussed in the last paragraph. Since the cost of phases 1 and 3 is large, performance differences in phase 2 are no longer as visible. We note that if we had chosen a test matrix with a large cluster of nearby eigenvalues, then the cost of DSYEVX would have been much larger, without significantly affecting the timings of the other drivers. DSYEVR is the driver of choice.

Finally consider Figure 3.3, which compares the performance of five drivers for the linear least squares problem, DGELS, DGELSY, DGELSX, DGELSD and DGELSS, which are shown in order of decreasing speed. DGELS is the fastest. DGELSY and DGELSX use QR with pivoting, and so handle rank-deficient problems more reliably than DGELS but can be more expensive. DGELSD and DGELSS use the SVD, and so are the most reliable (and expensive) ways to solve rank deficient least squares problems. DGELS, DGELSX and DGELSS were in LAPACK 1.0, and DGELSY and DGELSD were introduced in LAPACK 3.0. The times are shown on a Compaq AlphaServer DS-20 for squares matrices with dimensions from 100 to 1000, and for one right-hand-side. The matrices were chosen at random (which means they are full rank). First consider DGELSY, which is meant to replace DGELSX. We can see that the speed of DGELSY is nearly indistinguishable from the fastest routine DGELS, whereas DGELSX is over 2.5 times slower for large matrices. Next consider DGELSD, which is meant to replace DGELSS. It is 3 to 5 times slower than the fastest routine, DGELS, whereas its predecessor DGELSS was 7 to 34 times slower. Thus both DGELSD and DGELSY are significantly faster than their predecessors.

megaflop SSYEV/DSYEV.

Figure 3.1: Timings of routines for computing all eigenvalues and eigenvectors of a symmetric tridiagonal matrix. The upper graph shows times in seconds on a Compaq AlphaServer DS-20. The lower graph shows times relative to the fastest routine DSTEGR, which appears as a horizontal line at 1.

Figure 3.2: Timings of driver routines for computing all eigenvalues and eigenvectors of a dense symmetric matrix. The upper graph shows times in seconds on an IBM Power3. The lower graph shows times relative to the fastest routine DSYEVR, which appears as a horizontal line at 1.

Figure 3.3: Timings of driver routines for the least squares problem. The upper graph shows times in seconds on a Compaq AlphaServer DS-20. The lower graph shows times relative to the fastest routine DGELS, which appears as a horizontal line at 1.

Table 3.12: Execution time and Megaflop rates for DGEMV and DGEMM

	DGEMV				DGEMM			
	Values of $n = m = k$							
	100		1000		100		1000	
	Time	Mflops	Time	Mflops	Time	Mflops	Time	Mflops
Dec Alpha Miata	.0151	66	27.778	36	.0018	543	1.712	584
Compaq AlphaServer DS-20	.0027	376	8.929	112	.0019	522	2.000	500
IBM Power 3	.0032	304	2.857	350	.0018	567	1.385	722
IBM PowerPC	.0435	23	40.000	25	.0063	160	4.717	212
Intel Pentium II	.0075	134	16.969	59	.0031	320	3.003	333
Intel Pentium III	.0071	141	14.925	67	.0030	333	2.500	400
SGI O2K (1 proc)	.0046	216	4.762	210	.0018	563	1.801	555
SGI O2K (4 proc)	5.000	0.2	2.375	421	.0250	40	0.517	1936
Sun Ultra 2 (1 proc)	.0081	124	17.544	57	.0033	302	3.484	287
Sun Enterprise 450 (1 proc)	.0037	267	11.628	86	.0021	474	1.898	527

Table 3.13: "Standard" floating point operation counts for LAPACK drivers for n-by-n matrices

Driver	Options	Operation Count
xGESV	1 right hand side	$.67 \cdot N^3$
xGEEV	eigenvalues only	$10.00 \cdot N^3$
xGEEV	eigenvalues and right eigenvectors	$26.33 \cdot N^3$
xGES{VD,DD}	singular values only	$2.67 \cdot N^3$
xGES{VD,DD}	singular values and left and right singular vectors	$6.67 \cdot N^3$

Table 3.14: Performance of DGESV for n-by-n matrices

	No. of proc.	nb	Values of n					
			100			1000		
			Time	$\frac{\text{Time}}{T(MM)}$	Mflops	Time	$\frac{\text{Time}}{T(MM)}$	Mflops
Dec Alpha Miata	1	28	.004	2.2	164	1.903	1.11	351
Compaq AlphaServer DS-20	1	28	.002	1.05	349	1.510	0.76	443
IBM Power 3	1	32	.003	1.67	245	1.210	0.87	552
Intel Pentium II	1	40	.006	1.94	123	2.730	0.91	245
Intel Pentium III	1	40	.005	1.67	136	2.270	0.91	294
SGI Origin 2000	1	64	.003	1.67	227	1.454	0.81	460
SGI Origin 2000	4	64	.004	0.16	178	1.204	2.33	555
Sun Ultra 2	1	64	.008	2.42	81	5.460	1.57	122
Sun Enterprise 450	1	64	.006	2.86	114	3.698	1.95	181

Table 3.15: Performance of DGEEV, eigenvalues only

	No. of proc.	nb	Values of n							
			100				1000			
			Time	$\frac{\text{Time}}{\text{T(MM)}}$	True Mflops	Synth Mflops	Time	$\frac{\text{Time}}{\text{T(MM)}}$	True Mflops	Synth Mflops
Dec Alpha Miata	1	28	.157	87.22	70	64	116.480	68.04	81	86
Compaq AS DS-20	1	28	.044	23.16	423	228	52.932	26.47	177	189
IBM Power 3	1	32	.060	33.33	183	167	91.210	65.86	103	110
Intel Pentium II	1	40	.100	32.26	110	100	107.940	35.94	87	93
Intel Pentium III	1	40	.080	26.67	137	133	91.230	36.49	103	110
SGI Origin 2000	1	64	.074	41.11	148	135	54.852	30.46	172	182
SGI Origin 2000	4	64	.093	3.72	117	107	42.627	82.45	222	235
Sun Ultra 2	1	64	.258	78.18	43	38	246.151	70.65	38	41
Sun Enterprise 450	1	64	.178	84.76	62	56	163.141	85.95	57	61

Table 3.16: Performance of DGEEV, eigenvalues and right eigenvectors

	No. of proc.	nb	Values of n							
			100				1000			
			Time	$\frac{\text{Time}}{\text{T(MM)}}$	True Mflops	Synth Mflops	Time	$\frac{\text{Time}}{\text{T(MM)}}$	True Mflops	Synth Mflops
Dec Alpha Miata	1	28	.308	171.11	86	86	325.650	190.22	73	81
Compaq AS DS-20	1	28	.092	48.42	290	287	159.409	79.70	149	165
IBM Power 3	1	32	.130	72.22	204	203	230.650	166.53	103	114
Intel Pentium II	1	40	.200	64.52	133	132	284.020	94.58	84	93
Intel Pentium III	1	40	.170	56.67	156	155	239.070	95.63	100	110
SGI Origin 2000	1	64	.117	65.00	228	226	197.455	109.64	121	133
SGI Origin 2000	4	64	.159	6.36	167	166	146.975	284.28	164	179
Sun Ultra 2	1	64	.460	139.39	58	58	601.732	172.71	39	44
Sun Enterprise 450	1	64	.311	148.10	85	85	418.011	220.24	57	63

Table 3.17: Performance of DGESDD, singular values only

	No. of proc.	nb	Values of n							
			100				1000			
			Time	$\frac{\text{Time}}{\text{T(MM)}}$	True Mflops	Synth Mflops	Time	$\frac{\text{Time}}{\text{T(MM)}}$	True Mflops	Synth Mflops
Dec Alpha Miata	1	28	.043	23.89	61	61	36.581	21.37	73	73
Compaq AS DS-20	1	28	.011	5.79	236	236	11.789	5.89	226	226
IBM Power 3	1	32	.020	11.11	133	133	8.090	5.84	330	330
Intel Pentium II	1	40	.040	12.90	67	67	29.120	9.70	92	92
Intel Pentium III	1	40	.030	10.00	89	89	25.830	10.33	103	103
SGI Origin 2000	1	64	.024	13.33	113	113	12.407	6.89	215	215
SGI Origin 2000	4	64	.058	2.32	46	46	4.926	9.53	541	541
Sun Ultra 2	1	64	.088	26.67	30	30	60.478	17.36	44	44
Sun Enterprise 450	1	64	.060	28.57	92	45	47.813	25.19	56	56

Table 3.18: Performance of DGESVD, singular values and left and right singular vectors

	No. of proc.	nb	Values of n							
			100				1000			
			Time	$\frac{\text{Time}}{\text{T(MM)}}$	True Mflops	Synth Mflops	Time	$\frac{\text{Time}}{\text{T(MM)}}$	True Mflops	Synth Mflops
Dec Alpha Miata	1	28	.222	123.33	77	30	320.985	187.49	48	21
Compaq AS DS-20	1	28	.053	27.89	326	126	142.843	71.42	107	47
IBM Power 3	1	32	.070	38.89	245	95	251.940	181.91	61	26
Intel Pentium II	1	40	.150	48.39	114	44	282.550	94.09	54	24
Intel Pentium III	1	40	.120	40.00	142	56	244.690	97.88	62	27
SGI Origin 2000	1	64	.074	41.11	232	90	176.134	97.80	87	38
SGI Origin 2000	4	64	.145	5.80	118	46	198.656	384.25	77	34
Sun Ultra 2	1	64	.277	83.94	62	24	570.290	163.69	27	12
Sun Enterprise 450	1	64	.181	86.19	95	37	402.456	212.04	38	17

Table 3.19: Performance of DGESDD, singular values and left and right singular vectors

	No. of proc.	nb	Values of n							
			100				1000			
			Time	$\frac{\text{Time}}{\text{T(MM)}}$	True Mflops	Synth Mflops	Time	$\frac{\text{Time}}{\text{T(MM)}}$	True Mflops	Synth Mflops
Dec Alpha Miata	1	28	.055	30.56	123	121	47.206	27.57	141	141
Compaq AS DS-20	1	28	.021	11.05	310	318	20.658	10.33	323	323
IBM Power 3	1	32	.025	13.89	268	267	15.230	11.00	438	438
Intel Pentium II	1	40	.060	19.35	112	111	44.270	14.74	151	151
Intel Pentium III	1	40	.050	16.67	134	133	38.930	15.57	171	171
SGI Origin 2000	1	64	.035	19.44	189	191	24.985	13.87	267	267
SGI Origin 2000	4	64	.091	3.64	73	73	8.779	16.89	759	760
Sun Ultra 2	1	64	.149	45.15	45	45	93.417	26.81	72	71
Sun Enterprise 450	1	64	.102	48.57	66	65	70.597	37.20	94	94

Chapter 4

Accuracy and Stability

In addition to providing faster routines than previously available, LAPACK provides more comprehensive and better error bounds. Our goal is to provide error bounds for most quantities computed by LAPACK.

In this chapter we explain our overall approach to obtaining error bounds, and provide enough information to use the software. The comments at the beginning of the individual routines should be consulted for more details. It is beyond the scope of this chapter to justify all the bounds we present. Instead, we give references to the literature. For example, standard material on error analysis can be found in [25, 55, 67, 95].

In order to make this chapter easy to read, we have labeled sections not essential for a first reading as **Further Details**. The sections not labeled as **Further Details** should provide all the information needed to understand and use the main error bounds computed by LAPACK. The **Further Details** sections provide mathematical background, references, and tighter but more expensive error bounds, and may be read later.

In section 4.1 we discuss the sources of numerical error, in particular roundoff error. Section 4.2 discusses how to measure errors, as well as some standard notation. Section 4.3 discusses further details of how error bounds are derived. Sections 4.4 through 4.12 present error bounds for linear equations, linear least squares problems, generalized linear least squares problems, the symmetric eigenproblem, the nonsymmetric eigenproblem, the singular value decomposition, the generalized symmetric definite eigenproblem, the generalized nonsymmetric eigenproblem and the generalized (or quotient) singular value decomposition respectively. Section 4.13 discusses the impact of fast Level 3 BLAS on the accuracy of LAPACK routines.

4.1 Sources of Error in Numerical Calculations

There are two sources of error whose effects can be measured by the bounds in this chapter: *roundoff error* and *input error*. Roundoff error arises from rounding results of floating-point operations during the algorithm. Input error is error in the input to the algorithm from prior calculations or

measurements. We describe roundoff error first, and then input error.

Almost all the error bounds LAPACK provides are multiples of *machine epsilon*, which we abbreviate by ϵ. Machine epsilon bounds the roundoff in individual floating-point operations. It may be loosely defined as the largest relative error in any floating-point operation that neither overflows nor underflows. (Overflow means the result is too large to represent accurately, and underflow means the result is too small to represent accurately.) Machine epsilon is available either by the function call SLAMCH('Epsilon') (or simply SLAMCH('E')) in single precision, or by the function call DLAMCH('Epsilon') (or DLAMCH('E')) in double precision. See section 4.1.1 and Table 4.1 for a discussion of common values of machine epsilon.

Since underflow is almost always less significant than roundoff, we will not consider it further. Overflow usually means the computation is invalid, but there are some LAPACK routines that routinely generate and handle overflows using the rules of IEEE arithmetic (see section 4.1.1).

Bounds on *input errors*, or errors in the input parameters inherited from prior computations or measurements, may be easily incorporated into most LAPACK error bounds. Suppose the input data is accurate to, say, 5 decimal digits (we discuss exactly what this means in section 4.2). Then one simply replaces ϵ by $\max(\epsilon, 10^{-5})$ in the error bounds.

4.1.1 Further Details: Floating Point Arithmetic

Roundoff error is bounded in terms of the *machine precision* ϵ, which is the smallest value satisfying

$$|fl(a \oplus b) - (a \oplus b)| \leq \epsilon \cdot |a \oplus b| \ ,$$

where a and b are floating-point numbers, $\oplus$ is any one of the four operations $+$, $-$, $\times$ and $\div$, and $fl(a \oplus b)$ is the floating-point result of $a \oplus b$. Machine epsilon, ϵ, is the smallest value for which this inequality is true for all $\oplus$, and for all a and b such that $a \oplus b$ is neither too large (magnitude exceeds the overflow threshold) nor too small (is nonzero with magnitude less than the underflow threshold) to be represented accurately in the machine. We also assume ϵ bounds the relative error in unary operations like square root:

$$|fl(\sqrt{a}) - (\sqrt{a})| \leq \epsilon \cdot |\sqrt{a}| \ .$$

A precise characterization of ϵ depends on the details of the machine arithmetic and sometimes even of the compiler. For example, if addition and subtraction are implemented without a guard digit[1] we must redefine ϵ to be the smallest number such that

$$|fl(a \pm b) - (a \pm b)| \leq \epsilon \cdot (|a| + |b|).$$

In order to assure portability, machine parameters such as machine epsilon, the overflow threshold and underflow threshold are computed at runtime by the auxiliary routine xLAMCH[2]. The alternative, keeping a fixed table of machine parameter values, would degrade portability because the table would have to be changed when moving from one machine, or even one compiler, to another.

[1]This is the case on Cybers, Cray X-MP, Cray Y-MP, Cray 2 and Cray C90.

[2]See subsection 2.2.3 for explanation of the naming convention used for LAPACK routines.

Actually, most machines, but not yet all, do have the same machine parameters because they implement IEEE Standard Floating Point Arithmetic [4, 5], which exactly specifies floating-point number representations and operations. For these machines, including all modern workstations and PCs[3], the values of these parameters are given in Table 4.1.

Table 4.1: Values of Machine Parameters in IEEE Floating Point Arithmetic

Machine parameter	Single Precision (32 bits)	Double Precision (64 bits)
Machine epsilon ϵ = xLAMCH('E')	$2^{-24} \approx 5.96 \cdot 10^{-8}$	$2^{-53} \approx 1.11 \cdot 10^{-16}$
Underflow threshold = xLAMCH('U')	$2^{-126} \approx 1.18 \cdot 10^{-38}$	$2^{-1022} \approx 2.23 \cdot 10^{-308}$
Overflow threshold = xLAMCH('O')	$2^{128}(1-\epsilon) \approx 3.40 \cdot 10^{38}$	$2^{1024}(1-\epsilon) \approx 1.79 \cdot 10^{308}$

As stated above, we will ignore overflow and underflow in discussing error bounds. References [24, 67] discuss extending error bounds to include underflow, and show that for many common computations, when underflow occurs it is less significant than roundoff. With some important exceptions described below, overflow usually means that a computation has failed so the error bounds do not apply.

Therefore, most of our error bounds will simply be proportional to machine epsilon. This means, for example, that if the same problem in solved in double precision and single precision, the error bound in double precision will be smaller than the error bound in single precision by a factor of $\epsilon_{\text{double}}/\epsilon_{\text{single}}$. In IEEE arithmetic, this ratio is $2^{-53}/2^{-24} \approx 10^{-9}$, meaning that one expects the double precision answer to have approximately nine more decimal digits correct than the single precision answer.

LAPACK routines are generally insensitive to the details of rounding and exception handling, like their counterparts in LINPACK and EISPACK. One algorithm, xLASV2, can return significantly more accurate results if addition and subtraction have a guard digit, but is still quite accurate if they do not (see the end of section 4.9).

However, several LAPACK routines do make assumptions about details of the floating point arithmetic. We list these routines here.

Infinity and NaN arithmetic. In IEEE arithmetic, there are specific rules for evaluating quantities like 1/0 and 0/0. Overflowed quantities and division-by-zero (like 1/0) result in a $\pm\infty$ symbol, which continues to propagate through the computation using rules like $3/\infty = 0$. In particular, there is no error message or termination of execution. Similarly, quantities like 0/0 and ∞/∞ must be replaced by NaN (the "Not a Number" symbol) and propagated as well. See [4, 5] for details. The following LAPACK routines, and the routines that call them, assume the presence of this infinity and NaN arithmetic for their correct functioning:

- xSTEGR, which computes eigenvalues and eigenvectors of symmetric tridiagonal matrices. It is called by the drivers for the symmetric and Hermitian eigenproblems xSYEVR,

[3] Important machines that do not implement the IEEE standard include the Cray XMP, Cray YMP, Cray 2, Cray C90, IBM 370 and DEC Vax. Some architectures have two (or more) modes, one that implements IEEE arithmetic, and another that is less accurate but faster.

xHEEVR and xSTEVR.[4]

- Accuracy of add/subtract. If there is a *guard digit* in addition and subtraction, or if there is no guard digit but addition and subtraction are performed in the way they are on the Cray C90, Cray YMP, Cray XMP or Cray 2, then we can guarantee that the following routines work correctly. (They could theoretically fail on a hexadecimal or decimal machine without a guard digit, but we know of no such machine.)
  - xSTEDC, which uses divide-and-conquer to find the eigenvalues and eigenvectors of a symmetric tridiagonal matrix. It is called by all the drivers for the symmetric, Hermitian, generalized symmetric definite and generalized Hermitian definite eigenvalue drivers with names ending in -EVD or -GVD.
  - xBDSDC, which uses divide-and-conquer to find the SVD of a bidiagonal matrix. It is called by xGESDD.
  - xLALSD, which uses divide-and-conquer to solve a bidiagonal least squares problem with the SVD. It is called by xGELSD.

4.2 How to Measure Errors

LAPACK routines return four types of floating-point output arguments:

- *Scalar*, such as an eigenvalue of a matrix,
- *Vector*, such as the solution x of a linear system $Ax = b$,
- *Matrix*, such as a matrix inverse A^{-1}, and
- *Subspace*, such as the space spanned by one or more eigenvectors of a matrix.

This section provides measures for errors in these quantities, which we need in order to express error bounds.

First consider *scalars*. Let the scalar $\hat{\alpha}$ be an approximation of the true answer α. We can measure the difference between α and $\hat{\alpha}$ either by the **absolute error** $|\hat{\alpha} - \alpha|$, or, if α is nonzero, by the **relative error** $|\hat{\alpha} - \alpha|/|\alpha|$. Alternatively, it is sometimes more convenient to use $|\hat{\alpha} - \alpha|/|\hat{\alpha}|$ instead of the standard expression for relative error (see section 4.2.1). If the relative error of $\hat{\alpha}$ is, say 10^{-5}, then we say that $\hat{\alpha}$ is *accurate to 5 decimal digits.*

In order to measure the error in *vectors*, we need to measure the *size* or *norm* of a vector x. A popular norm is the magnitude of the largest component, $\max_{1 \le i \le n} |x_i|$, which we denote $\|x\|_\infty$. This is read *the infinity norm of* x. See Table 4.2 for a summary of norms.

[4] xSYEVR and the other drivers call ILAENV to check if IEEE arithmetic is available, and uses another algorithm if it is not. If LAPACK is installed by following the directions in Chapter 6, then ILAENV will return the correct information about the availability of IEEE arithmetic. If xSYEVR or the other drivers are used without this installation procedure, then the default is for ILAENV to say the IEEE arithmetic is available, since this is most common and faster.

Table 4.2: Vector and matrix norms

	Vector	Matrix
one-norm	$\|x\|_1 = \sum_i \|x_i\|$	$\|A\|_1 = \max_j \sum_i \|a_{ij}\|$
two-norm	$\|x\|_2 = (\sum_i \|x_i\|^2)^{1/2}$	$\|A\|_2 = \max_{x\neq 0} \|Ax\|_2/\|x\|_2$
Frobenius norm	$\|x\|_F = \|x\|_2$	$\|A\|_F = (\sum_{ij} \|a_{ij}\|^2)^{1/2}$
infinity-norm	$\|x\|_\infty = \max_i \|x_i\|$	$\|A\|_\infty = \max_i \sum_j \|a_{ij}\|$

If $\hat{x}$ is an approximation to the exact vector x, we will refer to $\|\hat{x} - x\|_p$ as the absolute error in $\hat{x}$ (where p is one of the values in Table 4.2), and refer to $\|\hat{x} - x\|_p/\|x\|_p$ as the relative error in $\hat{x}$ (assuming $\|x\|_p \neq 0$). As with scalars, we will sometimes use $\|\hat{x} - x\|_p/\|\hat{x}\|_p$ for the relative error. As above, if the relative error of $\hat{x}$ is, say 10^{-5}, then we say that $\hat{x}$ is accurate to 5 decimal digits. The following example illustrates these ideas:

$$x = \begin{pmatrix} 1 \\ 100 \\ 9 \end{pmatrix} , \quad \hat{x} = \begin{pmatrix} 1.1 \\ 99 \\ 11 \end{pmatrix}$$

$$\|\hat{x} - x\|_\infty = 2 , \quad \frac{\|\hat{x} - x\|_\infty}{\|x\|_\infty} = .02 , \quad \frac{\|\hat{x} - x\|_\infty}{\|\hat{x}\|_\infty} = .0202$$

$$\|\hat{x} - x\|_2 = 2.238 , \quad \frac{\|\hat{x} - x\|_2}{\|x\|_2} = .0223 , \quad \frac{\|\hat{x} - x\|_2}{\|\hat{x}\|_2} = .0225$$

$$\|\hat{x} - x\|_1 = 3.1 , \quad \frac{\|\hat{x} - x\|_1}{\|x\|_1} = .0282 , \quad \frac{\|\hat{x} - x\|_1}{\|\hat{x}\|_1} = .0279$$

Thus, we would say that $\hat{x}$ approximates x to 2 decimal digits.

Errors in *matrices* may also be measured with norms. The most obvious generalization of $\|x\|_\infty$ to matrices would appear to be $\|A\| = \max_{i,j} |a_{ij}|$, but this does not have certain important mathematical properties that make deriving error bounds convenient (see section 4.2.1). Instead, we will use $\|A\|_\infty = \max_{1\leq i\leq m} \sum_{j=1}^n |a_{ij}|$, where A is an m-by-n matrix, or $\|A\|_1 = \max_{1\leq j\leq n} \sum_{i=1}^m |a_{ij}|$; see Table 4.2 for other matrix norms. As before $\|\hat{A} - A\|_p$ is the absolute error in $\hat{A}$, $\|\hat{A} - A\|_p/\|A\|_p$ is the relative error in $\hat{A}$, and a relative error in $\hat{A}$ of 10^{-5} means $\hat{A}$ is accurate to 5 decimal digits. The following example illustrates these ideas:

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 10 \end{pmatrix} , \quad \hat{A} = \begin{pmatrix} .44 & 2.36 & 3.04 \\ 3.09 & 5.87 & 6.66 \\ 7.36 & 7.77 & 9.07 \end{pmatrix}$$

$$\|\hat{A} - A\|_\infty = 2.44 , \quad \frac{\|\hat{A} - A\|_\infty}{\|A\|_\infty} = .0976 , \quad \frac{\|\hat{A} - A\|_\infty}{\|\hat{A}\|_\infty} = .1008$$

$$\|\hat{A} - A\|_2 = 1.75 , \quad \frac{\|\hat{A} - A\|_2}{\|A\|_2} = .1007 , \quad \frac{\|\hat{A} - A\|_2}{\|\hat{A}\|_2} = .1020$$

$$\|\hat{A} - A\|_1 = 1.83\,, \quad \frac{\|\hat{A} - A\|_1}{\|A\|_1} = .0963\,, \quad \frac{\|\hat{A} - A\|_1}{\|\hat{A}\|_1} = .0975$$
$$\|\hat{A} - A\|_F = 1.87\,, \quad \frac{\|\hat{A} - A\|_F}{\|A\|_F} = .1075\,, \quad \frac{\|\hat{A} - A\|_F}{\|\hat{A}\|_F} = .1082$$

so $\hat{A}$ is accurate to 1 decimal digit.

Here is some related notation we will use in our error bounds. The **condition number of a matrix** A is defined as $\kappa_p(A) \equiv \|A\|_p \cdot \|A^{-1}\|_p$, where A is square and invertible, and p is ∞ or one of the other possibilities in Table 4.2. The condition number measures how sensitive A^{-1} is to changes in A; the larger the condition number, the more sensitive is A^{-1}. For example, for the same A as in the last example,

$$A^{-1} \approx \begin{pmatrix} -.667 & -1.333 & 1 \\ -.667 & 3.667 & -2 \\ 1 & -2 & 1 \end{pmatrix} \text{ and } \kappa_\infty(A) = 158.33$$

LAPACK error estimation routines typically compute a variable called `RCOND`, which is the reciprocal of the condition number (or an approximation of the reciprocal). The reciprocal of the condition number is used instead of the condition number itself in order to avoid the possibility of overflow when the condition number is very large. Also, some of our error bounds will use the vector of absolute values of x, $|x|$ ($|x|_i = |x_i|$), or similarly $|A|$ ($|A|_{ij} = |a_{ij}|$).

Now we consider errors in *subspaces*. Subspaces are the outputs of routines that compute eigenvectors and invariant subspaces of matrices. We need a careful definition of error in these cases for the following reason. The nonzero vector x is called a *(right) eigenvector* of the matrix A with *eigenvalue* λ if $Ax = \lambda x$. From this definition, we see that $-x$, $2x$, or any other nonzero multiple βx of x is also an eigenvector. In other words, eigenvectors are not unique. This means we cannot measure the difference between two supposed eigenvectors $\hat{x}$ and x by computing $\|\hat{x} - x\|_2$, because this may be large while $\|\hat{x} - \beta x\|_2$ is small or even zero for some $\beta \neq 1$. This is true even if we normalize x so that $\|x\|_2 = 1$, since both x and $-x$ can be normalized simultaneously. So in order to define error in a useful way, we need to instead consider the set $\mathcal{S}$ of all scalar multiples $\{\beta x\ ,\ \beta \text{ a scalar}\}$ of x. The set $\mathcal{S}$ is called the *subspace spanned by* x, and is uniquely determined by any nonzero member of $\mathcal{S}$. We will measure the difference between two such sets by the *acute angle* between them. Suppose $\hat{\mathcal{S}}$ is spanned by $\{\hat{x}\}$ and $\mathcal{S}$ is spanned by $\{x\}$. Then the acute angle between $\hat{\mathcal{S}}$ and $\mathcal{S}$ is defined as

$$\theta(\hat{\mathcal{S}}, \mathcal{S}) = \theta(\hat{x}, x) \equiv \arccos \frac{|\hat{x}^T x|}{\|\hat{x}\|_2 \cdot \|x\|_2}\ .$$

One can show that $\theta(\hat{x}, x)$ does not change when either $\hat{x}$ or x is multiplied by any nonzero scalar. For example, if

$$x = \begin{pmatrix} 1 \\ 100 \\ 9 \end{pmatrix} \text{ and } \hat{x} = \begin{pmatrix} 1.1 \\ 99 \\ 11 \end{pmatrix}$$

as above, then $\theta(\gamma\hat{x}, \beta x) = .0209$ for any nonzero scalars β and γ.

Here is another way to interpret the angle θ between $\hat{S}$ and S. Suppose $\hat{x}$ is a unit vector ($||\hat{x}||_2 = 1$). Then there is a scalar β such that

$$||\hat{x} - \beta x||_2 = \frac{\sqrt{2}\sin\theta}{\sqrt{1+\cos\theta}} \approx \theta \, .$$

The approximation $\approx \theta$ holds when θ is much less than 1 (less than .1 will do nicely). If $\hat{x}$ is an approximate eigenvector with error bound $\theta(\hat{x}, x) \le \bar{\theta} \ll 1$, where x is a true eigenvector, there is another true eigenvector βx satisfying $||\hat{x} - \beta x||_2 \lesssim \bar{\theta}$. For example, if

$$\hat{x} = \begin{pmatrix} 1.1 \\ 99 \\ 11 \end{pmatrix} \cdot (1.1^2 + 99^2 + 11^2)^{-1/2} \text{ so } ||\hat{x}||_2 = 1 \text{ and } x = \begin{pmatrix} 1 \\ 100 \\ 9 \end{pmatrix}$$

then $||\hat{x} - \beta x||_2 \approx .0209$ for $\beta \approx .001$.

Some LAPACK routines also return subspaces spanned by more than one vector, such as the invariant subspaces of matrices returned by xGEESX. The notion of angle between subspaces also applies here; see section 4.2.1 for details.

Finally, many of our error bounds will contain a factor $p(n)$ (or $p(m, n)$), which grows as a function of matrix dimension n (or dimensions m and n). It represents a potentially different function for each problem. In practice, the true errors usually grow just linearly; using $p(n) = 10n$ in the error bound formulas will often give a reasonable bound. Therefore, we will refer to $p(n)$ as a "modestly growing" function of n. However it can occasionally be much larger, see section 4.2.1. **For simplicity, the error bounds computed by the code fragments in the following sections will use $p(n) = 1$. This means these computed error bounds may occasionally slightly underestimate the true error. For this reason we refer to these computed error bounds as "approximate error bounds".**

4.2.1 Further Details: How to Measure Errors

The relative error $|\hat{\alpha} - \alpha|/|\alpha|$ in the approximation $\hat{\alpha}$ of the true solution α has a drawback: it often cannot be computed directly, because it depends on the unknown quantity $|\alpha|$. However, we can often instead estimate $|\hat{\alpha} - \alpha|/|\hat{\alpha}|$, since $\hat{\alpha}$ is known (it is the output of our algorithm). Fortunately, these two quantities are necessarily close together, provided either one is small, which is the only time they provide a useful bound anyway. For example, $|\hat{\alpha} - \alpha|/|\hat{\alpha}| \le .1$ implies

$$9\frac{|\hat{\alpha} - \alpha|}{|\hat{\alpha}|} \le \frac{|\hat{\alpha} - \alpha|}{|\alpha|} \le 1.1\frac{|\hat{\alpha} - \alpha|}{|\hat{\alpha}|} \, ,$$

so they can be used interchangeably.

Table 4.2 contains a variety of norms we will use to measure errors. These norms have the properties that $||Ax||_p \le ||A||_p \cdot ||x||_p$, and $||AB||_p \le ||A||_p \cdot ||B||_p$, where p is one of 1, 2, ∞, and F. These properties are useful for deriving error bounds.

An error bound that uses a given norm may be changed into an error bound that uses another norm. This is accomplished by multiplying the first error bound by an appropriate function of the

problem dimension. Table 4.3 gives the factors $f_{pq}(n)$ such that $\|x\|_p \le f_{pq}(n)\|x\|_q$, where n is the dimension of x.

Table 4.3: Bounding One Vector Norm in Terms of Another

Values of $f_{pq}(n)$ such that $\|x\|_p \le f_{pq}(n)\|x\|_q$, where x is an n-vector

		q		
		1	2	∞
p	1	1	$\sqrt{n}$	n
	2	1	1	$\sqrt{n}$
	∞	1	1	1

Table 4.4 gives the factors $f_{pq}(m,n)$ such that $\|A\|_p \le f_{pq}(m,n)\|A\|_q$, where A is m-by-n.

The two-norm of A, $\|A\|_2$, is also called the **spectral norm** of A, and is equal to the **largest singular value** $\sigma_{\max}(A)$ of A. We shall also need to refer to the **smallest singular value** $\sigma_{\min}(A)$ of A; its value can be defined in a similar way to the definition of the two-norm in Table 4.2, namely as $\min_{x\neq 0} \|Ax\|_2/\|x\|_2$ when A has at least as many rows as columns, and defined as $\min_{x\neq 0} \|A^T x\|_2/\|x\|_2$ when A has more columns than rows. The two-norm, Frobenius norm, and singular values of a matrix do not change if the matrix is multiplied by a real orthogonal (or complex unitary) matrix.

Now we define *subspaces* spanned by more than one vector, and *angles between subspaces.* Given a set of k n-dimensional vectors $\{x_1, ..., x_k\}$, they determine a subspace $\mathcal{S}$ consisting of all their possible linear combinations $\{\sum_{i=1}^k \beta_i x_i,\ \beta_i \text{ scalars }\}$. We also say that $\{x_1, ..., x_k\}$ *spans* $\mathcal{S}$. The difficulty in measuring the difference between subspaces is that the sets of vectors spanning them are not unique. For example, $\{x\}$, $\{-x\}$ and $\{2x\}$ all determine the same subspace. This means we cannot simply compare the subspaces spanned by $\{\hat{x}_1, ..., \hat{x}_k\}$ and $\{x_1, ..., x_k\}$ by comparing each $\hat{x}_i$ to x_i. Instead, we will measure the *angle* between the subspaces, which is independent of the spanning set of vectors. Suppose subspace $\hat{\mathcal{S}}$ is spanned by $\{\hat{x}_1, ..., \hat{x}_k\}$ and that subspace $\mathcal{S}$ is spanned by $\{x_1, ..., x_k\}$. If $k = 1$, we instead write more simply $\{\hat{x}\}$ and $\{x\}$. When $k = 1$, we defined the angle $\theta(\hat{\mathcal{S}}, \mathcal{S})$ between $\hat{\mathcal{S}}$ and $\mathcal{S}$ as the acute angle between $\hat{x}$ and x. When $k > 1$, we define the acute angle between $\hat{\mathcal{S}}$ and $\mathcal{S}$ as the largest acute angle between any vector $\hat{x}$ in $\hat{\mathcal{S}}$, and the closest vector x in $\mathcal{S}$ to $\hat{x}$:

$$\theta(\hat{\mathcal{S}}, \mathcal{S}) \equiv \max_{\substack{\hat{x}\in\hat{\mathcal{S}} \\ \hat{x}\neq 0}} \min_{\substack{x\in\mathcal{S} \\ x\neq 0}} \theta(\hat{x}, x) \ .$$

LAPACK routines which compute subspaces return vectors $\{\hat{x}_1, ..., \hat{x}_k\}$ spanning a subspace $\hat{\mathcal{S}}$ which are *orthonormal.* This means the n-by-k matrix $\hat{X} = [\hat{x}_1, ..., \hat{x}_k]$ satisfies $\hat{X}^H \hat{X} = I$. Suppose also that the vectors $\{x_1, ..., x_k\}$ spanning $\mathcal{S}$ are orthonormal, so $X = [x_1, ..., x_k]$ also satisfies $X^H X = I$. Then there is a simple expression for the angle between $\hat{\mathcal{S}}$ and $\mathcal{S}$:

$$\theta(\hat{\mathcal{S}}, \mathcal{S}) = \arccos \sigma_{\min}(\hat{X}^H X) \ .$$

Table 4.4: Bounding One Matrix Norm in Terms of Another

Values of $f_{pq}(m,n)$ such that $\|A\|_p \le f_{pq}(m,n)\|A\|_q$, where A is m-by-n

		q			
		1	2	F	∞
p	1	1	$\sqrt{m}$	$\sqrt{m}$	m
	2	$\sqrt{n}$	1	1	$\sqrt{m}$
	F	$\sqrt{n}$	$\sqrt{\min(m,n)}$	1	$\sqrt{m}$
	∞	n	$\sqrt{n}$	$\sqrt{n}$	1

For example, if

$$\hat{X} = \begin{pmatrix} -.79996 & .60005 \\ -.59997 & -.79990 \\ -.01 & -.01 \end{pmatrix} \quad \text{and} \quad X = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}$$

then $\theta(\hat{\mathcal{S}}, \mathcal{S}) = .01414$.

As stated above, all our bounds will contain a factor $p(n)$ (or $p(m,n)$), which measure how roundoff errors can grow as a function of matrix dimension n (or m and m). In practice, the true error usually grows just linearly with n, but we can generally only prove much weaker bounds of the form $p(n) = O(n^3)$. This is because we can not rule out the extremely unlikely possibility of rounding errors all adding together instead of canceling on average. Using $p(n) = O(n^3)$ would give very pessimistic and unrealistic bounds, especially for large n, so we content ourselves with describing $p(n)$ as a "modestly growing" polynomial function of n. Using $p(n) = 10n$ in the error bound formulas will often give a reasonable bound. For detailed derivations of various $p(n)$, see [55, 67, 103].

There is also one situation where $p(n)$ can grow as large as 2^{n-1}: Gaussian elimination. This typically occurs only on specially constructed matrices presented in numerical analysis courses [103, p. 212][67]. However, the expert drivers for solving linear systems, xGESVX and xGBSVX, provide error bounds incorporating $p(n)$, and so this rare possibility can be detected.

4.3 Further Details: How Error Bounds Are Derived

4.3.1 Standard Error Analysis

We illustrate standard error analysis with the simple example of evaluating the scalar function $y = f(z)$. Let the output of the subroutine which implements $f(z)$ be denoted alg(z); this includes the effects of roundoff. If alg$(z) = f(z+\delta)$ where δ is small, then we say alg is a **backward stable** algorithm for f, or that the **backward error** δ is small. In other words, alg(z) is the exact value of f at a slightly perturbed input $z + \delta$.[5]

[5]Sometimes our algorithms satisfy only alg$(z) = f(z+\delta) + \eta$ where both δ and η are small. This does not significantly change the following analysis.

Suppose now that f is a smooth function, so that we may approximate it near z by a straight line: $f(z+\delta) \approx f(z) + f'(z) \cdot \delta$. Then we have the simple error estimate

$$\text{alg}(z) - f(z) = f(z+\delta) - f(z) \approx f'(z) \cdot \delta.$$

Thus, if δ is small, and the derivative $f'(z)$ is moderate, the error $\text{alg}(z) - f(z)$ will be small[6]. This is often written in the similar form

$$\left|\frac{\text{alg}(z) - f(z)}{f(z)}\right| \approx \left|\frac{f'(z) \cdot z}{f(z)}\right| \cdot \left|\frac{\delta}{z}\right| \equiv \kappa(f,z) \cdot \left|\frac{\delta}{z}\right|.$$

This approximately bounds the **relative error** $\left|\frac{\text{alg}(z)-f(z)}{f(z)}\right|$ by the product of the **condition number of f at z**, $\kappa(f,z)$, and the **relative backward error** $|\frac{\delta}{z}|$. Thus we get an error bound by multiplying a condition number and a backward error (or bounds for these quantities). We call a problem **ill-conditioned** if its condition number is large, and **ill-posed** if its condition number is infinite (or does not exist)[7].

If f and z are vector quantities, then $f'(z)$ is a matrix (the Jacobian). So instead of using absolute values as before, we now measure δ by a vector norm $\|\delta\|$ and $f'(z)$ by a matrix norm $\|f'(z)\|$. The conventional (and coarsest) error analysis uses a norm such as the infinity norm. We therefore call this **normwise backward stability**. For example, a normwise stable method for solving a system of linear equations $Ax = b$ will produce a solution $\hat{x}$ satisfying $(A+E)\hat{x} = b + f$ where $\|E\|_\infty / \|A\|_\infty$ and $\|f\|_\infty / \|b\|_\infty$ are both small (close to machine epsilon). In this case the condition number is $\kappa_\infty(A) = \|A\|_\infty \cdot \|A^{-1}\|_\infty$ (see section 4.4 below).

Almost all of the algorithms in LAPACK (as well as LINPACK and EISPACK) are stable in the sense just described[8]: when applied to a matrix A they produce the exact result for a slightly different matrix $A + E$, where $\|E\|_\infty / \|A\|_\infty$ is of order ϵ. In a certain sense, a user can hardly ask for more, provided the data is at all uncertain.

It is often possible to compute the norm $\|E\|$ of the actual backward error by computing a residual r, such as $r = Ax - b$ or $r = Ax - \lambda x$, and suitably scaling its norm $\|r\|$. The expert driver routines for solving $Ax = b$ do this, for example. For details see [55, 67, 85, 95].

Condition numbers may be expensive to compute exactly. For example, it costs about $\frac{2}{3}n^3$ operations to solve $Ax = b$ for a general matrix A, and computing $\kappa_\infty(A)$ *exactly* costs an additional $\frac{4}{3}n^3$ operations, or twice as much. But $\kappa_\infty(A)$ can be *estimated* in only $O(n^2)$ operations beyond those $\frac{2}{3}n^3$ necessary for solution, a tiny extra cost. Therefore, most of LAPACK's condition numbers

[6]More generally, we only need Lipschitz continuity of f, and may use the Lipschitz constant in place of f' in deriving error bounds.

[7]This is a different use of the term ill-posed than used in other contexts. For example, to be well-posed (not ill-posed) in the sense of Hadamard, it is sufficient for f to be continuous, whereas we require Lipschitz continuity.

[8]There are some caveats to this statement. When computing the inverse of a matrix, the backward error E is small taking the columns of the computed inverse one at a time, with a different E for each column [47]. The same is true when computing the eigenvectors of a nonsymmetric matrix. When computing the eigenvalues and eigenvectors of $A - \lambda B$, $AB - \lambda I$ or $BA - \lambda I$, with A symmetric and B symmetric and positive definite (using xSYGV or xHEGV) then the method may not be backward normwise stable if B has a large condition number $\kappa_\infty(B)$, although it has useful error bounds in this case too (see section 4.10). Solving the Sylvester equation $AX + XB = C$ for the matrix X may not be backward stable, although there are again useful error bounds for X [66].

and error bounds are based on estimated condition numbers, using the method of [59, 62, 63]. The price one pays for using an estimated rather than an exact condition number is occasional (but very rare) underestimates of the true error; years of experience attest to the reliability of our estimators, although examples where they badly underestimate the error can be constructed [65]. Note that once a condition estimate is large enough, (usually $O(1/\epsilon)$), it confirms that the computed answer may be completely inaccurate, and so the exact magnitude of the condition estimate conveys little information.

4.3.2 Improved Error Bounds

The standard error analysis just outlined has a drawback: by using the infinity norm $\|\delta\|_\infty$ to measure the backward error, entries of equal magnitude in δ contribute equally to the final error bound $\kappa(f,z)(\|\delta\|/\|z\|)$. This means that if z is sparse or has some very tiny entries, a normwise backward stable algorithm may make very large changes in these entries compared to their original values. If these tiny values are known accurately by the user, these errors may be unacceptable, or the error bounds may be unacceptably large.

For example, consider solving a diagonal system of linear equations $Ax = b$. Each component of the solution is computed accurately by Gaussian elimination: $x_i = b_i/a_{ii}$. The usual error bound is approximately $\epsilon \cdot \kappa_\infty(A) = \epsilon \cdot \max_i |a_{ii}| / \min_i |a_{ii}|$, which can arbitrarily overestimate the true error, ϵ, if at least one a_{ii} is tiny and another one is large.

LAPACK addresses this inadequacy by providing some algorithms whose backward error δ is a tiny relative change in each component of z: $|\delta_i| = O(\epsilon)|z_i|$. This backward error retains both the sparsity structure of z as well as the information in tiny entries. These algorithms are therefore called **componentwise relatively backward stable**. Furthermore, computed error bounds reflect this stronger form of backward error[9].

If the input data has independent uncertainty in each component, each component must have at least a small *relative* uncertainty, since each is a floating-point number. In this case, the extra uncertainty contributed by the algorithm is not much worse than the uncertainty in the input data, so one could say the answer provided by a componentwise relatively backward stable algorithm is as accurate as the data warrants [1].

When solving $Ax = b$ using expert driver xyySVX or computational routine xyyRFS, for example, we almost always compute $\hat{x}$ satisfying $(A + E)\hat{x} = b + f$, where e_{ij} is a small relative change in a_{ij} and f_k is a small relative change in b_k. In particular, if A is diagonal, the corresponding error bound is always tiny, as one would expect (see the next section).

LAPACK can achieve this accuracy for linear equation solving, the bidiagonal singular value decomposition, and the symmetric tridiagonal eigenproblem, and provides facilities for achieving this accuracy for least squares problems. Future versions of LAPACK will also achieve this accuracy for other linear algebra problems, as discussed below.

[9]For other algorithms, the answers (and computed error bounds) are as accurate as though the algorithms were componentwise relatively backward stable, even though they are not. These algorithms are called *componentwise relatively forward stable.*

4.4 Error Bounds for Linear Equation Solving

Let $Ax = b$ be the system to be solved, and $\hat{x}$ the computed solution. Let n be the dimension of A. An approximate error bound for $\hat{x}$ may be obtained in one of the following two ways, depending on whether the solution is computed by a simple driver or an expert driver:

1. Suppose that $Ax = b$ is solved using the simple driver SGESV (subsection 2.3.1). Then the approximate error bound[10]

$$\frac{\|\hat{x} - x\|_\infty}{\|x\|_\infty} \leq \texttt{ERRBD}$$

can be computed by the following code fragment.

```
      EPSMCH = SLAMCH( 'E' )
*     Get infinity-norm of A
      ANORM = SLANGE( 'I', N, N, A, LDA, WORK )
*     Solve system; The solution X overwrites B
      CALL SGESV( N, 1, A, LDA, IPIV, B, LDB, INFO )
      IF( INFO.GT.0 ) THEN
         PRINT *,'Singular Matrix'
      ELSE IF (N .GT. 0) THEN
*        Get reciprocal condition number RCOND of A
         CALL SGECON( 'I', N, A, LDA, ANORM, RCOND, WORK, IWORK, INFO )
         RCOND = MAX( RCOND, EPSMCH )
         ERRBD = EPSMCH / RCOND
      END IF
```

For example, suppose[11] $\texttt{SLAMCH('E')} = 2^{-24} = 5.961 \cdot 10^{-8}$,

$$A = \begin{pmatrix} 4 & 16000 & 17000 \\ 2 & 5 & 8 \\ 3 & 6 & 10 \end{pmatrix} \text{ and } b = \begin{pmatrix} 100.1 \\ .1 \\ .01 \end{pmatrix}.$$

Then (to 4 decimal places)

$$x = \begin{pmatrix} -.3974 \\ -.3349 \\ .3211 \end{pmatrix}, \quad \hat{x} = \begin{pmatrix} -.3968 \\ -.3344 \\ .3207 \end{pmatrix},$$

ANORM $= 3.300 \cdot 10^4$, RCOND $= 3.907 \cdot 10^{-6}$, the true reciprocal condition number $= 3.902 \cdot 10^{-6}$, ERRBD $= 1.5 \cdot 10^{-2}$, and the true error $= 1.5 \cdot 10^{-3}$.

[10] As discussed in section 4.2, this approximate error bound may underestimate the true error by a factor $p(n)$ which is a modestly growing function of the problem dimension n. Often $p(n) \leq 10n$.

[11] This and other numerical examples were computed in IEEE single precision arithmetic [4] on a DEC 5000/120 workstation.

2. Suppose that $Ax = b$ is solved using the expert driver SGESVX (subsection 2.3.1). This routine provides an explicit error bound **FERR**, measured with the infinity-norm:

$$\frac{\|\hat{x} - x\|_\infty}{\|x\|_\infty} \leq \texttt{FERR}$$

For example, the following code fragment solves $Ax = b$ and computes an approximate error bound **FERR**:

```
 CALL SGESVX( 'E', 'N', N, 1, A, LDA, AF, LDAF, IPIV,
$             EQUED, R, C, B, LDB, X, LDX, RCOND, FERR, BERR,
$             WORK, IWORK, INFO )
 IF( INFO.GT.0 ) PRINT *,'(Nearly) Singular Matrix'
```

For the same A and b as above, $\hat{x} = \begin{pmatrix} -.3974 \\ -.3349 \\ .3211 \end{pmatrix}$, $\texttt{FERR} = 3.0 \cdot 10^{-5}$, and the actual error is $4.3 \cdot 10^{-7}$.

This example illustrates that the expert driver provides an error bound with less programming effort than the simple driver, and also that it may produce a significantly more accurate answer.

Similar code fragments, with obvious adaptations, may be used with all the driver routines for linear equations listed in Table 2.2. For example, if a symmetric system is solved using the simple driver xSYSV, then xLANSY must be used to compute **ANORM**, and xSYCON must be used to compute **RCOND**.

4.4.1 Further Details: Error Bounds for Linear Equation Solving

The conventional error analysis of linear equation solving goes as follows. Let $Ax = b$ be the system to be solved. Let $\hat{x}$ be the solution computed by LAPACK (or LINPACK) using any of their linear equation solvers. Let r be the residual $r = b - A\hat{x}$. In the absence of rounding error r would be zero and $\hat{x}$ would equal x; with rounding error one can only say the following:

> The normwise backward error of the computed solution $\hat{x}$, with respect to the infinity norm, is the pair E, f which minimizes
>
> $$\max\left(\frac{\|E\|_\infty}{\|A\|_\infty}, \frac{\|f\|_\infty}{\|b\|_\infty}\right)$$
>
> subject to the constraint $(A + E)\hat{x} = b + f$. The minimal value of $\max\left(\frac{\|E\|_\infty}{\|A\|_\infty}, \frac{\|f\|_\infty}{\|b\|_\infty}\right)$ is given by
>
> $$\omega_\infty = \frac{\|r\|_\infty}{\|A\|_\infty \cdot \|\hat{x}\|_\infty + \|b\|_\infty}.$$

One can show that the computed solution $\hat{x}$ satisfies $\omega_\infty \leq p(n) \cdot \epsilon$, where $p(n)$ is a modestly growing function of n. The corresponding condition number is $\kappa_\infty(A) \equiv \|A\|_\infty \cdot \|A^{-1}\|_\infty$. The error $x - \hat{x}$ is bounded by

$$\frac{\|x - \hat{x}\|_\infty}{\|x\|_\infty} \lesssim 2 \cdot \omega_\infty \cdot \kappa_\infty(A) = \texttt{ERRBD} .$$

In the first code fragment in the last section, $2 \cdot \omega_\infty$, which is $4.504 \cdot 10^{-8}$ in the numerical example, is approximated by $\epsilon = 2^{-24} = 5.960 \cdot 10^{-8}$. Approximations of $\kappa_\infty(A)$ — or, strictly speaking, its reciprocal **RCOND** — are returned by computational routines xyyCON (subsection 2.4.1) or driver routines xyySVX (subsection 2.3.1). The code fragment makes sure **RCOND** is at least $\epsilon =$ **EPSMCH** to avoid overflow in computing **ERRBD**. This limits **ERRBD** to a maximum of 1, which is no loss of generality since a relative error of 1 or more indicates the same thing: a complete loss of accuracy. Note that the value of **RCOND** returned by xyySVX may apply to a linear system obtained from $Ax = b$ by *equilibration*, i.e. scaling the rows and columns of A in order to make the condition number smaller. This is the case in the second code fragment in the last section, where the program chose to scale the rows by the factors returned in $\texttt{R} = (5.882 \cdot 10^{-5}, .125, .1)$ and scale the columns by the factors returned in $\texttt{C} = (3.333, 1.063, 1.)$, resulting in $\texttt{RCOND} = 3.454 \cdot 10^{-3}$.

As stated in section 4.3.2, this approach does not respect the presence of zero or tiny entries in A. In contrast, the LAPACK computational routines xyyRFS (subsection 2.4.1) or driver routines xyySVX (subsection 2.3.1) will (except in rare cases) compute a solution $\hat{x}$ with the following properties:

The componentwise backward error of the computed solution $\hat{x}$ is the pair E, f which minimizes

$$\max_{i,j,k} \left(\frac{|e_{ij}|}{|a_{ij}|}, \frac{|f_k|}{|b_k|} \right)$$

(where we interpret 0/0 as 0) subject to the constraint $(A + E)\hat{x} = b + f$. The minimal value of $\max_{i,j,k} \left(\frac{|e_{ij}|}{|a_{ij}|}, \frac{|f_k|}{|b_k|} \right)$ is given by

$$\omega_c = \max_i \frac{|r_i|}{(|A| \cdot |\hat{x}| + |b|)_i} .$$

One can show that for most problems the $\hat{x}$ computed by xyySVX satisfies $\omega_c \leq p(n) \cdot \epsilon$, where $p(n)$ is a modestly growing function of n. In other words, $\hat{x}$ is the exact solution of the perturbed problem $(A + E)\hat{x} = b + f$ where E and f are small relative perturbations in each entry of A and b, respectively. The corresponding condition number is $\kappa_c(A, b, \hat{x}) \equiv \| \, |A^{-1}|(|A| \cdot |\hat{x}| + |b|) \, \|_\infty / \|\hat{x}\|_\infty$. The error $x - \hat{x}$ is bounded by

$$\frac{\|x - \hat{x}\|_\infty}{\|\hat{x}\|_\infty} \leq \omega_c \cdot \kappa_c(A, b, \hat{x}).$$

The routines xyyRFS and xyySVX return ω_c, which is called **BERR** (for Backward ERRor), and a bound on the the actual error $\|x - \hat{x}\|_\infty / \|\hat{x}\|_\infty$, called **FERR** (for Forward ERRor), as in the second code fragment in the last section. **FERR** is actually calculated by the following formula, which can be smaller than the bound $\omega_c \cdot \kappa_c(A, b, \hat{x})$ given above:

$$\frac{\|x - \hat{x}\|_\infty}{\|\hat{x}\|_\infty} \le \mathbf{FERR} = \frac{\| \, |A^{-1}|(|\hat{r}| + n\epsilon(|A| \cdot |\hat{x}| + |b|))\|_\infty}{\|\hat{x}\|_\infty} \ .$$

Here, $\hat{r}$ is the computed value of the residual $b - A\hat{x}$, and the norm in the numerator is estimated using the same estimation subroutine used for **RCOND**.

The value of **BERR** for the example in the last section is $4.6 \cdot 10^{-8}$.

Even in the rare cases where xyyRFS fails to make **BERR** close to its minimum ϵ, the error bound **FERR** may remain small. See [6] for details.

4.5 Error Bounds for Linear Least Squares Problems

The linear least squares problem is to find x that minimizes $\|Ax - b\|_2$. We discuss error bounds for the most common case where A is m-by-n with $m > n$, and A has full rank; this is called an *overdetermined least squares problem* (the following code fragments deal with $m = n$ as well).

Let $\hat{x}$ be the solution computed by one of the driver routines xGELS, xGELSX, xGELSY, xGELSS, or xGELSD (see section 2.3.2). An approximate error bound[10]

$$\frac{\|\hat{x} - x\|_2}{\|x\|_2} \lesssim \mathbf{ERRBD}$$

may be computed in one of the following ways, depending on which type of driver routine is used:

1. Suppose the simple driver SGELS is used:

```
      EPSMCH = SLAMCH( 'E' )
*     Get the 2-norm of the right hand side B
      BNORM = SNRM2( M, B, 1 )
*     Solve the least squares problem; the solution X overwrites B
      CALL SGELS( 'N', M, N, 1, A, LDA, B, LDB, WORK, LWORK, INFO )
      IF ( MIN(M,N) .GT. 0 ) THEN
*        Get the 2-norm of the residual A*X-B
         RNORM = SNRM2( M-N, B( N+1 ), 1 )
*        Get the reciprocal condition number RCOND of A
         CALL STRCON('I', 'U', 'N', N, A, LDA, RCOND, WORK, IWORK, INFO)
         RCOND = MAX( RCOND, EPSMCH )
         IF ( BNORM .GT. 0.0 ) THEN
            SINT = RNORM / BNORM
         ELSE
            SINT = 0.0
```

```
         ENDIF
         COST = MAX( SQRT( (1.0E0 - SINT)*(1.0E0 + SINT) ), EPSMCH )
         TANT = SINT / COST
         ERRBD = EPSMCH*( 2.0E0/(RCOND*COST) + TANT / RCOND**2 )
      ENDIF
```

For example[11], if SLAMCH('E') $= 2^{-24} = 5.961 \cdot 10^{-8}$,

$$A = \begin{pmatrix} 4 & 3 & 5 \\ 2 & 5 & 8 \\ 3 & 6 & 10 \\ 4 & 5 & 11 \end{pmatrix} \text{ and } b = \begin{pmatrix} 100.1 \\ .1 \\ .01 \\ .01 \end{pmatrix},$$

then, to 4 decimal places,

$$x = \hat{x} = \begin{pmatrix} 38.49 \\ 21.59 \\ -23.88 \end{pmatrix},$$

BNORM $= 100.1$, RNORM $= 8.843$, RCOND $= 4.712 \cdot 10^{-2}$, ERRBD $= 4.9 \cdot 10^{-6}$, and the true error is $4.6 \cdot 10^{-7}$.

2. Suppose the expert driver SGELSX or SGELSY is used. This routine has an input argument RCND, which is used to determine the rank of the input matrix (briefly, the matrix is considered not to have full rank if its condition number exceeds 1/RCND). The code fragment below only computes error bounds if the matrix has been determined to have full rank. When the matrix does not have full rank, computing and interpreting error bounds is more complicated, and the reader is referred to the next section.

```
      EPSMCH = SLAMCH( 'E' )
*     Get the 2-norm of the right hand side B
      BNORM = SNRM2( M, B, 1 )
*     Solve the least squares problem; the solution X overwrites B
      RCND = 0
      CALL SGELSX( M, N, 1, A, LDA, B, LDB, JPVT, RCND, RANK, WORK,
     $             INFO )
      IF ( RANK.LT.N ) THEN
         PRINT *,'Matrix less than full rank'
      ELSE IF ( MIN( M,N ) .GT. 0 ) THEN
*        Get the 2-norm of the residual A*X-B
         RNORM = SNRM2( M-N, B( N+1 ), 1 )
*        Get the reciprocal condition number RCOND of A
         CALL STRCON('I', 'U', 'N', N, A, LDA, RCOND, WORK, IWORK, INFO)
         RCOND = MAX( RCOND, EPSMCH )
         IF ( BNORM .GT. 0.0 ) THEN
            SINT = RNORM / BNORM
         ELSE
```

```
      SINT = 0.0
   ENDIF
   COST = MAX( SQRT( (1.0E0 - SINT)*(1.0E0 + SINT) ), EPSMCH )
   TANT = SINT / COST
   ERRBD = EPSMCH*( 2.0E0/(RCOND*COST) + TANT / RCOND**2 )
END IF
```

The numerical results of this code fragment on the above A and b are the same as for the first code fragment.

3. Suppose the other type of expert driver SGELSS or SGELSD is used. This routine also has an input argument RCND, which is used to determine the rank of the matrix A. The same code fragment can be used to compute error bounds as for SGELSX or SGELSY, except that the call to SGELSX must be replaced by:

```
 CALL SGELSD( M, N, 1, A, LDA, B, LDB, S, RCND, RANK, WORK, LWORK,
$             IWORK, INFO )
```

and the call to STRCON must be replaced by:

```
RCOND = S( N ) / S( 1 )
```

Applied to the same A and b as above, the computed $\hat{x}$ is nearly the same, RCOND $= 5.428 \cdot 10^{-2}$, ERRBD $= 4.0 \cdot 10^{-6}$, and the true error is $6.6 \cdot 10^{-7}$.

4.5.1 Further Details: Error Bounds for Linear Least Squares Problems

The conventional error analysis of linear least squares problems goes as follows. As above, let $\hat{x}$ be the solution to minimizing $\|Ax - b\|_2$ computed by LAPACK using one of the least squares drivers xGELS, xGELSX, xGELSY, xGELSS or xGELSD (see subsection 2.3.2). We discuss the most common case, where A is overdetermined (i.e., has more rows than columns) and has full rank [16, 25, 55, 67]:

The computed solution $\hat{x}$ has a small normwise backward error. In other words $\hat{x}$ minimizes $\|(A+E)\hat{x} - (b+f)\|_2$, where E and f satisfy

$$\max\left(\frac{\|E\|_2}{\|A\|_2}, \frac{\|f\|_2}{\|b\|_2}\right) \le p(n)\epsilon$$

and $p(n)$ is a modestly growing function of n. We take $p(n) = 1$ in the code fragments above. Let $\kappa_2(A) = \sigma_{\max}(A)/\sigma_{\min}(A)$ (approximated by 1/RCOND in the above code fragments), $\rho = \|A\hat{x} - b\|_2$ (= RNORM above), and $\sin(\theta) = \rho/\|b\|_2$ (SINT = RNORM /

BNORM above). Here, θ is the acute angle between the vectors $A\hat{x}$ and b. Then when $p(n)\epsilon$ is small, the error $\hat{x} - x$ is bounded by

$$\frac{\|x - \hat{x}\|_2}{\|x\|_2} \lesssim p(n)\epsilon \left\{ \frac{2\kappa_2(A)}{\cos(\theta)} + \tan(\theta)\kappa_2^2(A) \right\},$$

where $\cos(\theta) =$ COST and $\tan(\theta) =$ TANT in the code fragments above.

We avoid overflow by making sure RCOND and COST are both at least $\epsilon =$ EPSMCH, and by handling the case of a zero B matrix separately (BNORM = 0).

$\kappa_2(A) = \sigma_{\max}(A)/\sigma_{\min}(A)$ may be computed directly from the singular values of A returned by xGELSS or xGELSD (as in the code fragment) or by xGESVD or xGESDD. It may also be approximated by using xTRCON following calls to xGELS, xGELSX or xGELSY. xTRCON estimates κ_∞ or κ_1 instead of κ_2, but these can differ from κ_2 by at most a factor of n.

If A is rank-deficient, xGELSS, xGELSD, xGELSY and xGELSX can be used to **regularize** the problem by treating all singular values less than a user-specified threshold (RCND $\cdot\, \sigma_{\max}(A)$) as exactly zero. The number of singular values treated as nonzero is returned in RANK. See [16, 55, 67] for error bounds in this case, as well as [28] for the underdetermined case. The ability to deal with rank-deficient matrices is the principal attraction of these four drivers, which are more expensive than the simple driver xGELS.

The solution of the overdetermined, full-rank problem may also be characterized as the solution of the linear system of equations

$$\begin{pmatrix} I & A \\ A^T & 0 \end{pmatrix} \begin{pmatrix} r \\ x \end{pmatrix} = \begin{pmatrix} b \\ 0 \end{pmatrix}$$

By solving this linear system using xyyRFS or xyySVX (see section 4.4) componentwise error bounds can also be obtained [7].

4.6 Error Bounds for Generalized Least Squares Problems

There are two kinds of generalized least squares problems that are discussed in section 4.6: the *linear equality constrained least squares problem* and the *general linear model problem.*

4.6.1 Linear Equality Constrained Least Squares Problem

The linear equality constrained least squares (LSE) problem is

$$\min_x \|Ax - b\| \quad \text{subject to} \quad Bx = d$$

where A is an m-by-n matrix, B is a p-by-n matrix, b is an m vector, and d is a p vector, with $p \le n \le p + m$.

The LSE problem is solved by the driver routine xGGLSE (see section 4.6). Let $\widehat{x}$ be the value of x computed by xGGLSE. The approximate error bound

$$\frac{\|x - \widehat{x}\|_2}{\|x\|_2} \lesssim \text{ERRBD}.$$

can be computed by the following code fragment

```
      EPSMCH = SLAMCH( 'E' )
*     Get the 2-norm of the right hand side C
      CNORM = SNRM2( M, C, 1 )
      print*,'CNORM = ',CNORM
*     Solve the least squares problem with equality constraints
      CALL SGGLSE( M, N, P, A, LDA, B, LDA, C, D, Xc, WORK, LWORK, IWORK, INFO )
*     Get the Frobenius norm of A and B
      ANORM = SLANTR( 'F', 'U', 'N', N, N, A, LDA, WORK )
      BNORM = SLANTR( 'F', 'U', 'N', P, P, B( 1, N-P+1 ), LDA, WORK )
      MN = MIN( M, N )
      IF( N.EQ.P ) THEN
         APPSNM = ZERO
         RNORM = SLANTR( '1', 'U', 'N', N, N, B, LDB, WORK(P+MN+N+1) )
         CALL STRCON( '1', 'U', 'N', N, B, LDB, RCOND, WORK( P+MN+N+1 ),
     $                IWORK, INFO )
         BAPSNM = ONE/ (RCOND * RNORM )
      ELSE
*        Estimate norm of (AP)^+
         RNORM = SLANTR( '1', 'U', 'N', N-P, N-P, A, LDA, WORK(P+MN+1) )
         CALL STRCON( '1', 'U', 'N', N-P, A, LDA, RCOND, WORK( P+MN+1 ),
     $                IWORK, INFO )
         APPSNM = ONE/ (RCOND * RNORM )
*        Estimate norm of B^+_A
         KASE = 0
         CALL SLACON( P, WORK( P+MN+1 ), WORK( P+MN+N+1 ), IWORK, EST, KASE )
   30    CONTINUE
            CALL STRSV( 'Upper', 'No trans', 'Non unit', P, B( 1, N-P+1 ),
     $                  LDB, WORK( P+MN+N+1 ), 1 )
            CALL SGEMV( 'No trans', N-P, P, -ONE, A( 1, N-P+1 ), LDA,
     $                  WORK( P+MN+N+1 ), 1, ZERO, WORK( P+MN+P+1 ), 1 )
            CALL STRSV( 'Upper', 'No transpose', 'Non unit', N-P, A, LDA,
     $                  WORK( P+MN+P+1 ), 1 )
            DO I = 1, P
               WORK( P+MN+I ) = WORK( P+MN+N+I )
            END DO
            CALL SLACON( N, WORK( P+MN+N+1 ), WORK( P+MN+1 ), IWORK, EST, KASE )
*
            IF( KASE.EQ.0 ) GOTO 40
            DO I = 1, P
               WORK( P+MN+N+I ) = WORK( MN+N+I )
            END DO
            CALL STRSV( 'Upper', 'Trans', 'Non unit', N-P, A, LDA,
```

```
     $                  WORK( P+MN+1 ), 1 )
            CALL SGEMV( 'Trans', N-P, P, -ONE, A( 1, N-P+1 ), LDA,
     $                  WORK( P+MN+1 ), 1, ONE, WORK( P+MN+N+1 ), 1 )
            CALL STRSV( 'Upper', 'Trans', 'Non unit', P, B( 1, N-P+1 ),
     $                  LDB, WORK( P+MN+N+1 ), 1 )
            CALL SLACON( P, WORK( P+MN+1 ), WORK( P+MN+N+1 ), IWORK, EST, KASE )
*
            IF( KASE.EQ.0 ) GOTO 40
         GOTO 30
   40    CONTINUE
         BAPSNM = EST
*
      END IF
*     Estimate norm of A*B^+_A
      IF( P+M.EQ.N ) THEN
         EST = ZERO
      ELSE
         R22RS = MIN( P, M-N+P )
         KASE = 0
         CALL SLACON( P, WORK( P+MN+P+1 ), WORK( P+MN+1 ), IWORK, EST, KASE )
   50    CONTINUE
            CALL STRSV( 'Upper', 'No trans', 'Non unit', P, B( 1, N-P+1 ),
     $                  LDB, WORK( P+MN+1 ), 1 )
            DO I = 1, R22RS
               WORK( P+MN+P+I ) = WORK( P+MN+I )
            END DO
            CALL STRMV( 'Upper', 'No trans', 'Non unit', R22RS,
     $                  A( N-P+1, N-P+1 ), LDA, WORK( P+MN+P+1 ), 1 )
            IF( M.LT.N ) THEN
               CALL SGEMV( 'No trans', R22RS, N-M, ONE, A( N-P+1, M+1 ), LDA,
     $                     WORK( P+MN+R22RS+1 ), 1, ONE, WORK( P+MN+P+1 ), 1 )
            END IF
            CALL SLACON( R22RS, WORK( P+MN+1 ), WORK( P+MN+P+1 ), IWORK, EST,
     $                   KASE  )
*
            IF( KASE.EQ.0 ) GOTO 60
            DO I = 1, R22RS
               WORK( P+MN+I ) = WORK( P+MN+P+I )
            END DO
            CALL STRMV( 'Upper', 'Trans', 'Non Unit', R22RS,
     $                  A( N-P+1, N-P+1 ), LDA, WORK( P+MN+1 ), 1 )
            IF( M.LT.N ) THEN
               CALL SGEMV( 'Trans', R22RS, N-M, ONE, A( N-P+1, M+1 ), LDA,
     $                     WORK( P+MN+P+1 ), 1, ZERO, WORK( P+MN+R22RS+1 ), 1 )
            END IF
            CALL STRSV( 'Upper', 'Trans', 'Non unit', P, B( 1, N-P+1 ), LDB,
     $                  WORK( P+MN+1 ), 1 )
            CALL SLACON( P, WORK( P+MN+P+1 ), WORK( P+MN+1 ), IWORK, EST, KASE )
*
            IF( KASE.EQ.0 ) GOTO 60
            GOTO 50
```

```
   60    CONTINUE
      END IF
      ABAPSN = EST
*     Get the 2-norm of Xc
      XNORM = SNRM2( N, Xc, 1 )
      IF( APPSNM.EQ.0.0E0 ) THEN
*        B is square and nonsingular
         ERRBD = EPSMCH*BNORM*BAPSNM
      ELSE
*        Get the 2-norm of the residual A*Xc - C
         RNORM = SNRM2( M-N+P, C( N-P+1 ), 1 )
*        Get the 2-norm of Xc
         XNORM = SNRM2( N, Xc, 1 )
*        Get the condition numbers
         CNDBA = BNORM*BAPSNM
         CNDAB = ANORM*APPSNM
*        Get the approximate error bound
         ERRBD = EPSMCH*( (1.0E0 + CNORM/(ANORM*XNORM))*CNDAB +
     $                RNORM/(ANORM*XNORM)*(1.0E0 + BNORM*ABAPSN/ANORM)*
     $                (CNDAB*CNDAB) + 2.0E0*CNDBA )
      END IF
```

For example, if SLAMCH('E') $= 2^{-24} = 5.961 \cdot 10^{-8}$,

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 3 & 1 & 1 \\ 1 & -1 & 3 & 1 \\ 1 & 1 & 1 & 3 \\ 1 & 1 & 1 & -1 \end{pmatrix}, b = \begin{pmatrix} 2 \\ 1 \\ 6 \\ 3 \\ 1 \end{pmatrix}, B = \begin{pmatrix} 1 & 1 & 1 & -1 \\ 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 \end{pmatrix} \quad \text{and} \quad d = \begin{pmatrix} 1 \\ 3 \\ -1 \end{pmatrix},$$

then (to 7 decimal places),

$$\hat{x} = \begin{pmatrix} 0.5000000 \\ -0.5000001 \\ 1.4999999 \\ 0.4999998 \end{pmatrix}.$$

The computed error bound ERRBD $= 5.7 \cdot 10^{-7}$, where CNDAB $= 2.09$, CNDBA $= 3.12$. The true error $= 1.2 \cdot 10^{-7}$. The exact solution is $x = [0.5, -0.5, 1.5, 0.5]^T$.

4.6.1.1 Further Details: Error Bounds for Linear Equality Constrained Least Squares Problems

In this subsection, we will summarize the available error bound. The reader may also refer to [2, 13, 18, 50] for further details.

Let $\hat{x}$ be the solution computed by the driver xGGLSE (see subsection 4.6). It is normwise stable in a mixed forward/backward sense [18, 13]. Specifically, $\hat{x} = \bar{x} + \Delta\bar{x}$, where $\bar{x}$ solves $\min\{\|b + \Delta b - (A + \Delta A)x\|_2 : (B + \Delta B)x = d\}$, and

$$\begin{aligned} \|\Delta\bar{x}\|_2 &\leq q(m,n,p)\epsilon\|\bar{x}\|_2, & \|\Delta b\|_2 &\leq q(m,n,p)\epsilon\|b\|_2, \\ \|\Delta A\|_F &\leq q(m,n,p)\epsilon\|A\|_F, & \|\Delta B\|_F &\leq q(m,n,p)\epsilon\|B\|_F, \end{aligned}$$

$q(m,n,p)$ is a modestly growing function of m, n, and p. We take $q(m,n,p) = 1$ in the code fragment above. Let $X^\dagger$ denote the Moore-Penrose pseudo-inverse of X. Let $\kappa_B(A) = \|A\|_F\|(AP)^\dagger\|_2($ = **CNDAB** above) and $\kappa_A(B) = \|B\|_F\|B_A^\dagger\|_2($ = **CNDBA** above) where $P = I - B^\dagger B$ and $B_A^\dagger = (I - (AP)^\dagger A)B^\dagger$. When $q(m,n,p)\epsilon$ is small, the error $x - \hat{x}$ is bounded by

$$\frac{\|x-\hat{x}\|}{\|x\|} \lesssim q(m,n,p)\epsilon\left\{\kappa_A(B) + \kappa_B(A)\left(\frac{\|b\|_2}{\|A\|_F\|x\|_2} + 1\right) + \kappa_B^2(A)\left(\frac{\|B\|_F}{\|A\|_F}\|AB_A^\dagger\|_2 + 1\right)\frac{\|r\|_2}{\|A\|_F\|x\|_2}\right\}.$$

When $B = 0$ and $d = 0$, we essentially recover error bounds for the linear least squares (LS) problem:

$$\frac{\|x-\widehat{x}\|_2}{\|x\|_2} \leq q(m,n)\epsilon\left\{\kappa(A)\left(\frac{\|b\|_2}{\|A\|_F\|x\|_2} + 1 + \kappa(A)\frac{\|r\|_2}{\|A\|_F\|x\|_2}\right)\right\},$$

where $\kappa(A) = \|A\|_F\|A^\dagger\|_2$. Note that the error in the standard least squares problem provided in section 4.5.1 is

$$\frac{\|x-\widehat{x}\|_2}{\|x\|_2} \lesssim p(n)\epsilon\left\{\frac{2\kappa(A)}{\cos(\theta)} + \tan(\theta)\kappa^2(A)\right\} = p(n)\epsilon\left\{\frac{2\kappa(A)\|b\|_2}{\|Ax\|_2} + \kappa^2(A)\frac{\|r\|_2}{\|Ax\|_2}\right\}$$

since $\sin(\theta) = \frac{\|A\widehat{x}-b\|_2}{\|b\|_2}$. If one assumes that $q(m,n) = p(n) = 1$, then the bounds are essentially the same.

4.6.2 General Linear Model Problem

The general linear model (GLM) problem is

$$\min_{x,y} \|y\| \quad \text{subject to} \quad d = Ax + By$$

where A is an n-by-m matrix, B is an n-by-p matrix, and d is a given n-vector, with $m \leq n \leq m+p$.

The GLM problem is solved by the driver routine xGGGLM (see section 4.6). Let $\widehat{x}$ and $\widehat{y}$ be the computed values of x and y, respectively. The approximate error bounds

$$\begin{aligned} \frac{\|x-\widehat{x}\|_2}{\|x\|_2} &\lesssim \textbf{XERRBD}, \\ \frac{\|y-\widehat{y}\|_2}{\|y\|_2} &\lesssim \textbf{YERRBD} \end{aligned}$$

can be computed by the following code fragment

```
      EPSMCH = SLAMCH( 'E' )
*     Compute the 2-norm of the left hand side D
      DNORM = SNRM2( N, D, 1 )
*     Solve the generalized linear model problem
      CALL SGGGLM( N, M, P, A, LDA, B, LDB, D, Xc, Yc, WORK,
     $             LWORK, IWORK, INFO )
*     Compute the F-norm of A and B
      ANORM = SLANTR( 'F', 'U', 'N', M, M, A, LDA, WORK( M+NP+1 ) )
      BNORM = SLANTR( 'F', 'U', 'N', N, P, B( 1, MAX( 1, P-N+1 ) ),
     $                LDB, WORK( M+NP+1 ) )
*     Compute the 2-norm of Xc
      XNORM = SNRM2( M, Xc, 1 )
*     Condition estimation
      IF( N.EQ.M ) THEN
         PBPSNM = ZERO
         TNORM = SLANTR( '1', 'U', 'N', N, N, A, LDA, WORK( M+NP+M+1 ) )
         CALL STRCON( '1', 'U', 'N', N, A, LDA, RCOND, WORK( M+NP+M+1 ),
     $                IWORK, INFO )
         ABPSNM = ONE / (RCOND * TNORM )
      ELSE
*        Compute norm of (PB)^+
         TNORM = SLANTR( '1', 'U', 'N', N-M, N-M, B( M+1, P-N+M+1 ), LDB,
     $                   WORK( M+NP+1 ) )
         CALL STRCON( '1', 'U', 'N', N-M, B( M+1, P-N+M+1 ), LDB, RCOND,
     $                WORK( M+NP+1 ), IWORK, INFO )
         PBPSNM = ONE / (RCOND * TNORM )
*        Estimate norm of A^+_B
         KASE = 0
         CALL SLACON( N, WORK( M+NP+1 ), WORK( M+NP+N+1 ), IWORK, EST, KASE )
   30    CONTINUE
            CALL STRSV( 'Upper', 'No transpose', 'Non unit', N-M,
     $                  B( M+1, P-N+M+1 ), LDB, WORK( M+NP+N+M+1 ), 1 )
            CALL SGEMV( 'No transpose', M, N-M, -ONE, B( 1, P-N+M+1 ),
     $                  LDB, WORK( M+NP+N+M+1 ), 1, ONE,
     $                  WORK( M+NP+N+1 ), 1 )
            CALL STRSV( 'Upper', 'No transpose', 'Non unit', M, A, LDA,
     $                  WORK( M+NP+N+1 ), 1 )
            DO I = 1, P
               WORK( M+NP+I ) = WORK( M+NP+N+I )
            END DO
            CALL SLACON( M, WORK( M+NP+N+1 ), WORK( M+NP+1 ), IWORK, EST, KASE )
            IF( KASE.EQ.0 ) GOTO 40
            CALL STRSV( 'Upper', 'Transpose', 'Non unit', M, A, LDA,
     $                  WORK( M+NP+1 ), 1 )
            CALL SGEMV( 'Transpose', M, N-M, -ONE, B( 1, P-N+M+1 ), LDB,
     $                  WORK( M+NP+1 ), 1, ZERO, WORK( M+NP+M+1 ), 1 )
            CALL STRSV( 'Upper', 'Transpose', 'Non unit', N-M,
     $                  B( M+1, P-N+M+1 ), LDB, WORK( M+NP+M+1 ), 1 )
            DO I = 1, N
               WORK( M+NP+N+I ) = WORK( M+NP+I )
            END DO
```

```
            CALL SLACON( N, WORK( M+NP+1 ), WORK( M+NP+N+1 ), IWORK, EST, KASE )
            IF( KASE.EQ.0 ) GOTO 40
         GOTO 30
   40    CONTINUE
         ABPSNM = EST
      END IF
*     Estimate norm of (A^+_B)*B
      IF( P+M.EQ.N ) THEN
         EST = ZERO
      ELSE
         KASE = 0
         CALL SLACON( P-N+M, WORK( M+NP+1 ), WORK( M+NP+M+1 ), IWORK, EST, KASE )
   50    CONTINUE
*
            IF( P.GE.N ) THEN
               CALL STRMV( 'Upper', 'No trans', 'Non Unit', M,
     $                     B( 1, P-N+1 ), LDB, WORK( M+NP+M+P-N+1 ), 1 )
               DO I = 1, M
                  WORK( M+NP+I ) = WORK( M+NP+M+P-N+I )
               END DO
            ELSE
               CALL SGEMV( 'No transpose', N-P, P-N+M, ONE, B, LDB,
     $                 WORK( M+NP+M+1 ), 1, ZERO, WORK( M+NP+1 ), 1 )
               CALL STRMV( 'Upper', 'No trans', 'Non Unit', P-N+M,
     $                     B( N-P+1, 1 ), LDB, WORK( M+NP+M+1 ), 1 )
               DO I = N-P+1, M
                  WORK( M+NP+I ) = WORK( M+NP+M-N+P+I )
               END DO
            END IF
            CALL STRSV( 'Upper', 'No transpose', 'Non unit', M, A, LDA,
     $                 WORK( M+NP+1 ), 1 )
            CALL SLACON( M, WORK( M+NP+M+1 ), WORK( M+NP+1 ), IWORK, EST, KASE )
*
            IF( KASE.EQ.0 ) GOTO 60
*
            CALL STRSV( 'Upper', 'Transpose', 'Non unit', M, A, LDA,
     $                 WORK( M+NP+1 ), 1 )
            IF( P.GE.N ) THEN
               CALL STRMV( 'Upper', 'Trans', 'Non Unit', M,
     $                     B( 1, P-N+1 ), LDB, WORK( M+NP+1 ), 1 )
               DO I = 1, M
                  WORK( M+NP+M+P-N+I ) = WORK( M+NP+I )
               END DO
               DO I = 1, P-N
                  WORK( M+NP+M+I ) = ZERO
               END DO
            ELSE
               CALL STRMV( 'Upper', 'Trans', 'Non Unit', P-N+M,
     $                     B( N-P+1, 1 ), LDB, WORK( M+NP+N-P+1 ), 1 )
               DO I = 1, P-N+M
                  WORK( M+NP+M+I ) = WORK( M+NP+N-P+I )
```

```
            END DO
            CALL SGEMV( 'Transpose', N-P, P-N+M, ONE, B, LDB,
     $                  WORK( M+NP+1 ), 1, ONE, WORK( M+NP+M+1 ), 1 )
         END IF
         CALL SLACON( P-N+M, WORK( M+NP+1 ), WORK( M+NP+M+1 ), IWORK, EST,
     $                 KASE )
*
         IF( KASE.EQ.0 ) GOTO 60
      GOTO 50
   60 CONTINUE
   END IF
   ABPSBN = EST
*  Get condition numbers and approximate error bounds
   CNDAB = ANORM*ABPSNM
   CNDBA = BNORM*PBPSNM
   IF( PBPSNM.EQ.0.0E+0 ) THEN
*     Then A is square and nonsingular
      XERRBD = EPSMCH*( CNDAB*( ONE+DNORM/(ANORM*XNORM) ) )
      YERRBD = 0.0E+0
   ELSE
      XERRBD = EPSMCH*( CNDAB*( ONE+DNORM/(ANORM*XNORM) ) +
     $              2.0E0*CNDAB*CNDBA*CNDBA*DNORM/(ANORM*XNORM) +
     $              ABPSBN*ABPSBN*PBPSNM*PBPSNM*ANORM*DNORM/XNORM )
      YERRBD = EPSMCH*( ABPSBN*ANORM*PBPSNM*PBPSNM +
     $              PBPSNM*(ANORM*XNORM/DNORM + 2.0E0*CNDBA*CNDBA +
     $              ONE) + CNDBA*PBPSNM )
   END IF
```

For example, if SLAMCH('E') $= 2^{-24} = 5.961 \cdot 10^{-8}$,

$$A = \begin{pmatrix} 1 & 2 & 1 & 4 \\ 1 & 3 & 2 & 1 \\ -1 & -2 & -1 & 1 \\ -1 & 2 & -1 & 5 \\ 1 & 0 & 0 & 1 \end{pmatrix}, \quad B = \begin{pmatrix} 1 & 2 & 2 \\ -1 & 1 & -2 \\ 3 & 1 & 6 \\ 1 & -1 & 2 \\ 2 & -2 & 4 \end{pmatrix} \quad \text{and} \quad d = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Then (to 7 decimal places)

$$\widehat{x} = \begin{pmatrix} -0.5466667 \\ 0.3200001 \\ 0.7200000 \\ -0.0533334 \end{pmatrix} \quad \text{and} \quad \widehat{y} = \begin{pmatrix} 0.1333334 \\ -0.1333334 \\ 0.2666667 \end{pmatrix}.$$

The computed error bounds XERRBD $= 1.2 \cdot 10^{-5}$ and YERRBD $= 6.4 \cdot 10^{-7}$, where CNDAB $= 26.97$, CNDBA $= 1.78$, The true errors in x and y are $1.2 \cdot 10^{-7}$ and $1.3 \cdot 10^{-7}$, respectively. Note that the exact solutions are $x = \frac{1}{75}[-41, 24, 54, -4]^T$ and $y = \frac{1}{15}[2, -2, 4]^T$.

4.6.2.1 Further Details: Error Bounds for General Linear Model Problems

In this subsection, we will summarize the available error bounds. The reader may also refer to [2, 13, 50, 80] for further details.

Let $\widehat{x}$ and $\widehat{y}$ be the solutions by the driver routine xGGGLM (see subsection 4.6). Then $\widehat{x}$ is normwise backward stable and $\widehat{y}$ is stable in a mixed forward/backward sense [13]. Specifically, we have $\widehat{x}$ and $\widehat{y} = \bar{y} + \Delta\bar{y}$, where $\widehat{x}$ and $\bar{y}$ solve $\min\{\|y\|_2 : (A+\Delta A)x + (B+\Delta B)y = d + \Delta d\}$, and

$$\begin{aligned} \|\Delta\bar{y}\|_2 &\le q(m,n,p)\epsilon\|\bar{y}\|_2, & \|\Delta d\|_2 &\le q(m,n,p)\epsilon\|d\|_2, \\ \|\Delta A\|_F &\le q(m,n,p)\epsilon\|A\|_F, & \|\Delta B\|_F &\le q(m,n,p)\epsilon\|B\|_F, \end{aligned}$$

and $q(m,n,p)$ is a modestly growing function of m, n, and p. We take $q(m,n,p) = 1$ in the code fragment above. Let $X^\dagger$ denote the Moore-Penrose pseudo-inverse of X. Let $\kappa_B(A) = \|A\|_F\|(A_B^\dagger)\|_2$(= CNDAB above) and $\kappa_A(B) = \|B\|_F\|(GB)^\dagger\|_2$(= CNDBA above) where $G = I - AA^\dagger$ and $A_B^\dagger = A^\dagger[I - B(GB)^\dagger]$. When $q(m,n,p)\epsilon$ is small, the errors $x - \widehat{x}$ and $y - \widehat{y}$ are bounded by

$$\begin{aligned} \frac{\|x-\widehat{x}\|_2}{\|x\|_2} &\le q(m,n,p)\epsilon\Bigg[\kappa_B(A)\left(1 + \frac{\|d\|_2}{\|A\|_F\|x\|_2} + 2\kappa_A^2(B)\frac{\|d\|_2}{\|A\|_F\|x\|_2}\right) \\ &\quad + \|A_B^\dagger B\|_2^2\|(GB)^\dagger\|_2^2\|A\|_F\frac{\|d\|_2}{\|x\|_2}\Bigg], \end{aligned}$$

and

$$\begin{aligned} \frac{\|y-\widehat{y}\|_2}{\|d\|_2} &\le q(m,n,p)\epsilon\Bigg[\|A_B^\dagger B\|_2\|A\|_F\|(GB)^\dagger\|_2^2 \\ &\quad + \|(GB)^\dagger\|_2\left(\|A\|_F\frac{\|x\|_2}{\|d\|_2} + 2\kappa_A^2(B) + 1\right) + \|B\|_F\|(GB)^\dagger\|_2^2\Bigg]. \end{aligned}$$

When $B = I$, the GLM problem is the standard LS problem. y is the residual vector $y = d - Ax$, and we have

$$\frac{\|x-\widehat{x}\|_2}{\|x\|_2} \le q(m,n)\epsilon\kappa(A)\left(1 + \frac{\|d\|_2}{\|A\|_F\|x\|_2} + \kappa(A)\frac{\|y\|_2}{\|A\|_F\|x\|_2}\right),$$

and

$$\frac{\|y-\widehat{y}\|_2}{\|y\|_2} \le q(m,n)\epsilon\kappa(A)\frac{\|y\|_2}{\|d\|_2}.$$

where $\kappa(A) = \kappa_B(A) = \|A\|_F\|A^\dagger\|_2$ and $\kappa_A(B) = 1$. The error bound of $x - \widehat{x}$ is the same as in the LSE problem (see section 4.6.1.1), which is essentially the same as given in section 4.5.1. The bound on the error in $\widehat{y}$ is the same as that provided in [55, section 5.3.7].

4.7 Error Bounds for the Symmetric Eigenproblem

The eigendecomposition of an n-by-n real symmetric matrix is the factorization $A = Z\Lambda Z^T$ ($A = Z\Lambda Z^H$ in the complex Hermitian case), where Z is orthogonal (unitary) and $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$ is real and diagonal, with $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$. The λ_i are the **eigenvalues** of A and the columns z_i of Z are the **eigenvectors**. This is also often written $Az_i = \lambda_i z_i$. The eigendecomposition of a symmetric matrix is computed by the driver routines xSYEV, xSYEVX, xSYEVD, xSYEVR, xSBEV, xSBEVX, xSBEVD, xSPEV, xSPEVX, xSPEVD, xSTEV, xSTEVX, xSTEVD and xSTEVR. The complex counterparts of these routines, which compute the eigendecomposition of complex Hermitian matrices, are the driver routines xHEEV, xHEEVX, xHEEVD, xHEEVR, xHBEV, xHBEVX, xHBEVD, xHPEV, xHPEVX, and xHPEVD (see subsection 2.3.4).

The approximate error bounds[10] for the computed eigenvalues $\hat{\lambda}_1 \leq \cdots \leq \hat{\lambda}_n$ are

$$|\hat{\lambda}_i - \lambda_i| \leq \texttt{EERRBD} \ .$$

The approximate error bounds for the computed eigenvectors $\hat{z}_i$, which bound the acute angles between the computed eigenvectors and true eigenvectors z_i, are:

$$\theta(\hat{z}_i, z_i) \leq \texttt{ZERRBD}(i) \ .$$

These bounds can be computed by the following code fragment:

```
      EPSMCH = SLAMCH( 'E' )
*     Compute eigenvalues and eigenvectors of A
*     The eigenvalues are returned in W
*     The eigenvector matrix Z overwrites A
      CALL SSYEV( 'V', UPLO, N, A, LDA, W, WORK, LWORK, INFO )
      IF( INFO.GT.0 ) THEN
         PRINT *,'SSYEV did not converge'
      ELSE IF ( N.GT.0 ) THEN
*        Compute the norm of A
         ANORM = MAX( ABS( W(1) ), ABS( W(N) ) )
         EERRBD = EPSMCH * ANORM
*        Compute reciprocal condition numbers for eigenvectors
         CALL SDISNA( 'Eigenvectors', N, N, W, RCONDZ, INFO )
         DO 10 I = 1, N
            ZERRBD( I ) = EPSMCH * ( ANORM / RCONDZ( I ) )
10       CONTINUE
      ENDIF
```

For example[11], if $\texttt{SLAMCH}('\texttt{E}') = 2^{-24} = 5.961 \cdot 10^{-8}$ and

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

then the eigenvalues, approximate error bounds, and true errors are

i	$\hat{\lambda}_i$	EERRBD	true $\lvert\hat{\lambda}_i - \lambda_i\rvert$	ZERRBD(i)	true $\theta(\hat{z}_i, z_i)$
1	$-.5157$	$6.7 \cdot 10^{-7}$	$1.6 \cdot 10^{-7}$	$9.8 \cdot 10^{-7}$	$1.2 \cdot 10^{-7}$
2	.1709	$6.7 \cdot 10^{-7}$	$3.2 \cdot 10^{-7}$	$9.8 \cdot 10^{-7}$	$7.0 \cdot 10^{-8}$
3	11.34	$6.7 \cdot 10^{-7}$	$2.8 \cdot 10^{-6}$	$6.1 \cdot 10^{-8}$	$9.7 \cdot 10^{-8}$

4.7.1 Further Details: Error Bounds for the Symmetric Eigenproblem

The usual error analysis of the symmetric eigenproblem (using any LAPACK routine in subsection 2.3.4 or any EISPACK routine) is as follows [85]:

The computed eigendecomposition $\hat{Z}\hat{\Lambda}\hat{Z}^T$ is nearly the exact eigendecomposition of $A+E$, i.e., $A+E = (\hat{Z}+\delta\hat{Z})\hat{\Lambda}(\hat{Z}+\delta\hat{Z})^T$ is a true eigendecomposition so that $\hat{Z}+\delta\hat{Z}$ is orthogonal, where $\|E\|_2/\|A\|_2 \le p(n)\epsilon$ and $\|\delta\hat{Z}\|_2 \le p(n)\epsilon$. Here $p(n)$ is a modestly growing function of n. We take $p(n) = 1$ in the above code fragment. Each computed eigenvalue $\hat{\lambda}_i$ differs from a true λ_i by at most

$$|\hat{\lambda}_i - \lambda_i| \le p(n) \cdot \epsilon \cdot \|A\|_2 = \texttt{EERRBD}\,.$$

Thus large eigenvalues (those near $\max_i |\lambda_i| = \|A\|_2$) are computed to high relative accuracy and small ones may not be.

There are two questions to ask about the computed eigenvectors: "Are they orthogonal?" and "How much do they differ from the true eigenvectors?" The answer to the first question is that except for eigenvectors computed by computational routine xSTEIN (which is called by drivers with names ending in -EVX when only a subset of the eigenvalues and eigenvectors are requested), the computed eigenvectors are always nearly orthogonal to working precision, independent of how much they differ from the true eigenvectors. In other words

$$|\hat{z}_i^T \hat{z}_j| = O(\epsilon)$$

for $i \ne j$. xSTEIN almost always returns orthogonal eigenvectors, but can occasionally fail when eigenvalues are tightly clustered.

Here is the answer to the second question about eigenvectors. The angular difference between the computed unit eigenvector $\hat{z}_i$ and a true unit eigenvector z_i satisfies the approximate bound

$$\theta(\hat{z}_i, z_i) \lesssim \frac{p(n)\epsilon\|A\|_2}{\text{gap}_i} = \texttt{ZERRBD}(i)$$

if $p(n)\epsilon$ is small enough. Here $\text{gap}_i = \min_{j\ne i} |\lambda_i - \lambda_j|$ is the **absolute gap** between λ_i and the nearest other eigenvalue. Thus, if λ_i is close to other eigenvalues, its corresponding eigenvector z_i may be inaccurate. The gaps may be easily computed from the array of computed eigenvalues using subroutine `SDISNA`. The gaps computed by `SDISNA` are ensured not to be so small as to cause overflow when used as divisors.

Let $\hat{\mathcal{S}}$ be the invariant subspace spanned by a collection of eigenvectors $\{\hat{z}_i\,,\, i \in \mathcal{I}\}$, where $\mathcal{I}$ is a subset of the integers from 1 to n. Let $\mathcal{S}$ be the corresponding true

subspace. Then

$$\theta(\hat{\mathcal{S}}, \mathcal{S}) \lesssim \frac{p(n)\epsilon\|A\|_2}{\text{gap}_{\mathcal{I}}}$$

where

$$\text{gap}_{\mathcal{I}} = \min_{\substack{i\in\mathcal{I} \\ j\notin\mathcal{I}}} |\lambda_i - \lambda_j|$$

is the absolute gap between the eigenvalues in $\mathcal{I}$ and the nearest other eigenvalue. Thus, a cluster of close eigenvalues which is far away from any other eigenvalue may have a well determined invariant subspace $\hat{\mathcal{S}}$ even if its individual eigenvectors are ill-conditioned[14].

In the special case of a real symmetric tridiagonal matrix T, the eigenvalues and eigenvectors can be computed much more accurately. xSYEV (and the other symmetric eigenproblem drivers) computes the eigenvalues and eigenvectors of a dense symmetric matrix by first reducing it to tridiagonal form T, and then finding the eigenvalues and eigenvectors of T. Reduction of a dense matrix to tridiagonal form T can introduce additional errors, so the following bounds for the tridiagonal case do not apply to the dense case.

The eigenvalues of T may be computed with small componentwise relative backward error $(O(\epsilon))$ by using subroutine xSTEBZ (subsection 2.4.4) or driver xSTEVX (subsection 2.3.4). If T is also positive definite, they may also be computed at least as accurately by xPTEQR or xSTEGR (subsection 2.4.4). To compute error bounds for the computed eigenvalues $\hat{\lambda}_i$ we must make some assumptions about T. The bounds discussed here are from [14]. Suppose T is positive definite, and write $T = DHD$ where $D = \text{diag}(t_{11}^{1/2}, \ldots, t_{nn}^{1/2})$ and $h_{ii} = 1$. Then the computed eigenvalues $\hat{\lambda}_i$ can differ from true eigenvalues λ_i by

$$|\hat{\lambda}_i - \lambda_i| \leq p(n) \cdot \epsilon \cdot \kappa_2(H) \cdot \lambda_i$$

where $p(n)$ is a modestly growing function of n. Thus if $\kappa_2(H)$ is moderate, each eigenvalue will be computed to high relative accuracy, no matter how tiny it is. The eigenvectors $\hat{z}_i$ computed by xPTEQR can differ from true eigenvectors z_i by at most about

$$\theta(\hat{z}_i, z_i) \lesssim \frac{p(n) \cdot \epsilon \cdot \kappa_2(H)}{\text{relgap}_i}$$

if $p(n)\epsilon$ is small enough, where $\text{relgap}_i = \min_{j\neq i} |\lambda_i - \lambda_j|/(\lambda_i + \lambda_j)$ is the **relative gap** between λ_i and the nearest other eigenvalue. Since the relative gap may be much larger than the absolute gap, this error bound may be much smaller than the previous one. Independent of the difference between the computed and true eigenvectors, the computed eigenvectors are orthogonal to nearly full precision, i.e. $|\hat{z}_i^T \hat{z}_j| = O(\epsilon)$ for $i \neq j$.

$\kappa_2(H)$ could be computed by applying xPTCON (subsection 2.4.1) to H. The relative gaps are easily computed from the array of computed eigenvalues.

[14]These bounds are special cases of those in section 4.8.

4.8 Error Bounds for the Nonsymmetric Eigenproblem

The nonsymmetric eigenvalue problem is more complicated than the symmetric eigenvalue problem. In this subsection, we state the simplest bounds and leave the more complicated ones to subsequent subsections.

Let A be an n-by-n nonsymmetric matrix, with eigenvalues $\lambda_1, \ldots, \lambda_n$. Let v_i be a right eigenvector corresponding to λ_i: $Av_i = \lambda_i v_i$. Let $\hat{\lambda}_i$ and $\hat{v}_i$ be the corresponding computed eigenvalues and eigenvectors, computed by expert driver routine xGEEVX (see subsection 2.3.4).

The approximate error bounds[10] for the computed eigenvalues are

$$|\hat{\lambda}_i - \lambda_i| \leq \texttt{EERRBD}(i) \ .$$

The approximate error bounds for the computed eigenvectors $\hat{v}_i$, which bound the acute angles between the computed eigenvectors and true eigenvectors v_i, are

$$\theta(\hat{v}_i, v_i) \leq \texttt{VERRBD}(i) \ .$$

These bounds can be computed by the following code fragment:

```
      EPSMCH = SLAMCH( 'E' )
*     Compute the eigenvalues and eigenvectors of A
*     WR contains the real parts of the eigenvalues
*     WI contains the real parts of the eigenvalues
*     VL contains the left eigenvectors
*     VR contains the right eigenvectors
      CALL SGEEVX( 'P', 'V', 'V', 'B', N, A, LDA, WR, WI,
     $             VL, LDVL, VR, LDVR, ILO, IHI, SCALE, ABNRM,
     $             RCONDE, RCONDV, WORK, LWORK, IWORK, INFO )
      IF( INFO.GT.0 ) THEN
         PRINT *,'SGEEVX did not converge'
      ELSE IF ( N.GT.0 ) THEN
         DO 10 I = 1, N
            EERRBD(I) = EPSMCH*ABNRM/RCONDE(I)
            VERRBD(I) = EPSMCH*ABNRM/RCONDV(I)
10       CONTINUE
      ENDIF
```

For example[11], if $\texttt{SLAMCH}('\texttt{E}') = 2^{-24} = 5.961 \cdot 10^{-8}$ and

$$A = \begin{pmatrix} 58 & 9 & 2 \\ 186 & 383 & 96 \\ -912 & -1551 & -388 \end{pmatrix}$$

then true eigenvalues, approximate eigenvalues, approximate error bounds, and true errors are

i	λ_i	$\hat{\lambda}_i$	EERRBD(i)	true $\lvert\hat{\lambda}_i - \lambda_i\rvert$	VERRBD(i)	true $\theta(\hat{v}_i, v_i)$
1	50	50.00	$8.5 \cdot 10^{-4}$	$2.7 \cdot 10^{-4}$	$2.6 \cdot 10^{-5}$	$8.5 \cdot 10^{-6}$
2	2	1.899	$3.1 \cdot 10^{-1}$	$1.0 \cdot 10^{-1}$	$1.9 \cdot 10^{-4}$	$7.7 \cdot 10^{-5}$
3	1	1.101	$3.1 \cdot 10^{-1}$	$1.0 \cdot 10^{-1}$	$1.8 \cdot 10^{-4}$	$7.5 \cdot 10^{-5}$

4.8.1 Further Details: Error Bounds for the Nonsymmetric Eigenproblem

4.8.1.1 Overview

In this subsection, we will summarize all the available error bounds. Later subsections will provide further details. The reader may also refer to [12, 95].

Bounds for individual eigenvalues and eigenvectors are provided by driver xGEEVX (subsection 2.3.4) or computational routine xTRSNA (subsection 2.4.5). Bounds for clusters of eigenvalues and their associated invariant subspace are provided by driver xGEESX (subsection 2.3.4) or computational routine xTRSEN (subsection 2.4.5).

We let $\hat{\lambda}_i$ be the ith computed eigenvalue and λ_i an ith true eigenvalue.[15] Let $\hat{v}_i$ be the corresponding computed right eigenvector, and v_i a true right eigenvector (so $Av_i = \lambda_i v_i$). If $\mathcal{I}$ is a subset of the integers from 1 to n, we let $\lambda_{\mathcal{I}}$ denote the average of the selected eigenvalues: $\lambda_{\mathcal{I}} = (\sum_{i\in\mathcal{I}} \lambda_i)/(\sum_{i\in\mathcal{I}} 1)$, and similarly for $\hat{\lambda}_{\mathcal{I}}$. We also let $\mathcal{S}_{\mathcal{I}}$ denote the subspace spanned by $\{v_i\,,\, i \in \mathcal{I}\}$; it is called a right invariant subspace because if v is any vector in $\mathcal{S}_{\mathcal{I}}$ then Av is also in $\mathcal{S}_{\mathcal{I}}$. $\hat{\mathcal{S}}_{\mathcal{I}}$ is the corresponding computed subspace.

The algorithms for the nonsymmetric eigenproblem are normwise backward stable: they compute the exact eigenvalues, eigenvectors and invariant subspaces of slightly perturbed matrices $A + E$, where $\|E\| \le p(n)\epsilon\|A\|$. Some of the bounds are stated in terms of $\|E\|_2$ and others in terms of $\|E\|_F$; one may use $p(n)\epsilon\|A\|_1$ to approximate either quantity. The code fragment in the previous subsection approximates $\|E\|$ by $\epsilon \cdot$ **ABNRM**, where **ABNRM** $= \|A\|_1$ is returned by xGEEVX.

xGEEVX (or xTRSNA) returns two quantities for each $\hat{\lambda}_i$, $\hat{v}_i$ pair: s_i and sep_i. xGEESX (or xTRSEN) returns two quantities for a selected subset $\mathcal{I}$ of eigenvalues: $s_{\mathcal{I}}$ and $\text{sep}_{\mathcal{I}}$. s_i (or $s_{\mathcal{I}}$) is a reciprocal condition number for the computed eigenvalue $\hat{\lambda}_i$ (or $\hat{\lambda}_{\mathcal{I}}$), and is referred to as **RCONDE** by xGEEVX (or xGEESX). sep_i (or $\text{sep}_{\mathcal{I}}$) is a reciprocal condition number for the right eigenvector $\hat{v}_i$ (or $\mathcal{S}_{\mathcal{I}}$), and is referred to as **RCONDV** by xGEEVX (or xGEESX). The approximate error bounds for eigenvalues, averages of eigenvalues, eigenvectors, and invariant subspaces provided in Table 4.5 are true for sufficiently small $\|E\|$, which is why they are called asymptotic.

If the problem is ill-conditioned, the asymptotic bounds may only hold for extremely small $\|E\|$.

[15] Although such a one-to-one correspondence between computed and true eigenvalues exists, it is not as simple to describe as in the symmetric case. In the symmetric case the eigenvalues are real and simply sorting provides the one-to-one correspondence, so that $\hat{\lambda}_1 \le \cdots \le \hat{\lambda}_n$ and $\lambda_1 \le \cdots \le \lambda_n$. With nonsymmetric matrices $\hat{\lambda}_i$ is usually just the computed eigenvalue closest to λ_i, but in very ill-conditioned problems this is not always true. In the most general case, the one-to-one correspondence may be described in the following nonconstructive way: Let λ_i be the eigenvalues of A and $\hat{\lambda}_i$ be the eigenvalues of $A + E$. Let $\lambda_i(t)$ be the eigenvalues of $A + tE$, where t is a parameter, which is initially zero, so that we may set $\lambda_i(0) = \lambda_i$. As t increase from 0 to 1, $\lambda_i(t)$ traces out a curve from λ_i to $\hat{\lambda}_i$, providing the correspondence. Care must be taken when the curves intersect, and the correspondence may not be unique.

Table 4.5: Asymptotic error bounds for the nonsymmetric eigenproblem

Simple eigenvalue	$\lvert\hat{\lambda}_i - \lambda_i\rvert \lesssim \lVert E\rVert_2/s_i$
Eigenvalue cluster	$\lvert\hat{\lambda}_{\mathcal{I}} - \lambda_{\mathcal{I}}\rvert \lesssim \lVert E\rVert_2/s_{\mathcal{I}}$
Eigenvector	$\theta(\hat{v}_i, v_i) \lesssim \lVert E\rVert_F/\mathrm{sep}_i$
Invariant subspace	$\theta(\hat{\mathcal{S}}_{\mathcal{I}}, \mathcal{S}_{\mathcal{I}}) \lesssim \lVert E\rVert_F/\mathrm{sep}_{\mathcal{I}}$

Table 4.6: Global error bounds for the nonsymmetric eigenproblem assuming $\lVert E\rVert_F < s_i \cdot \mathrm{sep}_i/4$

Eigenvalue cluster	$\lvert\hat{\lambda}_{\mathcal{I}} - \lambda_{\mathcal{I}}\rvert \le 2\lVert E\rVert_2/s_{\mathcal{I}}$
Eigenvector	$\theta(\hat{v}_i, v_i) \le \arctan(2\lVert E\rVert_F/(\mathrm{sep}_i - 4\lVert E\rVert_F/s_i))$
Invariant subspace	$\theta(\hat{\mathcal{S}}_{\mathcal{I}}, \mathcal{S}_{\mathcal{I}}) \le \arctan(2\lVert E\rVert_F/(\mathrm{sep}_{\mathcal{I}} - 4\lVert E\rVert_F/s_{\mathcal{I}}))$

Therefore, in Table 4.6 we also provide global bounds which are guaranteed to hold for all $\lVert E\rVert_F < s \cdot \mathrm{sep}/4$.

We also have the following bound, which is true for all E: all the λ_i lie in the union of n disks, where the i-th disk is centered at $\hat{\lambda}_i$ and has radius $n\lVert E\rVert_2/s_i$. If k of these disks overlap, so that any two points inside the k disks can be connected by a continuous curve lying entirely inside the k disks, and if no larger set of $k+1$ disks has this property, then exactly k of the λ_i lie inside the union of these k disks. Figure 4.1 illustrates this for a 10-by-10 matrix, with 4 such overlapping unions of disks, two containing 1 eigenvalue each, one containing 2 eigenvalues, and one containing 6 eigenvalues.

Finally, the quantities s and sep tell use how we can best (block) diagonalize a matrix A by a similarity, $V^{-1}AV = \mathrm{diag}(A_{11}, \ldots, A_{bb})$, where each diagonal block A_{ii} has a selected subset of the eigenvalues of A. Such a decomposition may be useful in computing functions of matrices, for example. The goal is to choose a V with a nearly minimum condition number $\kappa_2(V)$ which performs this decomposition, since this generally minimizes the error in the decomposition. This may be done as follows. Let A_{ii} be n_i-by-n_i. Then columns $1 + \sum_{j=1}^{i-1} n_j$ through $\sum_{j=1}^{i} n_j$ of V span the invariant subspace of A corresponding to the eigenvalues of A_{ii}; these columns should be chosen to be any orthonormal basis of this space (as computed by xGEESX, for example). Let $s_{\mathcal{I}_i}$ be the value corresponding to the cluster of eigenvalues of A_{ii}, as computed by xGEESX or xTRSEN. Then $\kappa_2(V) \le b/\min_i(s_{\mathcal{I}_i})$, and no other choice of V can make its condition number smaller than $1/\min_i(s_{\mathcal{I}_i})$ [26]. Thus choosing orthonormal subblocks of V gets $\kappa_2(V)$ to within a factor b of its minimum value.

In the case of a real symmetric (or complex Hermitian) matrix, $s = 1$ and sep is the absolute gap, as defined in subsection 4.7. The bounds in Table 4.5 then reduce to the bounds in subsection 4.7.

Figure 4.1: Bounding eigenvalues inside overlapping disks

4.8.1.2 Balancing and Conditioning

There are two preprocessing steps one may perform on a matrix A in order to make its eigenproblem easier. The first is **permutation**, or reordering the rows and columns to make A more nearly upper triangular (closer to Schur form): $A' = PAP^T$, where P is a permutation matrix. If A' is permutable to upper triangular form (or close to it), then no floating-point operations (or very few) are needed to reduce it to Schur form. The second is **scaling** by a diagonal matrix D to make the rows and columns of A' more nearly equal in norm: $A'' = DA'D^{-1}$. Scaling can make the matrix norm smaller with respect to the eigenvalues, and so possibly reduce the inaccuracy contributed by roundoff [106, Chap. II/11]. We refer to these two operations as **balancing**.

Balancing is performed by driver xGEEVX, which calls computational routine xGEBAL. The user may tell xGEEVX to optionally permute, scale, do both, or do neither; this is specified by input parameter `BALANC`. Permuting has no effect on the condition numbers or their interpretation as described in previous subsections. Scaling, however, does change their interpretation, as we now describe.

The output parameters of xGEEVX — `SCALE` (real array of length N), `ILO` (integer), `IHI` (integer) and `ABNRM` (real) — describe the result of balancing a matrix A into A'', where N is the dimension of A. The matrix A'' is block upper triangular, with at most three blocks: from 1 to `ILO`-1, from `ILO` to `IHI`, and from `IHI`$+1$ to N. The first and last blocks are upper triangular, and so already in Schur form. These are not scaled; only the block from `ILO` to `IHI` is scaled. Details of the scaling and permutation are described in `SCALE` (see the specification of xGEEVX or xGEBAL for details). The one-norm of A'' is returned in `ABNRM`.

The condition numbers described in earlier subsections are computed for the balanced matrix A'', and so some interpretation is needed to apply them to the eigenvalues and eigenvectors of the original matrix A. To use the bounds for eigenvalues in Tables 4.5 and 4.6, we must replace $\|E\|_2$ and $\|E\|_F$ by $O(\epsilon)\|A''\| = O(\epsilon) \cdot$ `ABNRM`. To use the bounds for eigenvectors, we also need to take into account that bounds on rotations of eigenvectors are for the eigenvectors x'' of A'', which are related to the eigenvectors x of A by $DPx = x''$, or $x = P^T D^{-1} x''$. One coarse but simple way to do this is as follows: let θ'' be the bound on rotations of x'' from Table 4.5 or Table 4.6 and let θ be the desired bound on rotation of x. Let

$$\kappa(D) = \frac{\max_{\mathtt{ILO} \le i \le \mathtt{IHI}} \mathtt{SCALE}(i)}{\min_{\mathtt{ILO} \le i \le \mathtt{IHI}} \mathtt{SCALE}(i)}$$

be the condition number of D. Then

$$\sin\theta \le \kappa(D) \cdot \sin\theta'' \ .$$

The numerical example in subsection 4.8 does no scaling, just permutation.

4.8.1.3 Computing s and sep

To explain s and sep, we need to introduce the **spectral projector** P [94, 76], and the **separation of two matrices** A and B, $\text{sep}(A, B)$ [94, 98].

We may assume the matrix A is in Schur form, because reducing it to this form does not change the values of s and sep. Consider a cluster of $m \geq 1$ eigenvalues, counting multiplicities. Further assume the n-by-n matrix A is

$$A = \begin{pmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{pmatrix} \tag{4.1}$$

where the eigenvalues of the m-by-m matrix A_{11} are exactly those in which we are interested. In practice, if the eigenvalues on the diagonal of A are in the wrong order, routine xTREXC can be used to put the desired ones in the upper left corner as shown.

We define the **spectral projector**, or simply projector P belonging to the eigenvalues of A_{11} as

$$P = \begin{pmatrix} I_m & R \\ 0 & 0 \end{pmatrix} \tag{4.2}$$

where R satisfies the system of linear equations

$$A_{11}R - RA_{22} = A_{12}. \tag{4.3}$$

Equation (4.3) is called a Sylvester equation. Given the Schur form (4.1), we solve equation (4.3) for R using the subroutine xTRSYL.

We can now define s for the eigenvalues of A_{11}:

$$s = \frac{1}{\|P\|_2} = \frac{1}{\sqrt{1 + \|R\|_2^2}}. \tag{4.4}$$

In practice we do not use this expression since $\|R\|_2$ is hard to compute. Instead we use the more easily computed underestimate

$$\frac{1}{\sqrt{1 + \|R\|_F^2}} \tag{4.5}$$

which can underestimate the true value of s by no more than a factor $\sqrt{\min(m, n-m)}$. This underestimation makes our error bounds more conservative. This approximation of s is called `RCONDE` in xGEEVX and xGEESX.

The **separation** $\text{sep}(A_{11}, A_{22})$ of the matrices A_{11} and A_{22} is defined as the smallest singular value of the linear map in (4.3) which takes X to $A_{11}X - XA_{22}$, i.e.,

$$\text{sep}(A_{11}, A_{22}) = \min_{X \neq 0} \frac{\|A_{11}X - XA_{22}\|_F}{\|X\|_F}. \tag{4.6}$$

This formulation lets us estimate $\text{sep}(A_{11}, A_{22})$ using the condition estimator xLACON [59, 62, 63], which estimates the norm of a linear operator $\|T\|_1$ given the ability to compute Tx and T^Tx

quickly for arbitrary x. In our case, multiplying an arbitrary vector by T means solving the Sylvester equation (4.3) with an arbitrary right hand side using xTRSYL, and multiplying by T^T means solving the same equation with A_{11} replaced by A_{11}^T and A_{22} replaced by A_{22}^T. Solving either equation costs at most $O(n^3)$ operations, or as few as $O(n^2)$ if $m \ll n$. Since the true value of sep is $\|T\|_2$ but we use $\|T\|_1$, our estimate of sep may differ from the true value by as much as $\sqrt{m(n-m)}$. This approximation to sep is called **RCONDV** by xGEEVX and xGEESX.

Another formulation which in principle permits an exact evaluation of $\mathrm{sep}(A_{11}, A_{22})$ is

$$\mathrm{sep}(A_{11}, A_{22}) = \sigma_{\min}(I_{n-m} \otimes A_{11} - A_{22}^T \otimes I_m) \tag{4.7}$$

where $X \otimes Y \equiv [x_{ij}Y]$ is the Kronecker product of X and Y. This method is generally impractical, however, because the matrix whose smallest singular value we need is $m(n-m)$ dimensional, which can be as large as $n^2/4$. Thus we would require as much as $O(n^4)$ extra workspace and $O(n^6)$ operations, much more than the estimation method of the last paragraph.

The expression $\mathrm{sep}(A_{11}, A_{22})$ measures the "separation" of the spectra of A_{11} and A_{22} in the following sense. It is zero if and only if A_{11} and A_{22} have a common eigenvalue, and small if there is a small perturbation of either one that makes them have a common eigenvalue. If A_{11} and A_{22} are both Hermitian matrices, then $\mathrm{sep}(A_{11}, A_{22})$ is just the gap, or minimum distance between an eigenvalue of A_{11} and an eigenvalue of A_{22}. On the other hand, if A_{11} and A_{22} are non-Hermitian, $\mathrm{sep}(A_{11}, A_{22})$ may be much smaller than this gap.

4.9 Error Bounds for the Singular Value Decomposition

The singular value decomposition (SVD) of a real m-by-n matrix A is defined as follows. Let $r = \min(m, n)$. The SVD of A is $A = U\Sigma V^T$ ($A = U\Sigma V^H$ in the complex case), where U and V are orthogonal (unitary) matrices and $\Sigma = \mathrm{diag}(\sigma_1, \ldots, \sigma_r)$ is diagonal, with $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r \geq 0$. The σ_i are the **singular values** of A and the leading r columns u_i of U and v_i of V the **left and right singular vectors,** respectively. The SVD of a general matrix is computed by xGESVD or xGESDD (see subsection 2.3.4).

The approximate error bounds[10] for the computed singular values $\hat{\sigma}_1 \geq \cdots \geq \hat{\sigma}_r$ are

$$|\hat{\sigma}_i - \sigma_i| \leq \mathtt{SERRBD} \ .$$

The approximate error bounds for the computed singular vectors $\hat{v}_i$ and $\hat{u}_i$, which bound the acute angles between the computed singular vectors and true singular vectors v_i and u_i, are

$$\begin{aligned} \theta(\hat{v}_i, v_i) &\leq \mathtt{VERRBD}(i) \\ \theta(\hat{u}_i, u_i) &\leq \mathtt{UERRBD}(i) \ . \end{aligned}$$

These bounds can be computing by the following code fragment.

```
      EPSMCH = SLAMCH( 'E' )
*     Compute singular value decomposition of A
```

```
*     The singular values are returned in S
*     The left singular vectors are returned in U
*     The transposed right singular vectors are returned in VT
      CALL  SGESVD( 'S', 'S', M, N, A, LDA, S, U, LDU, VT, LDVT,
     $              WORK, LWORK, INFO )
      IF( INFO.GT.0 ) THEN
         PRINT *,'SGESVD did not converge'
      ELSE IF ( MIN(M,N) .GT. 0 ) THEN
         SERRBD  = EPSMCH * S(1)
*        Compute reciprocal condition numbers for singular vectors
         CALL SDISNA( 'Left', M, N, S, RCONDU, INFO )
         CALL SDISNA( 'Right', M, N, S, RCONDV, INFO )
         DO 10 I = 1, MIN(M,N)
            VERRBD( I ) = EPSMCH*( S(1)/RCONDV( I ) )
            UERRBD( I ) = EPSMCH*( S(1)/RCONDU( I ) )
10       CONTINUE
      END IF
```

For example[11], if SLAMCH('E') $= 2^{-24} = 5.961 \cdot 10^{-8}$ and

$$A = \begin{pmatrix} 4 & 3 & 5 \\ 2 & 5 & 8 \\ 3 & 6 & 10 \\ 4 & 5 & 11 \end{pmatrix},$$

then the singular values, approximate error bounds, and true errors are given below.

i	$\hat{\sigma}_i$	SERRBD	true $\lvert\hat{\sigma}_i - \sigma_i\rvert$	VERRBD(i)	true $\theta(\hat{v}_i, v_i)$	UERRBD(i)	true $\theta(\hat{u}_i, u_i)$
1	21.05	$1.3 \cdot 10^{-6}$	$1.7 \cdot 10^{-6}$	$6.7 \cdot 10^{-8}$	$8.1 \cdot 10^{-8}$	$6.7 \cdot 10^{-8}$	$1.5 \cdot 10^{-7}$
2	2.370	$1.3 \cdot 10^{-6}$	$5.8 \cdot 10^{-7}$	$1.0 \cdot 10^{-6}$	$2.9 \cdot 10^{-7}$	$1.0 \cdot 10^{-6}$	$2.4 \cdot 10^{-7}$
3	1.143	$1.3 \cdot 10^{-6}$	$3.2 \cdot 10^{-7}$	$1.0 \cdot 10^{-6}$	$3.0 \cdot 10^{-7}$	$1.1 \cdot 10^{-6}$	$2.4 \cdot 10^{-7}$

4.9.1 Further Details: Error Bounds for the Singular Value Decomposition

The usual error analysis of the SVD algorithms xGESVD and xGESDD in LAPACK (see subsection 2.3.4) or the routines in LINPACK and EISPACK is as follows [25, 55]:

> The SVD algorithm is backward stable. This means that the computed SVD, $\hat{U}\hat{\Sigma}\hat{V}^T$, is nearly the exact SVD of $A+E$ where $\|E\|_2/\|A\|_2 \le p(m,n)\epsilon$, and $p(m,n)$ is a modestly growing function of m and n. This means $A + E = (\hat{U} + \delta\hat{U})\hat{\Sigma}(\hat{V} + \delta\hat{V})$ is the true SVD, so that $\hat{U} + \delta\hat{U}$ and $\hat{V} + \delta\hat{V}$ are both orthogonal, where $\|\delta\hat{U}\| \le p(m,n)\epsilon$, and $\|\delta\hat{V}\| \le p(m,n)\epsilon$. Each computed singular value $\hat{\sigma}_i$ differs from true σ_i by at most
>
> $$|\hat{\sigma}_i - \sigma_i| \le p(m,n) \cdot \epsilon \cdot \sigma_1 = \texttt{SERRBD},$$

(we take $p(m,n) = 1$ in the code fragment). Thus large singular values (those near σ_1) are computed to high relative accuracy and small ones may not be.

There are two questions to ask about the computed singular vectors: "Are they orthogonal?" and "How much do they differ from the true eigenvectors?" The answer to the first question is yes, the computed singular vectors are always nearly orthogonal to working precision, independent of how much they differ from the true singular vectors. In other words

$$|\hat{u}_i^T \hat{u}_j| = O(\epsilon)$$

for $i \neq j$.

Here is the answer to the second question about singular vectors. The angular difference between the computed left singular vector $\hat{u}_i$ and a true u_i satisfies the approximate bound

$$\theta(\hat{u}_i, u_i) \lesssim \frac{p(m,n)\epsilon\|A\|_2}{\text{gap}_i} = \texttt{UERRBD}(i)$$

where $\text{gap}_i = \min_{j \neq i} |\sigma_i - \sigma_j|$ is the **absolute gap** between σ_i and the nearest other singular value. We take $p(m,n) = 1$ in the code fragment. Thus, if σ_i is close to other singular values, its corresponding singular vector u_i may be inaccurate. When $n < m$, then gap_n must be redefined as $\min(\min_{j \neq n}(|\sigma_n - \sigma_j|, \sigma_n))$. The gaps may be easily computed from the array of computed singular values using function `SDISNA`. The gaps computed by `SDISNA` are ensured not to be so small as to cause overflow when used as divisors. The same bound applies to the computed right singular vector $\hat{v}_i$ and a true vector v_i.

Let $\hat{\mathcal{S}}$ be the space spanned by a collection of computed left singular vectors $\{\hat{u}_i\,,\, i \in \mathcal{I}\}$, where $\mathcal{I}$ is a subset of the integers from 1 to n. Let $\mathcal{S}$ be the corresponding true space. Then

$$\theta(\hat{\mathcal{S}}, \mathcal{S}) \lesssim \frac{p(m,n)\epsilon\|A\|_2}{\text{gap}_{\mathcal{I}}}.$$

where

$$\text{gap}_{\mathcal{I}} = \min_{\substack{i \in \mathcal{I} \\ j \notin \mathcal{I}}} |\sigma_i - \sigma_j|$$

is the absolute gap between the singular values in $\mathcal{I}$ and the nearest other singular value. Thus, a cluster of close singular values which is far away from any other singular value may have a well determined space $\hat{\mathcal{S}}$ even if its individual singular vectors are ill-conditioned. The same bound applies to a set of right singular vectors $\{\hat{v}_i\,,\, i \in \mathcal{I}\}$[16].

In the special case of bidiagonal matrices, the singular values and singular vectors may be computed much more accurately. A bidiagonal matrix B has nonzero entries only on the main diagonal and the diagonal immediately above it (or immediately below it). xGESVD computes the SVD of a general matrix by first reducing it to bidiagonal form B, and then calling xBDSQR (subsection 2.4.6) to compute the SVD of B. xGESDD is similar, but calls xBDSDC to compute the SVD

[16]These bounds are special cases of those in sections 4.7 and 4.8, since the singular values and vectors of A are simply related to the eigenvalues and eigenvectors of the Hermitian matrix $\begin{pmatrix} 0 & A^H \\ A & 0 \end{pmatrix}$ [55, p. 427].

of B. Reduction of a dense matrix to bidiagonal form B can introduce additional errors, so the following bounds for the bidiagonal case do not apply to the dense case.

> Each computed singular value of a bidiagonal matrix is accurate to nearly full relative accuracy, no matter how tiny it is:
>
> $$|\hat{\sigma}_i - \sigma_i| \leq p(m,n) \cdot \epsilon \cdot \sigma_i.$$
>
> The following bounds apply only to xBDSQR. The computed left singular vector $\hat{u}_i$ has an angular error at most about
>
> $$\theta(\hat{u}_i, u_i) \lesssim \frac{p(m,n)\epsilon}{\text{relgap}_i}$$
>
> where $\text{relgap}_i = \min_{j \neq i} |\sigma_i - \sigma_j|/(\sigma_i + \sigma_j)$ is the **relative gap** between σ_i and the nearest other singular value. The same bound applies to the right singular vector $\hat{v}_i$ and v_i. Since the relative gap may be much larger than the absolute gap, this error bound may be much smaller than the previous one. The relative gaps may be easily computed from the array of computed singular values.

In the very special case of 2-by-2 bidiagonal matrices, xBDSQR and xBDSDC call auxiliary routine xLASV2 to compute the SVD. xLASV2 will actually compute nearly correctly rounded singular vectors independent of the relative gap, but this requires accurate computer arithmetic: if leading digits cancel during floating-point subtraction, the resulting difference must be exact. On machines without guard digits one has the slightly weaker result that the algorithm is componentwise relatively backward stable, and therefore the accuracy of the singular vectors depends on the relative gap as described above.

Jacobi's method [34, 99, 91] is another algorithm for finding singular values and singular vectors of matrices. It is slower than the algorithms based on first bidiagonalizing the matrix, but is capable of computing more accurate answers in several important cases.

4.10 Error Bounds for the Generalized Symmetric Definite Eigenproblem

There are three types of problems to consider. In all cases A and B are real symmetric (or complex Hermitian) and B is positive definite. These decompositions are computed for real symmetric matrices by the driver routines xSYGV, xSYGVX, xSYGVD, xSPGV, xSPGVX, xSPGVD, and (for type 1 only) xSBGV, xSBGVX and xSBGVD (see subsection 2.3.5.1). These decompositions are computed for complex Hermitian matrices by the driver routines xHEGV, xHEGVX, xHEGVD, xHPGV, xHPGVX, xHPGVD, and (for type 1 only) xHBGV, xHBGVX, xHBGVD (see subsection 2.3.5.1). In each of the following three decompositions, Λ is real and diagonal with diagonal entries $\lambda_1 \leq \cdots \leq \lambda_n$, and the columns z_i of Z are linearly independent vectors. The λ_i are called **eigenvalues** and the z_i are **eigenvectors**.

1. $A - \lambda B$. The eigendecomposition may be written $Z^T AZ = \Lambda$ and $Z^T BZ = I$ (or $Z^H AZ = \Lambda$ and $Z^H BZ = I$ if A and B are complex). This may also be written $Az_i = \lambda_i Bz_i$.

2. $AB - \lambda I$. The eigendecomposition may be written $Z^{-1}AZ^{-T} = \Lambda$ and $Z^T BZ = I$ ($Z^{-1}AZ^{-H} = \Lambda$ and $Z^H BZ = I$ if A and B are complex). This may also be written $ABz_i = \lambda_i z_i$.

3. $BA - \lambda I$. The eigendecomposition may be written $Z^T AZ = \Lambda$ and $Z^T B^{-1}Z = I$ ($Z^H AZ = \Lambda$ and $Z^H B^{-1}Z = I$ if A and B are complex). This may also be written $BAz_i = \lambda_i z_i$.

The approximate error bounds[10] for the computed eigenvalues $\hat{\lambda}_1 \leq \cdots \leq \hat{\lambda}_n$ are

$$|\hat{\lambda}_i - \lambda_i| \leq \texttt{EERRBD}(i) .$$

The approximate error bounds for the computed eigenvectors $\hat{z}_i$, which bound the acute angles between the computed eigenvectors and true eigenvectors z_i, are

$$\theta(\hat{z}_i, z_i) \leq \texttt{ZERRBD}(i) .$$

These bounds are computed differently, depending on which of the above three problems are to be solved. The following code fragments show how.

1. First we consider error bounds for problem 1.

```
      EPSMCH = SLAMCH( 'E' )
*     Solve the eigenproblem A - lambda B (ITYPE = 1)
      ITYPE = 1
*     Compute the norms of A and B
      ANORM = SLANSY( '1', UPLO, N, A, LDA, WORK )
      BNORM = SLANSY( '1', UPLO, N, B, LDB, WORK )
*     The eigenvalues are returned in W
*     The eigenvectors are returned in A
      CALL SSYGV( ITYPE, 'V', UPLO, N, A, LDA, B, LDB, W, WORK,
     $            LWORK, INFO )
      IF( INFO.GT.0 .AND. INFO.LE.N ) THEN
         PRINT *,'SSYGV did not converge'
      ELSE IF( INFO.GT.N ) THEN
         PRINT *,'B not positive definite'
      ELSE IF ( N.GT.0 ) THEN
*        Get reciprocal condition number RCONDB of Cholesky factor of B
         CALL STRCON( '1', UPLO, 'N', N, B, LDB, RCONDB, WORK, IWORK,
     $                INFO )
         RCONDB = MAX( RCONDB, EPSMCH )
         CALL SDISNA( 'Eigenvectors', N, N, W, RCONDZ, INFO )
         DO 10 I = 1, N
            EERRBD( I ) = ( EPSMCH / RCONDB**2 ) * ( ANORM / BNORM +
     $                      ABS( W(I) ) )
```

```
            ZERRBD( I ) = ( EPSMCH / RCONDB**3 ) * ( ( ANORM / BNORM )
     $                    / RCONDZ(I) + ( ABS( W(I) ) / RCONDZ(I) ) *
     $                    RCONDB )
10       CONTINUE
      END IF
```

For example[11], if SLAMCH('E') $= 2^{-24} = 5.961 \cdot 10^{-8}$,

$$A = \begin{pmatrix} 100000 & 10099 & 2109 \\ 10099 & 100020 & 10012 \\ 2109 & 10112 & -48461 \end{pmatrix} \text{ and } B = \begin{pmatrix} 99 & 10 & 2 \\ 10 & 100 & 10 \\ 2 & 10 & 100 \end{pmatrix}$$

then ANORM $= 120231$, BNORM $= 120$, and RCONDB $= .8326$, and the approximate eigenvalues, approximate error bounds, and true errors are

i	λ_i	EERRBD(i)	true $\lvert\hat{\lambda}_i - \lambda_i\rvert$	ZERRBD(i)	true $\theta(\hat{v}_i, v_i)$
1	-500.0	$1.3 \cdot 10^{-4}$	$9.0 \cdot 10^{-6}$	$9.8 \cdot 10^{-8}$	$1.0 \cdot 10^{-9}$
2	1000.	$1.7 \cdot 10^{-4}$	$4.6 \cdot 10^{-5}$	$1.9 \cdot 10^{-5}$	$1.2 \cdot 10^{-7}$
3	1010.	$1.7 \cdot 10^{-4}$	$1.0 \cdot 10^{-4}$	$1.9 \cdot 10^{-5}$	$1.1 \cdot 10^{-7}$

This code fragment cannot be adapted to use xSBGV (or xHBGV), because xSBGV does not return a conventional Cholesky factor in B, but rather a "split" Cholesky factorization (performed by xPBSTF).

2. Problem types 2 and 3 have the same error bounds. We illustrate only type 2.

```
      EPSMCH = SLAMCH( 'E' )
*     Solve the eigenproblem A*B - lambda I (ITYPE = 2)
      ITYPE = 2
*     Compute the norms of A and B
      ANORM = SLANSY( '1', UPLO, N, A, LDA, WORK )
      BNORM = SLANSY( '1', UPLO, N, B, LDB, WORK )
*     The eigenvalues are returned in W
*     The eigenvectors are returned in A
      CALL SSYGV( ITYPE, 'V', UPLO, N, A, LDA, B, LDB, W, WORK,
     $            LWORK, INFO )
      IF( INFO.GT.0 .AND. INFO.LE.N ) THEN
         PRINT *,'SSYGV did not converge'
      ELSE IF( INFO.GT.N ) THEN
         PRINT *,'B not positive definite'
      ELSE IF ( N.GT.0 ) THEN
*        Get reciprocal condition number RCONDB of Cholesky factor of B
         CALL STRCON( '1', UPLO, 'N', N, B, LDB, RCONDB, WORK, IWORK,
     $                INFO )
         RCONDB = MAX( RCONDB, EPSMCH )
```

```
         CALL SDISNA( 'Eigenvectors', N, N, W, RCONDZ, INFO )
         DO 10 I = 1, N
            EERRBD(I) = ( ANORM * BNORM ) * EPSMCH +
     $                  ( EPSMCH / RCONDB**2 ) * ABS( W(I) )
            ZERRBD(I) = ( EPSMCH / RCONDB ) * ( ( ANORM * BNORM ) /
     $                    RCONDZ(I) + 1.0 / RCONDB )
10       CONTINUE
      END IF
```

For the same A and B as above, the approximate eigenvalues, approximate error bounds, and true errors are

i	λ_i	EERRBD(i)	true $\lvert\hat{\lambda}_i - \lambda_i\rvert$	ZERRBD(i)	true $\theta(\hat{v}_i, v_i)$
1	$-4.817 \cdot 10^6$	1.3	$6.0 \cdot 10^{-3}$	$1.7 \cdot 10^{-7}$	$7.0 \cdot 10^{-9}$
2	$8.094 \cdot 10^6$	1.6	1.5	$3.4 \cdot 10^{-7}$	$3.3 \cdot 10^{-8}$
3	$1.219 \cdot 10^7$	1.9	4.5	$3.4 \cdot 10^{-7}$	$4.7 \cdot 10^{-8}$

4.10.1 Further Details: Error Bounds for the Generalized Symmetric Definite Eigenproblem

The error analysis of the driver routines for the generalized symmetric definite eigenproblem goes as follows. In all cases $\text{gap}_i = \min_{j \neq i} |\lambda_i - \lambda_j|$ is the **absolute gap** between λ_i and the nearest other eigenvalue.

1. $A - \lambda B$. The computed eigenvalues $\hat{\lambda}_i$ can differ from true eigenvalues λ_i by at most about

$$|\hat{\lambda}_i - \lambda_i| \lesssim p(n)\epsilon \cdot (\|B^{-1}\|_2 \|A\|_2 + \kappa_2(B) \cdot |\hat{\lambda}_i|) = \texttt{EERRBD}(i) \ .$$

   The angular difference between the computed eigenvector $\hat{z}_i$ and a true eigenvector z_i is

$$\theta(\hat{z}_i, z_i) \lesssim p(n)\epsilon \frac{\|B^{-1}\|_2 \|A\|_2 (\kappa_2(B))^{1/2} + \kappa_2(B)|\hat{\lambda}_i|}{\text{gap}_i} = \texttt{ZERRBD}(i) \ .$$

2. $AB - \lambda I$ or $BA - \lambda I$. The computed eigenvalues $\hat{\lambda}_i$ can differ from true eigenvalues λ_i by at most about

$$|\hat{\lambda}_i - \lambda_i| \lesssim p(n)\epsilon \cdot (\|B\|_2 \|A\|_2 + \kappa_2(B) \cdot |\hat{\lambda}_i|) = \texttt{EERRBD}(i) \ .$$

   The angular difference between the computed eigenvector $\hat{z}_i$ and a true eigenvector z_i is

$$\theta(\hat{z}_i, z_i) \lesssim p(n)\epsilon \left(\frac{\|B\|_2 \|A\|_2 (\kappa_2(B))^{1/2}}{\text{gap}_i} + \kappa_2(B) \right) = \texttt{ZERRBD}(i) \ .$$

The code fragments above replace $p(n)$ by 1, and makes sure neither RCONDB nor RCONDZ is so small as to cause overflow when used as divisors in the expressions for error bounds.

These error bounds are large when B is ill-conditioned with respect to inversion ($\kappa_2(B)$ is large). It is often the case that the eigenvalues and eigenvectors are much better conditioned than indicated here. We mention three ways to get tighter bounds. The first way is effective when the diagonal entries of B differ widely in magnitude[17]:

1. $A - \lambda B$. Let $D = \mathrm{diag}(b_{11}^{-1/2}, \ldots, b_{nn}^{-1/2})$ be a diagonal matrix. Then replace B by DBD and A by DAD in the above bounds.

2. $AB - \lambda I$ or $BA - \lambda I$. Let $D = \mathrm{diag}(b_{11}^{-1/2}, \ldots, b_{nn}^{-1/2})$ be a diagonal matrix. Then replace B by DBD and A by $D^{-1}AD^{-1}$ in the above bounds.

The second way to get tighter bounds does not actually supply guaranteed bounds, but its estimates are often better in practice. It is not guaranteed because it assumes the algorithm is backward stable, which is not necessarily true when B is ill-conditioned. It estimates the **chordal distance** between a true eigenvalue λ_i and a computed eigenvalue $\hat{\lambda}_i$:

$$\chi(\hat{\lambda}_i, \lambda_i) = \frac{|\hat{\lambda}_i - \lambda_i|}{\sqrt{1+\hat{\lambda}_i^2} \cdot \sqrt{1+\lambda_i^2}}.$$

To interpret this measure we write $\lambda_i = \tan\theta$ and $\hat{\lambda}_i = \tan\hat{\theta}$. Then $\chi(\hat{\lambda}_i, \lambda_i) = |\sin(\hat{\theta} - \theta)|$. In other words, if $\hat{\lambda}_i$ represents the one-dimensional subspace $\hat{\mathcal{S}}$ consisting of the line through the origin with slope $\hat{\lambda}_i$, and λ_i represents the analogous subspace $\mathcal{S}$, then $\chi(\hat{\lambda}_i, \lambda_i)$ is the sine of the acute angle $\theta(\hat{\mathcal{S}}, \mathcal{S})$ between these subspaces. Thus χ is bounded by one, and is small when both arguments are large[18]. It applies only to the first problem, $A - \lambda B$:

> Suppose a computed eigenvalue $\hat{\lambda}_i$ of $A - \lambda B$ is the exact eigenvalue of a perturbed problem $(A+E) - \lambda(B+F)$. Let x_i be the unit eigenvector ($\|x_i\|_2 = 1$) for the exact eigenvalue λ_i. Then if $\|E\|$ is small compared to $\|A\|$, and if $\|F\|$ is small compared to $\|B\|$, we have
>
> $$\chi(\hat{\lambda}_i, \lambda_i) \lesssim \frac{\|E\| + \|F\|}{\sqrt{(x_i^H A x_i)^2 + (x_i^H B x_i)^2}}.$$
>
> Thus $1/\sqrt{(x_i^H A x_i)^2 + (x_i^H B x_i)^2}$ is a condition number for eigenvalue λ_i.

The third way applies only to the first problem $A - \lambda B$, and only when A is positive definite. We use a different algorithm:

1. Compute the Cholesky factorization of $A = U_A^T U_A$, using xPOTRF.

[17] This bound is guaranteed only if the Level 3 BLAS are implemented in a conventional way, not in a fast way as described in section 4.13.

[18] Another interpretation of chordal distance is as half the usual Euclidean distance between the projections of $\hat{\lambda}_i$ and λ_i on the Riemann sphere, i.e., half the length of the chord connecting the projections.

2. Compute the Cholesky factorization of $B = U_B^T U_B$, using xPOTRF.
3. Compute the generalized singular value decomposition of the pair U_A, U_B using xTGSJA. The squares of the generalized singular values are the desired eigenvalues.

See sections 2.3.5.3 and 2.4.9 for a discussion of the generalized singular value decomposition, and section 4.12 for a discussion of the relevant error bound. This approach can give a tighter error bound than the above bounds when B is ill conditioned but $A + B$ is well-conditioned.

Other yet more refined algorithms and error bounds are discussed in [14, 95, 103].

4.11 Error Bounds for the Generalized Nonsymmetric Eigenproblem

We start by stating the simplest error bounds for individual eigenvalues and eigenvectors and leave the more complicated ones to subsequent subsections.

As discussed in section 2.4.8 , from a computational point of view it is more natural to define the generalized nonsymmetric eigenvalue problem in the form $\beta Ax = \alpha Bx$ with $\lambda = \alpha/\beta$ instead of $Ax = \lambda Bx$. The eigenvalue λ is represented as a pair (α, β), where a finite eigenvalue has $\beta \neq 0$ and an infinite eigenvalue has $\beta = 0$. As in the standard nonsymmetric eigenvalue problem we have both right and left eigenvectors $x \neq 0$ and $y \neq 0$, respectively, defined as

$$\beta Ax = \alpha Bx, \quad \beta y^H A = \alpha y^H B. \tag{4.8}$$

Error bounds for eigenvalues are stated in terms of the distance between pairs (α, β) and (α', β'). Let $\lambda = \alpha/\beta$ and $\lambda' = \alpha'/\beta'$. Then the *chordal distance* between λ and λ' (see section 4.10.1) can equivalently be expressed as the chordal distance between two pairs:

$$\mathcal{X}(\lambda, \lambda') = \mathcal{X}((\alpha, \beta), (\alpha', \beta')) = \frac{|\alpha\beta' - \beta\alpha'|}{\sqrt{|\alpha|^2 + |\beta|^2}\sqrt{|\alpha'|^2 + |\beta'|^2}}. \tag{4.9}$$

Now we state our error bounds. Let $(\alpha_i, \beta_i), i = 1, \ldots, n$ be the eigenvalues of (A, B), let x_i be a right eigenvector corresponding to (α_i, β_i): $\beta_i A x_i = \alpha_i B x_i$ and let $(\hat{\alpha}_i, \hat{\beta}_i)$ and $\hat{x}_i$ be the corresponding eigenvalues and eigenvectors computed by the expert driver routine xGGEVX (see subsection 2.3.5.2).

The approximate error bounds[10] for the computed eigenvalues are

$$\mathcal{X}((\hat{\alpha}_i, \hat{\beta}_i), (\alpha_i, \beta_i)) \leq \texttt{EERRBD}(i).$$

The approximate error bounds for the computed eigenvectors $\hat{x}_i$, which bound the acute angles between the computed eigenvectors and the true eigenvectors x_i are

$$\theta(\hat{x}_i, x_i) \leq \texttt{VERRBD}(i).$$

The same bounds also hold for the computed left eigenvectors.

These bounds can be computed by the following code fragment:

```
      EPSMCH = SLAMCH( 'E' )
*     Compute the generalized eigenvalues and eigenvectors of (A,B)
*     ALPHAR/BETA contains the real parts of the eigenvalues
*     ALPHAI/BETA contains the imaginary parts of the eigenvalues
*     VL contains the left eigenvectors
*     VR contains the right eigenvectors
      CALL SGGEVX( 'P', 'V', 'V', 'B', N, A, LDA, B, LDB, ALPHAR,
     $             ALPHAI, BETA, VL, LDVL, VR, LDVR, ILO, IHI, LSCALE,
     $             RSCALE, ABNRM, BBNRM, RCONDE, RCONDV, WORK, LWORK,
     $             IWORK, BWORK, INFO )
      IF( INFO.GT.0 ) THEN
         PRINT *,'INFO =', info, ' from SGGEVX.'
      ELSE IF( N.GT.0 ) THEN
         ABNORM = SLAPY2( ABNRM, BBNRM )
         DO 10 I = 1,N
            EERRBD(I) = EPSMCH*ABNORM/RCONDE(I)
            VERRBD(I) = EPSMCH*ABNORM/RCONDV(I)
   10    CONTINUE
      END IF
```

For example, suppose[19] SLAMCH('E') $= 2^{-24} = 5.960 \cdot 10^{-8}$ and

$$A = \begin{pmatrix} -132 & -88 & 84 & 104 \\ -158.4 & -79.2 & 76.8 & 129.6 \\ 129.6 & 81.6 & -79.2 & -100.8 \\ 160 & 84 & -80 & -132 \end{pmatrix}$$

and

$$B = \begin{pmatrix} -60 & -50 & 40 & 50 \\ -69 & -46.4 & 38 & 58.2 \\ 58.8 & 46 & -37.6 & -48 \\ 70 & 50 & -40 & -60 \end{pmatrix}.$$

For this problem, the exact eigenvalues, eigenvectors, and eigenvalue condition numbers are known. Then the true eigenvalues, computed eigenvalues, approximate error bounds, and true error bounds are given in the following table.

i	1	2	3	4
λ_i	1	2	3	4
$\hat{\lambda}_i$	$1.0000529 \cdot 10^0$	$1.9999847 \cdot 10^0$	$2.9999785 \cdot 10^0$	$4.0002117 \cdot 10^0$
EERRBD(i)	$9.4562565 \cdot 10^{-5}$	$5.3651773 \cdot 10^{-5}$	$5.6895351 \cdot 10^{-5}$	$1.2976544 \cdot 10^{-4}$
$\mathcal{X}(\hat{\lambda}_i, \lambda_i)$	$3.6398560 \cdot 10^{-5}$	$2.3351108 \cdot 10^{-5}$	$2.2801253 \cdot 10^{-6}$	$1.0059956 \cdot 10^{-5}$
VERRBD(i)	$1.4116328 \cdot 10^{-4}$	$1.4498082 \cdot 10^{-4}$	$6.8483077 \cdot 10^{-4}$	$5.5552053 \cdot 10^{-4}$
$\theta(\hat{l}_i, l_i)$	$1.0050300 \cdot 10^{-5}$	$6.9755580 \cdot 10^{-6}$	$1.3587955 \cdot 10^{-5}$	$4.2988235 \cdot 10^{-6}$
$\theta(\hat{r}_i, r_i)$	$5.2165419 \cdot 10^{-5}$	$1.4475762 \cdot 10^{-5}$	$5.1648690 \cdot 10^{-5}$	$7.9673846 \cdot 10^{-5}$

[19] This numerical example is computed in IEEE single precision arithmetic on a SUN Sparcstation 10 workstation.

4.11.1 Further Details: Error Bounds for the Generalized Nonsymmetric Eigenproblem

4.11.1.1 Overview

In this subsection, we summarize all the available error bounds. Later sections will provide further details. The error bounds presented here apply to regular matrix pairs only.

Bounds for individual eigenvalues and eigenvectors are provided by the driver xGGEVX (subsection 2.3.5.2) or the computational routine xTGSNA (subsection 2.4.8). Bounds for cluster of eigenvalues and their associated pair of deflating subspaces are provided by the driver xGGESX (subsection 2.3.5.2) or the computational routine xTGSEN (subsection 2.4.8).

We let $(\hat{\alpha}_i, \hat{\beta}_i)$ be the ith computed eigenvalue pair and (α_i, β_i) the ith exact eigenvalue pair.[20] Let $\hat{x}_i$ and $\hat{y}_i$ be the corresponding computed right and left eigenvectors, and x_i and y_i the exact right and left eigenvectors (so that $\beta_i A x_i = \alpha_i B x_i$ and $\beta_i y_i^H A = \alpha_i y_i^H B$). As in the standard nonsymmetric eigenvalue problem, we also want to bound the error in the average of a cluster of eigenvalues, corresponding to a subset $\mathcal{I}$ of the integers from 1 to n. However, since there are both finite and infinite eigenvalues, we need a proper definition for the average of the eigenvalues $\lambda_i = \alpha_i/\beta_i$ for $i \in \mathcal{I}$. Here we let $(\alpha_\mathcal{I}, \beta_\mathcal{I})$ denote the average of the selected eigenvalues[21]: $(\alpha_\mathcal{I}, \beta_\mathcal{I}) = (\sum_{i\in\mathcal{I}} \alpha_i, \sum_{i\in\mathcal{I}} \beta_i)/(\sum_{i\in\mathcal{I}} 1)$, and similarly for $(\hat{\alpha}_\mathcal{I}, \hat{\beta}_\mathcal{I})$. We also let $\mathcal{L}_\mathcal{I}$ and $\mathcal{R}_\mathcal{I}$ denote the exact pair of left and right deflating subspaces associated with the cluster of selected eigenvalues. Similarly, $\hat{\mathcal{L}}_\mathcal{I}$ and $\hat{\mathcal{R}}_\mathcal{I}$ are the corresponding computed pair of left and right deflating subspaces.

The algorithms for the generalized nonsymmetric eigenproblem are normwise backward stable; the computed eigenvalues, eigenvectors and deflating subspaces are the exact ones of slightly perturbed matrices $A + E$ and $B + F$, where $\|(E, F)\|_F \le p(n)\epsilon\|(A, B)\|_F$. The code fragment in the previous subsection approximates $\|(E, F)\|_F$ by $\epsilon \cdot$ **ABNORM**, where **ABNORM** $= \sqrt{\mathbf{ABNRM}^2 + \mathbf{BBNRM}^2}$, and the values **ABNRM** and **BBNRM** returned by xGGEVX are the 1-norm of the matrices A and B, respectively.

xGGEVX (or xTGSNA) returns reciprocal condition numbers for each eigenvalue pair $(\hat{\alpha}_i, \hat{\beta}_i)$ and corresponding left and right eigenvectors $\hat{y}_i$ and $\hat{x}_i$: s_i and $\mathrm{Dif}_l(i)$. s_i is a reciprocal condition number for the computed eigenpair $(\hat{\alpha}_i, \hat{\beta}_i)$, and is referred to as **RCONDE(i)** by xGGEVX. $\mathrm{Dif}_l(i)$ is a reciprocal condition number for the left and right eigenvectors $\hat{y}_i$ and $\hat{x}_i$, and is referred to as

[20] As for the nonsymmetric eigenvalue problem there is an one-to-one correspondence between computed and exact eigenvalue pairs, but the correspondence is not as simple to describe as in the generalized symmetric definite case. (Since the eigenvalues are real, sorting provides the one-to-one correspondence). For well-conditioned generalized eigenvalues $(\hat{\alpha}_i, \hat{\beta}_i)$ is usually the closest eigenvalue pair to (α_i, β_i) in the chordal metric, but in ill-conditioned cases this is not always true. The (nonconstructive) correspondence between computed and exact eigenvalues described in footnote 15 for the standard nonsymmetric eigenvalue problem is also applicable here.

[21] In order to make the definition $(\alpha_\mathcal{I}, \beta_\mathcal{I})$ of the cluster average in chordal sense meaningful for all cases that can appear we have to make a normalization of the matrix pair in generalized Schur form. First, we make each nonzero b_{ii} nonnegative real by premultiplying by the appropriate complex number with unit absolute value. Secondly, if all b_{ii} in the cluster are zero, then we make all real parts of each nonzero a_{ii} nonnegative real. This means that if there is at least one finite eigenvalue in the cluster (i.e., at least one b_{ii} is nonzero in B_{11}), then $\mathrm{trace}(B_{11}) = \sum_{i\in\mathcal{I}} \beta_i$ is nonzero and the cluster average is finite. If all b_{ii} are zero and some a_{ii} is nonzero then $\mathrm{trace}(A_{11}) = \sum_{i\in\mathcal{I}} \alpha_i$ is nonzero and the cluster average is infinity. Note the pencil is singular if one pair $(a_{ii}, b_{ii}) = (0, 0)$ or close to singular if both a_{ii} and b_{ii} are tiny (see Section 4.11.1.4).

RCONDV(i) by xGGEVX (see subsection 4.11.1.3 for definitions). Similarly, xGGESX (or xTGSEN) returns condition numbers for eigenvalue clusters and deflating subspaces corresponding to a subset $\mathcal{I}$ of the eigenvalues. These are $l_{\mathcal{I}}$ and $r_{\mathcal{I}}$, the reciprocal values of the left and right projection norms p and q, and estimates of the separation between two matrix pairs defined by $\mathrm{Dif}_u(\mathcal{I})$ and $\mathrm{Dif}_l(\mathcal{I})$ (see subsection 4.11.1.3 for definitions). xGGESX reports $l_{\mathcal{I}}$ and $r_{\mathcal{I}}$ in RCONDE(1) and RCONDE(2) (PL and PR in xTGSEN)), respectively, and estimates of $\mathrm{Dif}_u(\mathcal{I})$ and $\mathrm{Dif}_l(\mathcal{I})$ in RCONDV(1) and RCONDV(2) (DIF(1) and DIF(2) in xTGSEN), respectively.

As for the nonsymmetric eigenvalue problem we provide both asymptotic and global error bounds. The asymptotic approximate error bounds for eigenvalues, averages of eigenvalues, eigenvectors, and deflating subspaces provided in Table 4.7 are true only for sufficiently small $\|(E,F)\|_F$.

Table 4.7: Asymptotic error bounds for the generalized nonsymmetric eigenvalue problem

Simple eigenvalue:	$\mathcal{X}((\hat{\alpha}_i,\hat{\beta}_i),(\alpha_i,\beta_i)) \lesssim \|(E,F)\|_F/s_i$
Eigenvalue cluster:	$\mathcal{X}((\hat{\alpha}_{\mathcal{I}},\hat{\beta}_{\mathcal{I}}),(\alpha_{\mathcal{I}},\beta_{\mathcal{I}})) \lesssim \|(E,F)\|_F/l_{\mathcal{I}}$
Eigenvector pair:	
Left	$\theta_{\max}(\hat{y}_i,y_i) \lesssim \|(E,F)\|_F/\mathrm{Dif}_l(i)$
Right	$\theta_{\max}(\hat{x}_i,x_i) \lesssim \|(E,F)\|_F/\mathrm{Dif}_l(i)$
Deflating subspace pair:	
Left	$\theta_{\max}(\widehat{\mathcal{L}}_{\mathcal{I}},\mathcal{L}_{\mathcal{I}}) \lesssim \|(E,F)\|_F/\mathrm{Dif}_l(\mathcal{I})$
Right	$\theta_{\max}(\widehat{\mathcal{R}}_{\mathcal{I}},\mathcal{R}_{\mathcal{I}}) \lesssim \|(E,F)\|_F/\mathrm{Dif}_l(\mathcal{I})$

If the problem is ill–conditioned, the asymptotic bounds may only hold for extremely small values of $\|(E,F)\|_F$. Therefore, we also provide similar global error bounds, which are valid for all perturbations that satisfy an upper bound on $\|(E,F)\|_F$. The global error bounds in Table 4.8 are guaranteed to hold for all $\|(E,F)\|_F < \Delta$, where

- $\Delta \equiv \frac{1}{4}\min(l_i,r_i)\min(\mathrm{Dif}_u(i),\mathrm{Dif}_l(i))$ for an individual eigenvector pair, and
- $\Delta \equiv \frac{1}{4}\min(l_{\mathcal{I}},r_{\mathcal{I}})\min(\mathrm{Dif}_u(\mathcal{I}),\mathrm{Dif}_l(\mathcal{I}))$ for a cluster of eigenvalues or a deflating subspace pair.

We let $\delta \equiv \|(E,F)\|_F/\Delta$ in Table 4.8. If δ is small, then the computed pair of left and right deflating subspaces (or computed left and right eigenvectors) are small perturbations of the exact pair of deflating subspaces (or the true left and right eigenvectors). The error bounds conform with the corresponding bounds for the nonsymmetric eigenproblem (see subsection 4.8.1.1).

For ill–conditioned problems the restriction Δ on $\|(E,F)\|_F$ may also be small. Indeed, a small value of Δ shows that the cluster of eigenvalues (in the (1,1)–blocks of (A,B)) is ill–conditioned in the sense that small perturbations of (A,B) may imply that one eigenvalue in the cluster moves and coalesces with another eigenvalue (outside the cluster). Accordingly, this also means that the associated (left and right) deflating subspaces are sensitive to small perturbations, since the size of the perturbed subspaces may change for small perturbations of (A,B). See also the discussion of singular problems in section 4.11.1.4.

Table 4.8: Global error bounds for the generalized nonsymmetric eigenvalue problem assuming $\delta \equiv \|(E,F)\|_F/\Delta < 1$.

Eigenvalue cluster:	$\mathcal{X}((\hat{\alpha}_\mathcal{I},\hat{\beta}_\mathcal{I}),(\alpha_\mathcal{I},\beta_\mathcal{I})) \le 2\|(E,F)\|_F/l_\mathcal{I}$
Eigenvector pair:	
Left	$\theta_{\max}(\hat{y}_i, y_i) \le \arctan\left(\delta \cdot l_i/(1-\delta\sqrt{1-l_i^2})\right)$
Right	$\theta_{\max}(\hat{x}_i, x_i) \le \arctan\left(\delta \cdot r_i/(1-\delta\sqrt{1-r_i^2})\right)$
Deflating subspace pair:	
Left	$\theta_{\max}(\widehat{\mathcal{L}}_\mathcal{I}, \mathcal{L}_\mathcal{I}) \le \arctan\left(\delta \cdot l_\mathcal{I}/(1-\delta\sqrt{1-l_\mathcal{I}^2})\right)$
Right	$\theta_{\max}(\widehat{\mathcal{R}}_\mathcal{I}, \mathcal{R}_\mathcal{I}) \le \arctan\left(\delta \cdot r_\mathcal{I}/(1-\delta\sqrt{1-r_\mathcal{I}^2})\right)$

As for the nonsymmetric eigenvalue problem we have global error bounds for eigenvalues which are true for all E and F. Let (A,B) be a diagonalizable matrix pair. We let the columns of $\widehat{Y}$ and $\widehat{X}$ be the computed left and right eigenvectors associated with the computed generalized eigenvalue pairs $(\hat{\alpha}_i,\hat{\beta}_i), i=1,\ldots,n$. Moreover, we assume that $\widehat{Y}$ and $\widehat{X}$ are normalized such that $|\hat{\alpha}_i|^2+|\hat{\beta}_i|^2=1$ and $\|\hat{y}_i\|_2=1$, i.e., we overwrite $\hat{y}_i$ with $\hat{y}_i/\|\hat{y}_i\|_2$, $(\hat{\alpha}_i,\hat{\beta}_i)$ with $(\hat{\alpha}_i,\hat{\beta}_i)/(|\hat{\alpha}_i|^2+|\hat{\beta}_i|^2)^{1/2}$ and $\hat{x}_i$ with $\hat{x}_i\|\hat{y}_i\|_2/(|\hat{\alpha}_i|^2+|\hat{\beta}_i|^2)^{1/2}$. Then all eigenvalues (α_i,β_i) of (A,B) with $|\alpha_i|^2+|\beta_i|^2=1$ lie in the union of n regions ("spheres")

$$\left\{(\alpha,\beta), |\alpha|^2+|\beta|^2=1 : \mathcal{X}((\alpha,\beta),(\hat{\alpha}_i,\hat{\beta}_i)) \le n\|(E,F)\|_2/s_i\right\}. \tag{4.10}$$

If k of the regions overlap, so that any two points inside the k "spheres" can be connected by a continuous curve lying entirely inside the k regions, and if no larger set of $k+1$ regions has this property, then exactly k of the eigenvalues (α_i,β_i) lie inside the union of these "spheres". In other words, the global error bound with respect to an individual eigenvalue (α_i,β_i) is only useful if it defines a region that does not intersect with regions corresponding to other eigenvalues. If two or more regions intersect, then we can only say that a (true) eigenvalue of (A,B) lies in the union of the overlapping regions. If $\|(E,F)\|_2$ is so large that $(A+E,B+F)$ could be singular, which means that the eigenvalues are not well-determined by the data, then the error bound from (4.10) will be so large as to not limit the eigenvalues at all; see section 4.11.1.4 for details.

Notation Conversion For easy of reference, the following table summarizes the notation used in mathematical expression of the error bounds in tables 4.7 and 4.8 and in the corresponding driver and computational routines.

Mathematical notation	Driver Routines name	Driver Routines parameter	Computational Routines name	Computational Routines parameter
s_i	xGGEVX	RCONDE(i)	xTGSNA	S(i)
$\mathrm{Dif}_l(i)$	xGGEVX	RCONDV(i)	xTGSNA	DIF(i)
$l_{\mathcal{I}}$	xGGESX	RCONDE(1)	xTGSEN	PL
$r_{\mathcal{I}}$	xGGESX	RCONDE(2)	xTGSEN	PR
$\mathrm{Dif}_u(\mathcal{I})$	xGGESX	RCONDV(1)	xTGSEN	DIF(1)
$\mathrm{Dif}_l(\mathcal{I})$	xGGESX	RCONDV(2)	xTGSEN	DIF(2)

The quantities l_i, r_i, $\mathrm{Dif}_u(i)$ and $\mathrm{Dif}_l(i)$ used in Table 4.8 (for the global error bounds of the ith computed eigenvalue pair $(\hat{\alpha}, \hat{\beta})$ and the left and right eigenvectors $\hat{y}_i$ and $\hat{x}_i$) can be obtained by calling xTGSEN with $\mathcal{I} = \{i\}$.

4.11.1.2 Balancing and Conditioning

As in the standard nonsymmetric eigenvalue problem (section 4.8.1.2), two preprocessing steps may be performed on the input matrix pair (A, B). The first one is a **permutation**, reordering the rows and columns to attempt to make A and B block upper triangular, and therefore to reduce the order of the eigenvalue problems to be solved: we let $(A', B') = P_1(A, B)P_2$, where P_1 and P_2 are permutation matrices. The second one is a **scaling** by two-sided diagonal transformation D_1 and D_2 to make the elements of $A'' = D_1 A' D_2$ and $B'' = D_1 B' D_2$ have magnitudes as close to unity as possible, so as to reduce the effect of the roundoff error made by the later algorithm [100]. We refer to these two operations as *balancing*.

Balancing is performed by driver xGGEVX, which calls computational routine xGGBAL. The user may choose to optionally permute, scale, do both or do either; this is specified by the input parameter **BALANC** when xGGEVX is called. Permuting has no effect on the condition numbers or their interpretation as described in the previous subsections. Scaling does, however, change their interpretation, as we now describe.

The output parameters of xGGEVX – **ILO**(integer), **IHI**(integer), **LSCALE**(real array of length N), **RSCALE**(real array of length N), **ABNRM**(real) and **BBNRM**(real) – describe the result of balancing the matrix pair (A, B) to (A'', B''), where N is the dimension of (A, B). The matrix pair (A'', B'') has block upper triangular structure, with at most three blocks: from 1 to **ILO**-1, from **ILO** to **IHI**, and from **IHI**+1 to N (see section 2.4.8). The first and last blocks are upper triangular, and so already in generalized Schur form. These blocks are not scaled; only the block from **ILO** to **IHI** is scaled. Details of the left permutations (P_1) and scaling (D_1) and the right permutations (P_2) and scaling (D_2) are described in **LSCALE** and **RSCALE**, respectively. (See the specification of xGGEVX or xGGBAL for more information). The one-norms of A'' and B'' are returned in **ABNRM** and **BBNRM**, respectively.

The condition numbers described in earlier subsections are computed for the balanced matrix pair (A'', B'') in xGGEVX, and so some interpretation is needed to apply them to the eigenvalues and eigenvectors of the original matrix pair (A, B). To use the bounds for eigenvalues in Tables 4.7 and 4.8, we must replace $\|(E, F)\|_F$ by $O(\epsilon)\|(A'', B'')\|_F = O(\epsilon)\sqrt{\mathbf{ABNRM}^2 + \mathbf{BBNRM}^2}$. To use the

bounds for eigenvectors, we also need to take into account that bounds on rotation of the right and left eigenvectors are for the right and left eigenvectors x'' and y'' of A'' and B'', respectively, which are related to the right and left eigenvectors x and y by $x'' = D_2^{-1}P_2^T x$ and $y'' = D_1^{-1}P_1 y$, or $x = P_2 D_2 x''$ and $y = P_1^T D_1 x''$ respectively. Let θ'' be the bound on the rotation of x'' from Table 4.7 and Table 4.8 and let θ be the desired bound on the rotation of x. Let

$$\kappa(D_2) = \frac{\max_{\mathtt{ILO}\le i\le \mathtt{IHI}} \mathtt{RSCALE}(i)}{\min_{\mathtt{ILO}\le i\le \mathtt{IHI}} \mathtt{RSCALE}(i)}$$

be the condition number of the right scaling D_2 with respect to matrix inversion. Then

$$\sin\theta \le \kappa(D_2)\sin\theta''.$$

Similarly, for the bound of the angles ϕ and ϕ'' of the left eigenvectors y'' and y, we have

$$\sin\phi \le \kappa(D_1)\sin\phi'',$$

where $\kappa(D_1)$ is the condition number of the left scaling D_1 with respect to inversion,

$$\kappa(D_1) = \frac{\max_{\mathtt{ILO}\le i\le \mathtt{IHI}} \mathtt{LSCALE}(i)}{\min_{\mathtt{ILO}\le i\le \mathtt{IHI}} \mathtt{LSCALE}(i)}.$$

The numerical example in section 4.11 does no scaling, just permutation.

4.11.1.3 Computing s_i, $l_{\mathcal{I}}$, $r_{\mathcal{I}}$ and Dif_u, Dif_l

To explain s_i, $l_{\mathcal{I}}$, $r_{\mathcal{I}}$, Dif_u and Dif_l in Table 4.7 and Table 4.8, we need to introduce a condition number for an individual eigenvalue, block diagonalization of a matrix pair and the separation of two matrix pairs.

Let $(\alpha_i, \beta_i) \ne (0,0)$ be a simple eigenvalue of (A, B) with left and right eigenvectors y_i and x_i, respectively. s_i is the reciprocal condition number for a simple eigenvalue of (A, B) [95]:

$$s_i = \frac{\sqrt{|y_i^H A x_i|^2 + |y_i^H B x_i|^2}}{\|x_i\|_2 \|y_i\|_2}. \tag{4.11}$$

Notice that $y_i^H A x_i / y_i^H B x_i$ is equal to $\lambda_i = \alpha_i/\beta_i$. The condition number s_i in (4.11) is independent of the normalization of the eigenvectors. In the error bound of Table 4.7 for a simple eigenvalue and in (4.10), s_i is returned as `RCONDE(i)` by xGGEVX (as `S(i)` by xTGSNA).

We assume that the matrix pair (A, B) is in the generalized Schur form. Consider a cluster of m eigenvalues, counting multiplicities. Moreover, assume the n-by-n matrix pair (A, B) is

$$(A, B) = \left(\begin{pmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{pmatrix}, \begin{pmatrix} B_{11} & B_{12} \\ 0 & B_{22} \end{pmatrix} \right), \tag{4.12}$$

where the eigenvalues of the m-by-m matrix pair (A_{11}, B_{11}) are exactly those in which we are interested. In practice, if the eigenvalues on the (block) diagonal of (A, B) are not in the desired order, routine xTGEXC can be used to put the desired ones in the upper left corner as shown [73].

An equivalence transformation that block-diagonalizes (A, B) can be expressed as

$$\begin{pmatrix} I_m & -L \\ 0 & I_{n-m} \end{pmatrix} \left(\begin{pmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{pmatrix}, \begin{pmatrix} B_{11} & B_{12} \\ 0 & B_{22} \end{pmatrix} \right) \begin{pmatrix} I_m & R \\ 0 & I_{n-m} \end{pmatrix} = \left(\begin{pmatrix} A_{11} & 0 \\ 0 & A_{22} \end{pmatrix}, \begin{pmatrix} B_{11} & 0 \\ 0 & B_{22} \end{pmatrix} \right). \tag{4.13}$$

Solving for (L, R) in (4.13) is equivalent to solving the system of linear equations

$$\begin{aligned} A_{11}R - LA_{22} &= -A_{12} \\ B_{11}R - LB_{22} &= -B_{12} \end{aligned}. \tag{4.14}$$

Equation (4.14) is called a *generalized Sylvester equation* [71, 75]. Given the generalized Schur form (4.12), we solve equation (4.14) for L and R using the subroutine xTGSYL.

$l_{\mathcal{I}}$ and $r_{\mathcal{I}}$ for the eigenvalues of (A_{11}, B_{11}) are defined as

$$l_{\mathcal{I}} = \frac{1}{\sqrt{1 + \|L\|_F^2}}, \quad r_{\mathcal{I}} = \frac{1}{\sqrt{1 + \|R\|_F^2}}. \tag{4.15}$$

In the perturbation theory for the generalized eigenvalue problem, $p = l_{\mathcal{I}}^{-1}$ and $q = r_{\mathcal{I}}^{-1}$ play the same role as the norm of the spectral projector $\|P\|$ does for the standard eigenvalue problem in section 4.8.1.3. Indeed, if $B = I$, then $p = q$ and p equals the norm of the projection onto an invariant subspace of A. For the generalized eigenvalue problem we need both a left and a right projection norm since the left and right deflating subspaces are (usually) different. In Table 4.8, l_i and $l_{\mathcal{I}}$ denote the left projector norm corresponding to an individual eigenvalue pair $(\hat{\alpha}_i, \hat{\beta}_i)$ and a cluster of eigenvalues defined by the subset $\mathcal{I}$, respectively. Similar notation is used for r_i and $r_{\mathcal{I}}$. The values of $l_{\mathcal{I}}$ and $r_{\mathcal{I}}$ are returned as `RCONDE(1)` and `RCONDE(2)` from xGGESX (as `PL` and `PR` from xTGSEN).

The *separation* of two matrix pairs (A_{11}, B_{11}) and (A_{22}, B_{22}) is defined as the smallest singular value of the linear map in (4.14) which takes (L, R) to $(A_{11}R - LA_{22}, B_{11}R - LB_{22})$ [94]:

$$\mathrm{Dif}_u[(A_{11}, B_{11}), (A_{22}, B_{22})] = \inf_{\|(L,R)\|_F = 1} \|(A_{11}R - LA_{22}, B_{11}R - LB_{22})\|_F. \tag{4.16}$$

Dif_u is a generalization of the separation between two matrices ($\mathrm{sep}(A_{11}, A_{22})$ in (4.6)) to two matrix pairs, and it measures the separation of their spectra in the following sense. If (A_{11}, B_{11}) and (A_{22}, B_{22}) have a common eigenvalue, then Dif_u is zero, and it is small if there is a small perturbation of either (A_{11}, B_{11}) or (A_{22}, B_{22}) that makes them have a common eigenvalue.

Notice that $\mathrm{Dif}_u[(A_{22}, B_{22}), (A_{11}, B_{11})]$ does not generally equal $\mathrm{Dif}_u[(A_{11}, B_{11}), (A_{22}, B_{22})]$ (unless A_{ii} and B_{ii} are symmetric for $i = 1, 2$). Accordingly, the ordering of the arguments plays a role for the separation of two matrix pairs, while it does not for the separation of two matrices ($\mathrm{sep}(A_{11}, A_{22}) = \mathrm{sep}(A_{22}, A_{11})$). Therefore, we introduce the notation

$$\mathrm{Dif}_l[(A_{11}, B_{11}), (A_{22}, B_{22})] = \mathrm{Dif}_u[(A_{22}, B_{22}), (A_{11}, B_{11})]. \tag{4.17}$$

An associated generalized Sylvester operator $(A_{22}R - LA_{11}, B_{22}R - LB_{11})$ in the definition of Dif_l is obtained from block-diagonalizing a regular matrix pair in *lower* block triangular form, just as the operator $(A_{11}R - LA_{22}, B_{11}R - LB_{22})$ in the definition of Dif_u arises from block-diagonalizing a regular matrix pair (4.12) in *upper* block triangular form.

In the error bounds of Tables 4.7 and 4.8, $\mathrm{Dif}_l(i)$ and $\mathrm{Dif}_l(\mathcal{I})$ denote $\mathrm{Dif}_l[(A_{11}, B_{11}), (A_{22}, B_{22})]$, where (A_{11}, B_{11}) corresponds to an individual eigenvalue pair $(\hat{\alpha}_i, \hat{\beta}_i)$ and a cluster of eigenvalues defined by the subset $\mathcal{I}$, respectively. Similar notation is used for $\mathrm{Dif}_u(i)$ and $\mathrm{Dif}_u(\mathcal{I})$. xGGESX reports estimates of $\mathrm{Dif}_u(\mathcal{I})$ and $\mathrm{Dif}_l(\mathcal{I})$ in `RCONDV(1)` and `RCONDV(2)` (`DIF(1)` and `DIF(2)` in xTGSEN), respectively.

From a matrix representation of (4.14) it is possible to formulate an exact expression of Dif_u as

$$\mathrm{Dif}_u[(A_{11}, B_{11}), (A_{22}, B_{22})] = \sigma_{\min}(Z_u) = \mathrm{Dif}_u, \tag{4.18}$$

where Z_u is the $2m(n-m)$-by-$2m(n-m)$ matrix

$$Z_u = \begin{pmatrix} I_{n-m} \otimes A_{11} & -A_{22}^T \otimes I_m \\ I_{n-m} \otimes B_{11} & -B_{22}^T \otimes I_m \end{pmatrix},$$

and $\otimes$ is the Kronecker product. A method based directly on forming Z_u is generally impractical, since Z_u can be as large as $n^2/2 \times n^2/2$. Thus we would require as much as $O(n^4)$ extra workspace and $O(n^6)$ operations, much more than the estimation methods that we now describe.

We instead compute an estimate of Dif_u as the reciprocal value of an estimate of $\mathrm{Dif}_u^{-1} = \|Z_u^{-1}\|_2 = 1/\sigma_{\min}(Z_u)$, where Z_u is the matrix representation of the generalized Sylvester operator. It is possible to estimate $\|Z_u^{-1}\|_2$ by solving generalized Sylvester equations in triangular form. We provide both Frobenius norm and one norm Dif_u estimates [74]. The one norm estimate makes the condition estimation uniform with the nonsymmetric eigenvalue problem. The Frobenius norm estimate offers a low cost and equally reliable estimator. The one norm estimate is a factor 3 to 10 times more expensive [74]. From the definition of Dif_l (4.17) we see that Dif_l estimates can be computed by using the algorithms for estimating Dif_u.

Frobenius norm estimate: From the $Z_u x = b$ representation of the generalized Sylvester equation (4.14) we get a lower bound on Dif_u^{-1}:

$$\|(L, R)\|_F / \|(C, F)\|_F = \|x\|_2 / \|b\|_2 \le \|Z_u^{-1}\|_2. \tag{4.19}$$

To get an improved estimate we try to choose right hand sides (C, F) such that the associated solution (L, R) has as large norm as possible, giving the Dif_u estimator

$$\mathtt{DIF(1)} \equiv \|(C, F)\|_F / \|(L, R)\|_F. \tag{4.20}$$

Methods for computing such (C, F) are described in [75, 74]. The work to compute `DIF(1)` is comparable to solve a generalized Sylvester equation, which costs only $2m^2(n-m) + 2m(n-m)^2$ operations if the matrix pairs are in generalized Schur form. `DIF(2)` is the Frobenius norm Dif_l estimate.

One norm norm estimate: From the relationship

$$\frac{1}{\sqrt{2m(n-m)}} \|Z_u^{-1}\|_1 \le \|Z_u^{-1}\|_2 \le \sqrt{2m(n-m)} \|Z_u^{-1}\|_1, \tag{4.21}$$

we know that $\|Z_u^{-1}\|_1$ can never differ more than a factor $\sqrt{2m(n-m)}$ from $\|Z_u^{-1}\|_2$. So it makes sense to compute an one norm estimate of Dif_u. xLACON implements a method for estimating

the one norm of a square matrix, using reverse communication for evaluating matrix and vector products [59, 64]. We apply this method to $\|Z_u^{-1}\|_1$ by providing the solution vectors x and y of $Z_u x = z$ and a transposed system $Z_u^T y = z$, where z is determined by xLACON. In each step only one of these generalized Sylvester equations is solved using blocked algorithms [74]. xLACON returns v and EST such that $Z_u^{-1} w = v$ and $\mathrm{EST} = \|v\|_1 / \|w\|_1 \leq \|Z_u^{-1}\|_1$, resulting in the one-norm-based estimate

$$\mathtt{DIF(1)} \equiv 1/\mathrm{EST}. \tag{4.22}$$

The cost for computing this bound is roughly equal to the number of steps in the reverse communication times the cost for one generalized Sylvester solve. `DIF(2)` is the one norm Dif_l estimate.

The expert driver routines xGGEVX and xGGESX compute the Frobenius norm estimate (4.20). The routine xTGSNA also computes the Frobenius norm estimate (4.20) of Dif_u and Dif_l. The routine xTGSEN optionally computes the Frobenius norm estimate (4.20) or the one norm estimate (4.22). The choice of estimate is controlled by the input parameter `IJOB`.

4.11.1.4 Singular Eigenproblems

In this section, we give a brief discussion of singular matrix pairs (A, B).

If the determinant of $A - \lambda B$ is zero for all values of λ (or the determinant of $\beta A - \alpha B$ is zero for all (α, β)), the pair (A, B) is said to be **singular**. The eigenvalue problem of a singular pair is much more complicated than for a regular pair.

Consider for example the singular pair

$$A = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, \qquad B = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix},$$

which has one finite eigenvalue 1 and one indeterminate eigenvalue 0/0. To see that neither eigenvalue is well determined by the data, consider the slightly different problem

$$A' = \begin{pmatrix} 1 & \epsilon_1 \\ \epsilon_2 & 0 \end{pmatrix}, \qquad B' = \begin{pmatrix} 1 & \epsilon_3 \\ \epsilon_4 & 0 \end{pmatrix},$$

where the ϵ_i are tiny nonzero numbers. Then it is easy to see 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 arbitrarily 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.

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 example, for A and B to nearly have a common null space (though this condition is not necessary). For example, consider

the 16-by-16 matrices

$$A'' = \begin{pmatrix} 0.1 & 1 & & & \\ & 0.1 & 1 & & \\ & & \ddots & \ddots & \\ & & & 0.1 & 1 \\ & & & & 0.1 \end{pmatrix} \quad \text{and} \quad B'' = A''.$$

Changing the $(n, 1)$ entries of A'' and B'' to 10^{-16}, respectively, makes both A and B singular, with a common null vector. Then, using a technique analogous to the one applied to (A, B) 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* arbitrary 16 complex values.

A complete understanding of the structure of a singular eigenproblem (A, B) requires a study of its *Kronecker canonical form*, a generalization of the *Jordan canonical form*. In addition to Jordan blocks for finite and infinite eigenvalues, the Kronecker form can contain "singular blocks", which occur only if $\det(A - \lambda B) \equiv 0$ for all λ (or if A and B are nonsquare). See [53, 93, 105, 97, 29] for more details. Other numerical software, called GUPTRI, is available for computing a generalization of the Schur canonical form for singular eigenproblems [30, 31].

The error bounds discussed in this guide hold for regular pairs only (they become unbounded, or otherwise provide no information, when (A, B) is close to singular). If a (nearly) singular pencil is reported by the software discussed in this guide, then a further study of the matrix pencil should be conducted, in order to determine whether meaningful results have been computed.

4.12 Error Bounds for the Generalized Singular Value Decomposition

The generalized (or quotient) singular value decomposition of an m-by-n matrix A and a p-by-n matrix B is the pair of factorizations

$$A = U\Sigma_1[0, R]Q^T \text{ and } B = V\Sigma_2[0, R]Q^T$$

where U, V, Q, R, Σ_1 and Σ_2 are defined as follows.

- U is m-by-m, V is p-by-p, Q is n-by-n, and all three matrices are orthogonal. If A and B are complex, these matrices are unitary instead of orthogonal, and Q^T should be replaced by Q^H in the pair of factorizations.
- R is r-by-r, upper triangular and nonsingular. $[0, R]$ is r-by-n. The integer r is the rank of $\begin{pmatrix} A \\ B \end{pmatrix}$, and satisfies $r \leq n$.
- Σ_1 is m-by-r, Σ_2 is p-by-r, both are real, nonnegative and diagonal, and $\Sigma_1^T\Sigma_1 + \Sigma_2^T\Sigma_2 = I$. Write $\Sigma_1^T\Sigma_1 = \mathrm{diag}(\alpha_1^2, \ldots, \alpha_r^2)$ and $\Sigma_2^T\Sigma_2 = \mathrm{diag}(\beta_1^2, \ldots, \beta_r^2)$, where α_i and β_i lie in the

interval from 0 to 1. The ratios $\alpha_1/\beta_1, \ldots, , \alpha_r/\beta_r$ are called the **generalized singular values** of the pair A, B. If $\beta_i = 0$, then the generalized singular value α_i/β_i is **infinite**. For details on the structure of Σ_1, Σ_2 and R, see section 2.3.5.3.

The generalized singular value decomposition is computed by driver routine xGGSVD (see section 2.3.5.3). We will give error bounds for the generalized singular values in the common case where $\begin{pmatrix} A \\ B \end{pmatrix}$ has full rank $r = n$. Let $\hat{\alpha}_i$ and $\hat{\beta}_i$ be the values of α_i and β_i, respectively, computed by xGGSVD. The approximate error bound[10] for these values is

$$|\hat{\alpha}_i - \alpha_i| + |\hat{\beta}_i - \beta_i| \leq \texttt{SERRBD} \ .$$

Note that if β_i is close to zero, then a true generalized singular value α_i/β_i can differ greatly in magnitude from the computed generalized singular value $\hat{\alpha}_i/\hat{\beta}_i$, even if SERRBD is close to its minimum ϵ.

Here is another way to interpret SERRBD: if we think of α_i and β_i as representing the *subspace* $\mathcal{S}$ consisting of the straight line through the origin with slope α_i/β_i, and similarly $\hat{\alpha}_i$ and $\hat{\beta}_i$ representing the subspace $\hat{\mathcal{S}}$, then SERRBD bounds the acute angle between $\mathcal{S}$ and $\hat{\mathcal{S}}$. Note that any two lines through the origin with nearly vertical slopes (very large α/β) are close together in angle. (This is related to the *chordal distance* in section 4.10.1.)

SERRBD can be computed by the following code fragment, which for simplicity assumes $m \geq n$. (The assumption $r = n$ implies only that $p + m \geq n$. Error bounds can also be computed when $p + m \geq n > m$, with slightly more complicated code.)

```
      EPSMCH = SLAMCH( 'E' )
*     Compute generalized singular values of A and B
      CALL SGGSVD( 'N', 'N', 'N', M, N, P, K, L, A, LDA, B,
     $             LDB, ALPHA, BETA, U, LDU, V, LDV, Q, LDQ,
     $             WORK, IWORK, INFO )
*     Compute rank of [A',B']'
      RANK = K+L
      IF( INFO.GT.0 ) THEN
         PRINT *,'SGGSVD did not converge'
      ELSE IF( RANK.LT.N ) THEN
         PRINT *,'[A**T,B**T]**T not full rank'
      ELSE IF ( M .GE. N .AND. N .GT. 0 ) THEN
*        Compute reciprocal condition number RCOND of R
         CALL STRCON( 'I', 'U', 'N', N, A, LDA, RCOND, WORK, IWORK,
     $                INFO )
         RCOND = MAX( RCOND, EPSMCH )
         SERRBD = EPSMCH / RCOND
      END IF
```

For example[11], if SLAMCH('E') $= 2^{-24} = 5.961 \cdot 10^{-8}$,

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \\ 4 & 5 & 6 \\ 7 & 8 & 8 \end{pmatrix} \text{ and } B = \begin{pmatrix} -2 & -3 & 3 \\ 4 & 6 & 5 \end{pmatrix}$$

then, to 4 decimal places,

$$\begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{pmatrix} = \begin{pmatrix} \hat{\alpha}_1 \\ \hat{\alpha}_2 \\ \hat{\alpha}_3 \end{pmatrix} = \begin{pmatrix} 1.000 \\ .7960 \\ 7.993 \cdot 10^{-2} \end{pmatrix} \text{ and } \begin{pmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{pmatrix} = \begin{pmatrix} \hat{\beta}_1 \\ \hat{\beta}_2 \\ \hat{\beta}_3 \end{pmatrix} = \begin{pmatrix} 0 \\ .6053 \\ .9968 \end{pmatrix},$$

SERRBD $= 1.4 \cdot 10^{-6}$, and the true errors are 0, $4.3 \cdot 10^{-7}$ and $1.5 \cdot 10^{-7}$.

4.12.1 Further Details: Error Bounds for the Generalized Singular Value Decomposition

The GSVD algorithm used in LAPACK ([83, 10, 8]) is backward stable:

> Let the computed GSVD of A and B be $\hat{U}\hat{\Sigma}_1[0, \hat{R}]\hat{Q}^T$ and $\hat{V}\hat{\Sigma}_2[0, \hat{R}]\hat{Q}^T$. This is nearly the exact GSVD of $A + E$ and $B + F$ in the following sense. E and F are small:
>
> $$\|E\|_2/\|A\|_2 \le p(n)\epsilon \text{ and } \|F\|_2/\|B\|_2 \le p(n)\epsilon \ ;$$
>
> there exist small $\delta\hat{Q}$, $\delta\hat{U}$, and $\delta\hat{V}$ such that $\hat{Q} + \delta\hat{Q}$, $\hat{U} + \delta\hat{U}$, and $\hat{V} + \delta\hat{V}$ are exactly orthogonal (or unitary):
>
> $$\|\delta\hat{Q}\|_2 \le p(n)\epsilon \ , \ \|\delta\hat{U}\|_2 \le p(n)\epsilon \text{ and } \|\delta\hat{V}\|_2 \le p(n)\epsilon \ ;$$
>
> and
>
> $$A + E = (\hat{U} + \delta\hat{U})\hat{\Sigma}_1[0, \hat{R}](\hat{Q} + \delta\hat{Q})^T \text{ and } B + F = (\hat{V} + \delta\hat{V})\hat{\Sigma}_2[0, \hat{R}](\hat{Q} + \delta\hat{Q})^T$$
>
> is the exact GSVD of $A + E$ and $B + F$. Here $p(n)$ is a modestly growing function of n, and we take $p(n) = 1$ in the above code fragment.
>
> Let α_i and β_i be the square roots of the diagonal entries of the exact $\Sigma_1^T\Sigma_1$ and $\Sigma_2^T\Sigma_2$, and let $\hat{\alpha}_i$ and $\hat{\beta}_i$ the square roots of the diagonal entries of the computed $\hat{\Sigma}_1^T\hat{\Sigma}_1$ and $\hat{\Sigma}_2^T\hat{\Sigma}_2$. Let
>
> $$G = \begin{pmatrix} A \\ B \end{pmatrix} \text{ and } \hat{G} = \begin{pmatrix} \hat{A} \\ \hat{B} \end{pmatrix} .$$
>
> Then provided G and $\hat{G}$ have full rank n, one can show [96, 82] that
>
> $$\left(\sum_{i=1}^{n}[(\hat{\alpha}_i - \alpha_i)^2 + (\hat{\beta}_i - \beta_i)^2]\right)^{1/2} \le \frac{\sqrt{2}\left\|\begin{pmatrix} E \\ F \end{pmatrix}\right\|}{\min(\sigma_{\min}(G), \sigma_{\min}(\hat{G}))} .$$

In the code fragment we approximate the numerator of the last expression by $\epsilon\|\hat{R}\|_\infty$ and approximate the denominator by $\|\hat{R}^{-1}\|_\infty^{-1}$ in order to compute SERRBD; STRCON returns an approximation RCOND to $1/(\|\hat{R}^{-1}\|_\infty\|\hat{R}\|_\infty)$.

We assume that the rank r of G equals n, because otherwise the α_is and β_is are not well determined. For example, if

$$A = \begin{pmatrix} 10^{-16} & 0 \\ 0 & 1 \end{pmatrix}, \quad B = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \text{ and } A' = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}, \quad B' = \begin{pmatrix} 10^{-16} & 0 \\ 0 & 1 \end{pmatrix}$$

then A and B have $\alpha_{1,2} = 1, .7071$ and $\beta_{1,2} = 0, .7071$, whereas A' and B' have $\alpha'_{1,2} = 0, .7071$ and $\beta'_{1,2} = 1, .7071$, which are completely different, even though $\|A-A'\| = 10^{-16}$ and $\|B-B'\| = 10^{-16}$. In this case, $\sigma_{\min}(G) = 10^{-16}$, so G is nearly rank-deficient.

The reason the code fragment assumes $m \geq n$ is that in this case $\hat{R}$ is stored overwritten on A, and can be passed to STRCON in order to compute RCOND. If $m \leq n$, then the first m rows of $\hat{R}$ are stored in A, and the last $n-m$ rows of $\hat{R}$ are stored in B. This complicates the computation of RCOND: either $\hat{R}$ must be copied to a single array before calling STRCON, or else the lower level subroutine SLACON must be used with code capable of solving linear equations with $\hat{R}$ and $\hat{R}^T$ as coefficient matrices.

4.13 Error Bounds for Fast Level 3 BLAS

The Level 3 BLAS specifications [40] specify the input, output and calling sequence for each routine, but allow freedom of implementation, subject to the requirement that the routines be numerically stable. Level 3 BLAS implementations can therefore be built using matrix multiplication algorithms that achieve a more favorable operation count (for suitable dimensions) than the standard multiplication technique, provided that these "fast" algorithms are numerically stable. The simplest fast matrix multiplication technique is Strassen's method, which can multiply two n-by-n matrices in fewer than $4.7n^{\log_2 7}$ operations, where $\log_2 7 \approx 2.807$.

The effect on the results in this chapter of using a fast Level 3 BLAS implementation can be explained as follows. In general, reasonably implemented fast Level 3 BLAS preserve all the bounds presented here (except those at the end of subsection 4.10), but the constant $p(n)$ may increase somewhat. Also, the iterative refinement routine xyyRFS may take more steps to converge.

This is what we mean by reasonably implemented fast Level 3 BLAS. Here, c_i denotes a constant depending on the specified matrix dimensions.

(1) If A is m-by-n, B is n-by-p and $\widehat{C}$ is the computed approximation to $C = AB$, then

$$\|\widehat{C} - AB\|_\infty \leq c_1(m,n,p)\epsilon\|A\|_\infty\|B\|_\infty + O(\epsilon^2).$$

(2) The computed solution $\widehat{X}$ to the triangular systems $TX = B$, where T is m-by-m and B is m-by-p, satisfies

$$\|T\widehat{X} - B\|_\infty \leq c_2(m,p)\epsilon\|T\|_\infty\|\widehat{X}\|_\infty + O(\epsilon^2).$$

For conventional Level 3 BLAS implementations these conditions hold with $c_1(m,n,p) = n^2$ and $c_2(m,p) = m(m+1)$. Strassen's method satisfies these bounds for slightly larger c_1 and c_2.

For further details, and references to fast multiplication techniques, see [27].

Chapter 5

Documentation and Software Conventions

5.1 Design and Documentation of Argument Lists

The argument lists of all LAPACK routines conform to a single set of conventions for their design and documentation.

Specifications of all LAPACK driver and computational routines are given in Part 2. These are derived from the specifications given in the leading comments in the code, but in Part 2 the specifications for real and complex versions of each routine have been merged, in order to save space.

5.1.1 Structure of the Documentation

The documentation of each LAPACK routine includes:

- the SUBROUTINE or FUNCTION statement, followed by statements declaring the type and dimensions of the arguments;
- a summary of the **Purpose** of the routine;
- descriptions of each of the **Arguments** in the order of the argument list;
- (optionally) **Further Details** (only in the code, not in Part 2);
- (optionally) **Internal Parameters** (only in the code, not in Part 2).

5.1.2 Order of Arguments

Arguments of an LAPACK routine appear in the following order:

- arguments specifying options;
- problem dimensions;
- array or scalar arguments defining the input data; some of them may be overwritten by results;
- other array or scalar arguments returning results;
- work arrays (and associated array dimensions);
- diagnostic argument INFO.

5.1.3 Argument Descriptions

The style of the argument descriptions is illustrated by the following example:

N	(input) INTEGER The number of columns of the matrix A. N $\geq$ 0.
A	(input/output) REAL array, dimension (LDA,N) On entry, the m-by-n matrix to be factored. On exit, the factors L and U from the factorization A = P*L*U; the unit diagonal elements of L are not stored.

The description of each argument gives:

- a classification of the argument as (input), (output), (input/output), (input or output)[1], (workspace) or (workspace/output)[2];
- the type of the argument;
- (for an array) its dimension(s);
- a specification of the value(s) that must be supplied for the argument (if it's an input argument), or of the value(s) returned by the routine (if it's an output argument), or both (if it's an input/output argument). In the last case, the two parts of the description are introduced by the phrases "On entry" and "On exit".
- (for a scalar input argument) any constraints that the supplied values must satisfy (such as "N $\geq$ 0" in the example above).

[1](Input or output) means that the argument may be either an input argument or an output argument, depending on the values of other arguments; for example, in the xyySVX driver routines, some arguments are used either as output arguments to return details of a factorization, or as input arguments to supply details of a previously computed factorization.

[2](Workspace/output) means that the argument is used principally as a work array, but may also return some useful information (in its first element)

5.1.4 Option Arguments

Arguments specifying options are usually of type CHARACTER*1. The meaning of each valid value is given, as in this example:

UPLO (input) CHARACTER*1
= 'U': Upper triangle of A is stored;
= 'L': Lower triangle of A is stored.

The corresponding lower-case characters may be supplied (with the same meaning), but any other value is illegal (see subsection 5.1.9).

A longer character string can be passed as the actual argument, making the calling program more readable, but only the first character is significant; this is a standard feature of Fortran 77. For example:

```
CALL SPOTRS('upper', . . . )
```

5.1.5 Problem Dimensions

It is permissible for the problem dimensions to be passed as zero, in which case the computation (or part of it) is skipped. Negative dimensions are regarded as erroneous.

5.1.6 Array Arguments

Each two-dimensional array argument is immediately followed in the argument list by its leading dimension, whose name has the form LD<array-name>. For example:

A (input/output) REAL/COMPLEX array, dimension (LDA,N)
...
LDA (input) INTEGER
The leading dimension of the array A. LDA $\geq$ max(1,M).

It should be assumed, unless stated otherwise, that vectors and matrices are stored in one- and two-dimensional arrays in the conventional manner. That is, if an array X of dimension (N) holds a vector x, then X(i) holds x_i for $i = 1, \ldots, n$. If a two-dimensional array A of dimension (LDA,N) holds an m-by-n matrix A, then A(i, j) holds a_{ij} for $i = 1, \ldots, m$ and $j = 1, \ldots, n$ (LDA must be at least m). See Section 5.3 for more about storage of matrices.

Note that array arguments are usually declared in the software as assumed-size arrays (last dimension *), for example:

```
REAL A( LDA, * )
```

although the documentation gives the dimensions as (LDA,N). The latter form is more informative since it specifies the required minimum value of the last dimension. However an assumed-size array declaration has been used in the software, in order to overcome some limitations in the Fortran 77 standard. In particular it allows the routine to be called when the relevant dimension (N, in this case) is zero. However actual array dimensions in the calling program must be at least 1 (LDA in this example).

5.1.7 Work Arrays

Many LAPACK routines require one or more work arrays to be passed as arguments. The name of a work array is usually WORK — sometimes IWORK, RWORK or BWORK to distinguish work arrays of integer, real or logical (Boolean) type.

Occasionally the first element of a work array is used to return some useful information: in such cases, the argument is described as (workspace/output) instead of simply (workspace).

A number of routines implementing block algorithms require workspace sufficient to hold one block of rows or columns of the matrix, for example, workspace of size n-by-nb, where nb is the block size. In such cases, the actual declared length of the work array must be passed as a separate argument LWORK, which immediately follows WORK in the argument-list.

See Section 5.2 for further explanation.

5.1.8 LWORK Query

If in doubt about the amount of workspace to supply to an LAPACK routine, the user may choose to supply $LWORK = -1$ and use the returned value in $WORK(1)$ as the correct value for $LWORK$. Setting $LWORK = -1$ does not invoke an error message from `XERBLA` and is defined as a global query.

The routines xGEESX and xGGESX are the only exceptions to this rule. For these routines, the value of LWORK is dependent upon the dimension of the invariant subspace (SDIM), and is not known on entry to the routine.

5.1.9 Error Handling and the Diagnostic Argument INFO

All documented routines have a diagnostic argument INFO that indicates the success or failure of the computation, as follows:

- INFO = 0: successful termination
- INFO < 0: illegal value of one or more arguments — no computation performed
- INFO > 0: failure in the course of computation

All driver and auxiliary routines check that input arguments such as N or LDA or option arguments of type character have permitted values. If an illegal value of the ith argument is detected, the routine sets INFO $= -i$, and then calls an error-handling routine XERBLA.

The standard version of XERBLA issues an error message and halts execution, so that no LAPACK routine would ever return to the calling program with INFO < 0. However, this might occur if a non-standard version of XERBLA is used.

5.2 Determining the Block Size for Block Algorithms

LAPACK routines that implement block algorithms need to determine what block size to use. The intention behind the design of LAPACK is that the choice of block size should be hidden from users as much as possible, but at the same time easily accessible to installers of the package when tuning LAPACK for a particular machine.

LAPACK routines call an auxiliary enquiry function ILAENV, which returns the optimal block size to be used, as well as other parameters. The version of ILAENV supplied with the package contains default values that led to good behavior over a reasonable number of our test machines, but to achieve optimal performance, it may be beneficial to tune ILAENV for your particular machine environment. Ideally a distinct implementation of ILAENV is needed for each machine environment (see also Chapter 6). The optimal block size may also depend on the routine, the combination of option arguments (if any), and the problem dimensions.

If ILAENV returns a block size of 1, then the routine performs the unblocked algorithm, calling Level 2 BLAS, and makes no calls to Level 3 BLAS.

Some LAPACK routines require a work array whose size is proportional to the block size (see subsection 5.1.7). The actual length of the work array is supplied as an argument LWORK. The description of the arguments WORK and LWORK typically goes as follows:

WORK (workspace) REAL array, dimension (LWORK)
On exit, if INFO = 0, then WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK. LWORK $\geq$ max(1,N).
For optimal performance LWORK $\geq$ N*NB, where NB is the optimal block size returned by ILAENV.

The routine determines the block size to be used by the following steps:

1. the optimal block size is determined by calling ILAENV;
2. if the value of LWORK indicates that enough workspace has been supplied, the routine uses the optimal block size;
3. otherwise, the routine determines the largest block size that can be used with the supplied amount of workspace;

4. if this new block size does not fall below a threshold value (also returned by ILAENV), the routine uses the new value;

5. otherwise, the routine uses the unblocked algorithm.

The minimum value of LWORK that would be needed to use the optimal block size, is returned in WORK(1).

Thus, the routine uses the largest block size allowed by the amount of workspace supplied, as long as this is likely to give better performance than the unblocked algorithm. WORK(1) is not always a simple formula in terms of N and NB.

The specification of LWORK gives the minimum value for the routine to return correct results. If the supplied value is less than the minimum — indicating that there is insufficient workspace to perform the unblocked algorithm — the value of LWORK is regarded as an illegal value, and is treated like any other illegal argument value (see subsection 5.1.9).

If in doubt about how much workspace to supply, users should supply a generous amount (assume a block size of 64, say), and then examine the value of WORK(1) on exit.

5.3 Matrix Storage Schemes

LAPACK allows the following different storage schemes for matrices:

- conventional storage in a two-dimensional array;
- packed storage for symmetric, Hermitian or triangular matrices;
- band storage for band matrices;
- the use of two or three one-dimensional arrays to store tridiagonal or bidiagonal matrices.

These storage schemes are compatible with those used in LINPACK and the BLAS, but EISPACK uses incompatible schemes for band and tridiagonal matrices.

In the examples below, $*$ indicates an array element that need not be set and is not referenced by LAPACK routines. Elements that "need not be set" are never read, written to, or otherwise accessed by the LAPACK routines. The examples illustrate only the relevant part of the arrays; array arguments may of course have additional rows or columns, according to the usual rules for passing array arguments in Fortran 77.

5.3.1 Conventional Storage

The default scheme for storing matrices is the obvious one described in subsection 5.1.6: a matrix A is stored in a two-dimensional array A, with matrix element a_{ij} stored in array element A(i, j).

If a matrix is **triangular** (upper or lower, as specified by the argument UPLO), only the elements of the relevant triangle are accessed. The remaining elements of the array need not be set. Such elements are indicated by $*$ in the examples below. For example, when $n = 4$:

UPLO	Triangular matrix A	Storage in array A
'U'	$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ & a_{22} & a_{23} & a_{24} \\ & & a_{33} & a_{34} \\ & & & a_{44} \end{pmatrix}$	$\begin{matrix} a_{11} & a_{12} & a_{13} & a_{14} \\ * & a_{22} & a_{23} & a_{24} \\ * & * & a_{33} & a_{34} \\ * & * & * & a_{44} \end{matrix}$
'L'	$\begin{pmatrix} a_{11} & & & \\ a_{21} & a_{22} & & \\ a_{31} & a_{32} & a_{33} & \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$	$\begin{matrix} a_{11} & * & * & * \\ a_{21} & a_{22} & * & * \\ a_{31} & a_{32} & a_{33} & * \\ a_{41} & a_{42} & a_{43} & a_{44} \end{matrix}$

Similarly, if the matrix is upper Hessenberg, elements below the first subdiagonal need not be set.

Routines that handle **symmetric** or **Hermitian** matrices allow for either the upper or lower triangle of the matrix (as specified by UPLO) to be stored in the corresponding elements of the array; the remaining elements of the array need not be set. For example, when $n = 4$:

UPLO	Hermitian matrix A	Storage in array A
'U'	$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ \bar{a}_{12} & a_{22} & a_{23} & a_{24} \\ \bar{a}_{13} & \bar{a}_{23} & a_{33} & a_{34} \\ \bar{a}_{14} & \bar{a}_{24} & \bar{a}_{34} & a_{44} \end{pmatrix}$	$\begin{matrix} a_{11} & a_{12} & a_{13} & a_{14} \\ * & a_{22} & a_{23} & a_{24} \\ * & * & a_{33} & a_{34} \\ * & * & * & a_{44} \end{matrix}$
'L'	$\begin{pmatrix} a_{11} & \bar{a}_{21} & \bar{a}_{31} & \bar{a}_{41} \\ a_{21} & a_{22} & \bar{a}_{32} & \bar{a}_{42} \\ a_{31} & a_{32} & a_{33} & \bar{a}_{43} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$	$\begin{matrix} a_{11} & * & * & * \\ a_{21} & a_{22} & * & * \\ a_{31} & a_{32} & a_{33} & * \\ a_{41} & a_{42} & a_{43} & a_{44} \end{matrix}$

5.3.2 Packed Storage

Symmetric, Hermitian or triangular matrices may be stored more compactly, if the relevant triangle (again as specified by UPLO) is packed **by columns** in a one-dimensional array. In LAPACK, arrays that hold matrices in packed storage, have names ending in 'P'. So:

- if UPLO = 'U', a_{ij} is stored in AP$(i + j(j-1)/2)$ for $i \leq j$;
- if UPLO = 'L', a_{ij} is stored in AP$(i + (2n-j)(j-1)/2)$ for $j \leq i$.

For example:

UPLO	Triangular matrix A	Packed storage in array AP
'U'	$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ & a_{22} & a_{23} & a_{24} \\ & & a_{33} & a_{34} \\ & & & a_{44} \end{pmatrix}$	$a_{11}\ \underbrace{a_{12}\ a_{22}}\ \underbrace{a_{13}\ a_{23}\ a_{33}}\ \underbrace{a_{14}\ a_{24}\ a_{34}\ a_{44}}$
'L'	$\begin{pmatrix} a_{11} & & & \\ a_{21} & a_{22} & & \\ a_{31} & a_{32} & a_{33} & \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$	$\underbrace{a_{11}\ a_{21}\ a_{31}\ a_{41}}\ \underbrace{a_{22}\ a_{32}\ a_{42}}\ \underbrace{a_{33}\ a_{43}}\ a_{44}$

Note that for real or complex symmetric matrices, packing the upper triangle by columns is equivalent to packing the lower triangle by rows; packing the lower triangle by columns is equivalent to packing the upper triangle by rows. For complex Hermitian matrices, packing the upper triangle by columns is equivalent to packing the conjugate of the lower triangle by rows; packing the lower triangle by columns is equivalent to packing the conjugate of the upper triangle by rows.

5.3.3 Band Storage

An m-by-n band matrix with kl subdiagonals and ku superdiagonals may be stored compactly in a two-dimensional array with $kl + ku + 1$ rows and n columns. Columns of the matrix are stored in corresponding columns of the array, and diagonals of the matrix are stored in rows of the array. This storage scheme should be used in practice only if $kl, ku \ll \min(m, n)$, although LAPACK routines work correctly for all values of kl and ku. In LAPACK, arrays that hold matrices in band storage have names ending in 'B'.

To be precise, a_{ij} is stored in $\mathrm{AB}(ku + 1 + i - j, j)$ for $\max(1, j - ku) \le i \le \min(m, j + kl)$. For example, when $m = n = 5$, $kl = 2$ and $ku = 1$:

Band matrix A	Band storage in array AB
$\begin{pmatrix} a_{11} & a_{12} & & & \\ a_{21} & a_{22} & a_{23} & & \\ a_{31} & a_{32} & a_{33} & a_{34} & \\ & a_{42} & a_{43} & a_{44} & a_{45} \\ & & a_{53} & a_{54} & a_{55} \end{pmatrix}$	$\begin{matrix} * & a_{12} & a_{23} & a_{34} & a_{45} \\ a_{11} & a_{22} & a_{33} & a_{44} & a_{55} \\ a_{21} & a_{32} & a_{43} & a_{54} & * \\ a_{31} & a_{42} & a_{53} & * & * \end{matrix}$

The elements marked $*$ in the upper left and lower right corners of the array AB need not be set, and are not referenced by LAPACK routines.

Note: when a band matrix is supplied for LU factorization, space must be allowed to store an additional kl superdiagonals, generated by fill-in as a result of row interchanges. This means that the matrix is stored according to the above scheme, but with $kl + ku$ superdiagonals.

Triangular band matrices are stored in the same format, with either $kl = 0$ if upper triangular, or $ku = 0$ if lower triangular.

For symmetric or Hermitian band matrices with kd subdiagonals or superdiagonals, only the upper or lower triangle (as specified by UPLO) need be stored:

- if UPLO = 'U', a_{ij} is stored in AB$(kd+1+i-j, j)$ for $\max(1, j-kd) \le i \le j$;
- if UPLO = 'L', a_{ij} is stored in AB$(1+i-j, j)$ for $j \le i \le \min(n, j+kd)$.

For example, when $n = 5$ and $kd = 2$:

UPLO	Hermitian band matrix A	Band storage in array AB
'U'	$\begin{pmatrix} a_{11} & a_{12} & a_{13} & & \\ \bar{a}_{12} & a_{22} & a_{23} & a_{24} & \\ \bar{a}_{13} & \bar{a}_{23} & a_{33} & a_{34} & a_{35} \\ & \bar{a}_{24} & \bar{a}_{34} & a_{44} & a_{45} \\ & & \bar{a}_{35} & \bar{a}_{45} & a_{55} \end{pmatrix}$	$\begin{matrix} * & * & a_{13} & a_{24} & a_{35} \\ * & a_{12} & a_{23} & a_{34} & a_{45} \\ a_{11} & a_{22} & a_{33} & a_{44} & a_{55} \end{matrix}$
'L'	$\begin{pmatrix} a_{11} & \bar{a}_{21} & \bar{a}_{31} & & \\ a_{21} & a_{22} & \bar{a}_{32} & \bar{a}_{42} & \\ a_{31} & a_{32} & a_{33} & \bar{a}_{43} & \bar{a}_{53} \\ & a_{42} & a_{43} & a_{44} & \bar{a}_{54} \\ & & a_{53} & a_{54} & a_{55} \end{pmatrix}$	$\begin{matrix} a_{11} & a_{22} & a_{33} & a_{44} & a_{55} \\ a_{21} & a_{32} & a_{43} & a_{54} & * \\ a_{31} & a_{42} & a_{53} & * & * \end{matrix}$

EISPACK routines use a different storage scheme for band matrices, in which rows of the matrix are stored in corresponding rows of the array, and diagonals of the matrix are stored in columns of the array (see Appendix D).

5.3.4 Tridiagonal and Bidiagonal Matrices

An unsymmetric tridiagonal matrix of order n is stored in three one-dimensional arrays, one of length n containing the diagonal elements, and two of length $n-1$ containing the subdiagonal and superdiagonal elements in elements $1 : n-1$.

A symmetric tridiagonal or bidiagonal matrix is stored in two one-dimensional arrays, one of length n containing the diagonal elements, and one of length $n-1$ containing the off-diagonal elements. (EISPACK routines store the off-diagonal elements in elements $2 : n$ of a vector of length n.)

5.3.5 Unit Triangular Matrices

Some LAPACK routines have an option to handle unit triangular matrices (that is, triangular matrices with diagonal elements = 1). This option is specified by an argument DIAG. If DIAG = 'U' (Unit triangular), the diagonal elements of the matrix need not be stored, and the corresponding array elements are not referenced by the LAPACK routines. The storage scheme for the rest of the matrix (whether conventional, packed or band) remains unchanged, as described in subsections 5.3.1, 5.3.2 and 5.3.3.

5.3.6 Real Diagonal Elements of Complex Matrices

Complex Hermitian matrices have diagonal matrices that are by definition purely real. In addition, some complex triangular matrices computed by LAPACK routines are defined by the algorithm to have real diagonal elements — in Cholesky or QR factorization, for example.

If such matrices are supplied as input to LAPACK routines, the imaginary parts of the diagonal elements are not referenced, but are assumed to be zero. If such matrices are returned as output by LAPACK routines, the computed imaginary parts are explicitly set to zero.

5.4 Representation of Orthogonal or Unitary Matrices

A real orthogonal or complex unitary matrix (usually denoted Q) is often represented in LAPACK as a product of **elementary reflectors** — also referred to as **elementary Householder matrices** (usually denoted H_i). For example,

$$Q = H_1 H_2 \ldots H_k.$$

Most users need not be aware of the details, because LAPACK routines are provided to work with this representation:

- routines whose names begin SORG- (real) or CUNG- (complex) can generate all or part of Q explicitly;
- routines whose name begin SORM- (real) or CUNM- (complex) can multiply a given matrix by Q or Q^H without forming Q explicitly.

The following further details may occasionally be useful.

An elementary reflector (or elementary Householder matrix) H of order n is a unitary matrix of the form

$$H = I - \tau v v^H \tag{5.1}$$

where τ is a scalar, and v is an n-vector, with $|\tau|^2 ||v||_2^2 = 2\mathrm{Re}(\tau)$; v is often referred to as the **Householder vector** . Often v has several leading or trailing zero elements, but for the purpose of this discussion assume that H has no such special structure.

There is some redundancy in the representation (5.1), which can be removed in various ways. The representation used in LAPACK (which differs from those used in LINPACK or EISPACK) sets $v_1 = 1$; hence v_1 need not be stored. In real arithmetic, $1 \leq \tau \leq 2$, except that $\tau = 0$ implies $H = I$.

In complex arithmetic, τ may be complex, and satisfies $1 \leq \mathrm{Re}(\tau) \leq 2$ and $|\tau - 1| \leq 1$. Thus a complex H is not Hermitian (as it is in other representations), but it is unitary, which is the important property. The advantage of allowing τ to be complex is that, given an arbitrary complex vector x, H can be computed so that

$$H^H x = \beta(1, 0, \ldots, 0)^T$$

with *real* β. This is useful, for example, when reducing a complex Hermitian matrix to real symmetric tridiagonal form, or a complex rectangular matrix to real bidiagonal form.

For further details, see Lehoucq [79].

Chapter 6

Installing LAPACK Routines

6.1 Points to Note

For anyone who obtains the complete LAPACK package from *netlib* (see Chapter 1), a comprehensive installation guide is provided. We recommend installation of the complete package as the most convenient and reliable way to make LAPACK available.

People who obtain copies of a few LAPACK routines from *netlib* need to be aware of the following points:

1. Double precision complex routines (names beginning Z-) use a COMPLEX*16 data type. This is an extension to the Fortran 77 standard, but is provided by many Fortran compilers on machines where double precision computation is usual. The following related extensions are also used:

   - the intrinsic function DCONJG, with argument and result of type COMPLEX*16;
   - the intrinsic functions DBLE and DIMAG, with COMPLEX*16 argument and DOUBLE PRECISION result, returning the real and imaginary parts, respectively;
   - the intrinsic function DCMPLX, with DOUBLE PRECISION argument(s) and COMPLEX*16 result;
   - COMPLEX*16 constants, formed from a pair of double precision constants in parentheses.

   Some compilers provide DOUBLE COMPLEX as an alternative to COMPLEX*16, and an intrinsic function DREAL instead of DBLE to return the real part of a COMPLEX*16 argument. If the compiler does not accept the constructs used in LAPACK, the installer will have to modify the code: for example, globally change COMPLEX*16 to DOUBLE COMPLEX, or selectively change DBLE to DREAL.[1]

[1] Changing DBLE to DREAL must be selective, because instances of DBLE with an *integer* argument must *not* be changed. The compiler should flag any instances of DBLE with a COMPLEX*16 argument if it does not accept them.

2. For optimal performance, a small set of tuning parameters must be set for each machine, or even for each configuration of a given machine (for example, different parameters may be optimal for different numbers of processors). These values, such as the block size, minimum block size, crossover point below which an unblocked routine should be used, and others, are set by calls to an inquiry function ILAENV. The default version of ILAENV provided with LAPACK uses generic values which often give satisfactory performance, but users who are particularly interested in performance may wish to modify this subprogram or substitute their own version. Further details on setting ILAENV for a particular environment are provided in section 6.2.

3. SLAMCH/DLAMCH determines properties of the floating-point arithmetic at run-time, such as the machine epsilon, underflow threshold, overflow threshold, and related parameters. It works satisfactorily on all commercially important machines of which we are aware, but will necessarily be updated from time to time as new machines and compilers are produced.

6.2 Installing ILAENV

Machine-dependent parameters such as the block size are set by calls to an inquiry function which may be set with different values on each machine. The declaration of the environment inquiry function is

```
INTEGER FUNCTION ILAENV( ISPEC, NAME, OPTS, N1, N2, N3, N4 )
```

where ISPEC, N1, N2, N3, and N4 are integer variables and NAME and OPTS are CHARACTER*(*). NAME specifies the subroutine name: OPTS is a character string of options to the subroutine; and N1–N4 are the problem dimensions. ISPEC specifies the parameter to be returned; the following values are currently used in LAPACK:

ISPEC = 1: NB, optimal block size
= 2: NBMIN, minimum block size for the block routine to be used
= 3: NX, crossover point (in a block routine, for N < NX, an unblocked routine should be used)
= 4: NS, number of shifts
= 6: NXSVD is the threshold point for which the QR factorization is performed prior to reduction to bidiagonal form. If M > NXSVD · N, then a QR factorization is performed.
= 8: MAXB, crossover point for block multishift QR
= 9: SMLSIZ, maximum size of the subproblems at the bottom of the computation tree in the divide-and-conquer algorithm
= 10: NAN, IEEE NaN arithmetic can be trusted not to trap
= 11: INFINITY, infinity arithmetic can be trusted not to trap

The three block size parameters, NB, NBMIN, and NX, are used in many different subroutines (see Table 6.1). NS and MAXB are used in the block multishift QR algorithm, xHSEQR. NXSVD is used in the driver routines xGELSS and xGESVD. SMLSIZ is used in the divide and conquer routines xBDSDC, xGELSD, xGESDD, and xSTEDC. The parameters NAN and INFINITY are used in the driver routines xSTEVR and xSYEVR/xCHEEVR to check for IEEE-754 compliance. If compliance is detected, then these driver routines call xSTEGR. Otherwise, a slower algorithm is selected.

real	complex	NB	NBMIN	NX
SGBTRF	CGBTRF	•		
SGEBRD	CGEBRD	•	•	•
SGEHRD	CGEHRD	•	•	•
SGELQF	CGELQF	•	•	•
SGEQLF	CGEQLF	•	•	•
SGEQRF	CGEQRF	•	•	•
SGERQF	CGERQF	•	•	•
SGETRF	CGETRF	•		
SGETRI	CGETRI	•	•	
SORGLQ	CUNGLQ	•	•	•
SORGQL	CUNGQL	•	•	•
SORGQR	CUNGQR	•	•	•
SORGRQ	CUNGRQ	•	•	•
SORMLQ	CUNMLQ	•	•	
SORMQL	CUNMQL	•	•	
SORMQR	CUNMQR	•	•	
SORMRQ	CUNMRQ	•	•	
SPBTRF	CPBTRF	•		
SPOTRF	CPOTRF	•		
SPOTRI	CPOTRI	•		
SSTEBZ		•		
SSYGST	CHEGST	•		
SSYTRD	CHETRD	•	•	•
SSYTRF	CHETRF	•	•	
	CSYTRF	•	•	
STRTRI	CTRTRI	•		

Table 6.1: Use of the block parameters NB, NBMIN, and NX in LAPACK

The LAPACK testing and timing programs use a special version of ILAENV where the parameters are set via a COMMON block interface. This is convenient for experimenting with different values of, say, the block size in order to exercise different parts of the code and to compare the relative performance of different parameter values.

The LAPACK timing programs were designed to collect data for all of the routines in Table 6.1. The range of problem sizes needed to determine the optimal block size or crossover point is machine-dependent, but the input files provided with the LAPACK test and timing package can be used

Figure 6.1: QR factorization on CRAY-2 (1 processor)

as a starting point. For subroutines that require a crossover point, it is best to start by finding the best block size with the crossover point set to 0, and then to locate the point at which the performance of the unblocked algorithm is beaten by the block algorithm. The best crossover point will be somewhat smaller than the point where the curves for the unblocked and blocked methods cross.

For example, for SGEQRF on a single processor of a CRAY-2, NB = 32 was observed to be a good block size, and the performance of the block algorithm with this block size surpasses the unblocked algorithm for square matrices between $N = 176$ and $N = 192$. Experiments with crossover points from 64 to 192 found that NX = 128 was a good choice, although the results for NX from 3*NB to 5*NB are broadly similar. This means that matrices with $N \leq 128$ should use the unblocked algorithm, and for $N > 128$ block updates should be used until the remaining submatrix has order less than 128. The performance of the unblocked (NB = 1) and blocked (NB = 32) algorithms for SGEQRF and for the blocked algorithm with a crossover point of 128 are compared in Figure 6.1.

By experimenting with small values of the block size, it should be straightforward to choose NBMIN, the smallest block size that gives a performance improvement over the unblocked algorithm. Note that on some machines, the optimal block size may be 1 (the unblocked algorithm gives the best performance); in this case, the choice of NBMIN is arbitrary. The prototype version of ILAENV sets NBMIN to 2, so that blocking is always done, even though this could lead to poor performance from a block routine if insufficient workspace is supplied (see chapter 7).

Complicating the determination of optimal parameters is the fact that the orthogonal factorization routines and SGEBRD accept non-square matrices as input. The LAPACK timing program allows M and N to be varied independently. We have found the optimal block size to be generally insensitive to the shape of the matrix, but the crossover point is more dependent on the matrix shape. For example, if $M >> N$ in the QR factorization, block updates may always be faster than

unblocked updates on the remaining submatrix, so one might set NX = NB if $M \geq 2N$.

Parameter values for the number of shifts, etc. used to tune the block multishift QR algorithm can be varied from the input files to the eigenvalue timing program. In particular, the performance of xHSEQR is particularly sensitive to the correct choice of block parameters. Setting NS = 2 will give essentially the same performance as EISPACK. Interested users should consult [3] for a description of the timing program input files.

Chapter 7

Troubleshooting

Successful use of LAPACK relies heavily on proper installation and testing of LAPACK and the BLAS. Frequently Asked Questions (FAQ) lists are maintained in the *blas* and *lapack* directories on *netlib* to answer some of the most common user questions. For the user's convenience, prebuilt LAPACK libraries are provided for a variety of computer architectures in the following URL:

`http://www.netlib.org/lapack/archives/`

Test suites are provided for the BLAS and LAPACK. It is highly recommended that each of these respective test suites be run prior to the use of the library. A comprehensive installation guide is provided for LAPACK. Refer to the *lapack* directory on *netlib* for further information.

We begin this chapter by discussing a set of first-step debugging hints to pinpoint where the problem is occurring. Following these debugging hints, we discuss the types of error messages that can be encountered during the execution of an LAPACK routine.

If these suggestions do not help evaluate specific difficulties, we suggest that the user review the following "bug report checklist" and then feel free to contact the authors at `lapack@cs.utk.edu`. The user should tell us the type of machine on which the tests were run, the compiler and compiler options that were used, and details of the BLAS library; also, the user should send a copy of the input file, if appropriate.

Bug Report Checklist

When the user sends e-mail to our mailing alias, some of the first questions we will ask are the following:

1. Have you run the BLAS and LAPACK test suites?
2. Have you checked the errata list (`release_notes`) on *netlib*?

   `http://www.netlib.org/lapack/release_notes`

3. If you are using an optimized BLAS library, have you tried using the reference implementation from *netlib*?

   `http://www.netlib.org/blas/`

4. Have you attempted to replicate this error using the appropriate LAPACK test code?

7.1 Installation Debugging Hints

If the user encounters difficulty in the installation process, we suggest the following:

- Obtain prebuilt LAPACK library on *netlib* for a variety of architectures.

  `http://www.netlib.org/lapack/archives/`

- Obtain sample `make.inc` files for a variety of architectures in the `LAPACK/INSTALL` directory in the lapack distribution tar file.
- Consult the LAPACK FAQ list on *netlib*.
- Consult the `release_notes` file in the *lapack* directory on *netlib*. This file contains a list of known difficulties that have been diagnosed and corrected (or will be corrected in the next release), or reported to the vendor as in the case of optimized BLAS.
- Always run the BLAS test suite to ensure that this library has been properly installed. If a problem is detected in the BLAS library, try linking to the reference implementation in the *blas* directory on *netlib*.

7.2 Common Errors in Calling LAPACK Routines

As a first step, the user should always carefully read the leading comments of the LAPACK routine. The leading comments give a detailed description of all input/output arguments and can be viewed in the source code, on the LAPACK webpage, or in this users' guide.

For the benefit of less experienced programmers, we list the most common programming errors in calling an LAPACK routine. These errors may cause the LAPACK routine to report a failure, as described in Section 7.3; they may cause an error to be reported by the system; or they may lead to wrong results — see also Section 7.4.

- wrong number of arguments
- arguments in the wrong order
- an argument of the wrong type (especially real and complex arguments of the wrong precision)
- wrong dimensions for an array argument

- insufficient space in a workspace argument
- failure to assign a value to an input argument

Some modern compilation systems, as well as software tools such as the portability checker in Toolpack [88], can check that arguments agree in number and type; and many compilation systems offer run-time detection of errors such as an array element out-of-bounds or use of an unassigned variable.

7.3 Failures Detected by LAPACK Routines

There are two ways in which an LAPACK routine may report a failure to complete a computation successfully.

7.3.1 Invalid Arguments and XERBLA

If an illegal value is supplied for one of the input arguments to an LAPACK routine, it will call the error handler XERBLA to write a message to the standard output unit of the form:

```
** On entry to SGESV  parameter number  4 had an illegal value
```

This particular message would be caused by passing to SGESV a value of LDA which was less than the value of the argument N. The documentation for SGESV in Part 2 states the set of acceptable input values: "LDA $\geq$ max(1,N)." This is required in order that the array A with leading dimension LDA can store an n-by-n matrix.[1] The arguments are checked in order, beginning with the first. In the above example, it may — from the user's point of view — be the value of N which is in fact wrong. Invalid arguments are often caused by the kind of error listed in Section 7.2.

In the model implementation of XERBLA which is supplied with LAPACK, execution stops after the message; but the call to XERBLA is followed by a RETURN statement in the LAPACK routine, so that if the installer removes the STOP statement in XERBLA, the result will be an immediate exit from the LAPACK routine with a negative value of INFO. It is good practice always to check for a non-zero value of INFO on return from an LAPACK routine. (We recommend however that XERBLA should not be modified to return control to the calling routine, unless absolutely necessary, since this would remove one of the built-in safety-features of LAPACK.)

7.3.2 Computational Failures and INFO > 0

A positive value of INFO on return from an LAPACK routine indicates a failure in the course of the algorithm. Common causes are:

[1] The requirement is stated "LDA $\geq$ max(1,N)" rather than simply "LDA $\geq$ N" because LDA must always be at least 1, even if N = 0, to satisfy the requirements of standard Fortran; on some systems, a zero or negative value of LDA would cause a run-time fault.

- a matrix is singular (to working precision);
- a symmetric matrix is not positive definite;
- an iterative algorithm for computing eigenvalues or eigenvectors fails to converge in the permitted number of iterations.

For example, if SGESVX is called to solve a system of equations with a coefficient matrix that is approximately singular, it may detect exact singularity at the ith stage of the LU factorization, in which case it returns INFO $= i$; or (more probably) it may compute an estimate of the reciprocal condition number that is less than machine precision, in which case it returns INFO $= n+1$. Again, the documentation in Part 2 should be consulted for a description of the error.

When a failure with INFO > 0 occurs, control is *always* returned to the calling program; XERBLA is *not* called, and no error message is written. It is worth repeating that it is good practice always to check for a non-zero value of INFO on return from an LAPACK routine.

A failure with INFO > 0 may indicate any of the following:

- an inappropriate routine was used: for example, if a routine fails because a symmetric matrix turns out not to be positive definite, consider using a routine for symmetric indefinite matrices.
- a single precision routine was used when double precision was needed: for example, if SGESVX reports approximate singularity (as illustrated above), the corresponding double precision routine DGESVX may be able to solve the problem (but nevertheless the problem is ill-conditioned).
- a programming error occurred in generating the data supplied to a routine: for example, even though theoretically a matrix should be well-conditioned and positive-definite, a programming error in generating the matrix could easily destroy either of those properties.
- a programming error occurred in calling the routine, of the kind listed in Section 7.2.

7.4 Wrong Results

Wrong results from LAPACK routines are most often caused by incorrect usage.

It is also possible that wrong results are caused by a bug outside of LAPACK, in the compiler or in one of the library routines, such as the BLAS, that are linked with LAPACK. Test procedures are available for both LAPACK and the BLAS, and the LAPACK installation guide [3] should be consulted for descriptions of the tests and for advice on resolving problems.

A list of known problems, compiler errors, and bugs in LAPACK routines is maintained on *netlib*; see Chapter 1.

Users who suspect they have found a new bug in an LAPACK routine are encouraged to report it promptly to the developers as directed in Chapter 1. The bug report should include a test case, a description of the problem and expected results, and the actions, if any, that the user has already taken to fix the bug.

7.5 Poor Performance

LAPACK relies on an efficient implementation of the BLAS. We have tried to make the performance of LAPACK "transportable" by performing most of the computation within the Level 1, 2, and 3 BLAS, and by isolating all of the machine-dependent tuning parameters in a single integer function ILAENV.

To avoid poor performance from LAPACK routines, note the following recommendations:

BLAS: One should use machine-specific optimized BLAS if they are available. Many manufacturers and research institutions have developed, or are developing, efficient versions of the BLAS for particular machines. The BLAS enable LAPACK routines to achieve high performance with transportable software. Users are urged to determine whether such an implementation of the BLAS exists for their platform. When such an optimized implementation of the BLAS is available, it should be used to ensure optimal performance. If such a machine-specific implementation of the BLAS does not exist for a particular platform, one should consider installing a publicly available set of BLAS that requires only an efficient implementation of the matrix-matrix multiply BLAS routine xGEMM. Examples of such implementations are [21, 72]. A machine-specific and efficient implementation of the routine GEMM can be automatically generated by publicly available software such as [102] and [15]. Although a reference implementation of the Fortran77 BLAS is available from the *blas* directory on *netlib*, these routines are not expected to perform as well as a specially tuned implementation on most high-performance computers – on some machines it may give much worse performance – but it allows users to run LAPACK software on machines that do not offer any other implementation of the BLAS.

ILAENV: For best performance, the LAPACK routine ILAENV should be set with optimal tuning parameters for the machine being used. The version of ILAENV provided with LAPACK supplies default values for these parameters that give good, but not optimal, average case performance on a range of existing machines. In particular, the performance of xHSEQR is particularly sensitive to the correct choice of block parameters; the same applies to the driver routines which call xHSEQR, namely xGEES, xGEESX, xGEEV and xGEEVX. Further details on setting parameters in ILAENV are found in section 6.

LWORK $\geq$ WORK(1): The performance of some routines depends on the amount of workspace supplied. In such cases, an argument, usually called WORK, is provided, accompanied by an integer argument LWORK specifying its length as a linear array. On exit, WORK(1) returns the amount of workspace required to use the optimal tuning parameters. If LWORK $<$ WORK(1), then insufficient workspace was provided to use the optimal parameters, and the performance may be less than possible. One should check LWORK $\geq$ WORK(1) on return from an LAPACK routine requiring user-supplied workspace to see if enough workspace has been provided. Note that the computation is performed correctly, even if the amount of workspace is less than optimal, unless LWORK is reported as an invalid value by a call to XERBLA as described in Section 7.3.

xLAMCH: Users should beware of the high cost of the *first* call to the LAPACK auxiliary routine

xLAMCH, which computes machine characteristics such as epsilon and the smallest invertible number. The first call dynamically determines a set of parameters defining the machine's arithmetic, but these values are saved and subsequent calls incur only a trivial cost. For performance testing, the initial cost can be hidden by including a call to xLAMCH in the main program, before any calls to LAPACK routines that will be timed. A sample use of SLAMCH is

```
XXXXXX = SLAMCH( 'P' )
```

or in double precision:

```
XXXXXX = DLAMCH( 'P' )
```

A cleaner but less portable solution is for the installer to save the values computed by xLAMCH for a specific machine and create a new version of xLAMCH with these constants set in DATA statements, taking care that no accuracy is lost in the translation.

Appendix A

Index of Driver and Computational Routines

Notes

1. This index lists related pairs of real and complex routines together, for example, SBDSQR and CBDSQR.

2. Driver routines are listed in bold type, for example **SGBSV** and **CGBSV**.

3. Routines are listed in alphanumeric order of the real (single precision) routine name (which always begins with S-). (See subsection 2.2.3 for details of the LAPACK naming scheme.)

4. Double precision routines are not listed here; they have names beginning with D- instead of S-, or Z- instead of C-.

5. This index gives only a brief description of the purpose of each routine. For a precise description, consult the specifications in Part 2, where the routines appear in the same order as here.

6. The text of the descriptions applies to both real and complex routines, except where alternative words or phrases are indicated, for example "symmetric/Hermitian", "orthogonal/unitary" or "quasi-triangular/triangular". For the real routines A^H is equivalent to A^T. (The same convention is used in Part 2.)

7. In a few cases, three routines are listed together, one for real symmetric, one for complex symmetric, and one for complex Hermitian matrices (for example SSPCON, CSPCON and CHPCON).

8. A few routines for real matrices have no complex equivalent (for example SSTEBZ).

Routine		Description
real	complex	
SBDSDC		Computes the singular value decomposition (SVD) of a real bidiagonal matrix, using a divide and conquer method.
SBDSQR	CBDSQR	Computes the singular value decomposition (SVD) of a real bidiagonal matrix, using the bidiagonal QR algorithm.
SDISNA		Computes the reciprocal condition numbers for the eigenvectors of a real symmetric or complex Hermitian matrix or for the left or right singular vectors of a general matrix.
SGBBRD	CGBBRD	Reduces a general band matrix to real upper bidiagonal form by an orthogonal/unitary transformation.
SGBCON	CGBCON	Estimates the reciprocal of the condition number of a general band matrix, in either the 1-norm or the infinity-norm, using the LU factorization computed by SGBTRF/CGBTRF.
SGBEQU	CGBEQU	Computes row and column scalings to equilibrate a general band matrix and reduce its condition number.
SGBRFS	CGBRFS	Improves the computed solution to a general banded system of linear equations $AX = B$, $A^TX = B$ or $A^HX = B$, and provides forward and backward error bounds for the solution.
SGBSV	**CGBSV**	Solves a general banded system of linear equations $AX = B$.
SGBSVX	**CGBSVX**	Solves a general banded system of linear equations $AX = B$, $A^TX = B$ or $A^HX = B$, and provides an estimate of the condition number and error bounds on the solution.
SGBTRF	CGBTRF	Computes an LU factorization of a general band matrix, using partial pivoting with row interchanges.
SGBTRS	CGBTRS	Solves a general banded system of linear equations $AX = B$, $A^TX = B$ or $A^HX = B$, using the LU factorization computed by SGBTRF/CGBTRF.
SGEBAK	CGEBAK	Transforms eigenvectors of a balanced matrix to those of the original matrix supplied to SGEBAL/CGEBAL.
SGEBAL	CGEBAL	Balances a general matrix in order to improve the accuracy of computed eigenvalues.
SGEBRD	CGEBRD	Reduces a general rectangular matrix to real bidiagonal form by an orthogonal/unitary transformation.
SGECON	CGECON	Estimates the reciprocal of the condition number of a general matrix, in either the 1-norm or the infinity-norm, using the LU factorization computed by SGETRF/CGETRF.
SGEEQU	CGEEQU	Computes row and column scalings to equilibrate a general rectangular matrix and reduce its condition number.
SGEES	**CGEES**	Computes the eigenvalues and Schur factorization of a general matrix, and orders the factorization so that selected eigenvalues are at the top left of the Schur form.

Routine		Description
real	complex	
SGEESX	**CGEESX**	Computes the eigenvalues and Schur factorization of a general matrix, orders the factorization so that selected eigenvalues are at the top left of the Schur form, and computes reciprocal condition numbers for the average of the selected eigenvalues, and for the associated right invariant subspace.
SGEEV	**CGEEV**	Computes the eigenvalues and left and right eigenvectors of a general matrix.
SGEEVX	**CGEEVX**	Computes the eigenvalues and left and right eigenvectors of a general matrix, with preliminary balancing of the matrix, and computes reciprocal condition numbers for the eigenvalues and right eigenvectors.
SGEHRD	CGEHRD	Reduces a general matrix to upper Hessenberg form by an orthogonal/unitary similarity transformation.
SGELQF	CGELQF	Computes an LQ factorization of a general rectangular matrix.
SGELS	**CGELS**	Computes the least squares solution to an overdetermined system of linear equations, $AX = B$ or $A^H X = B$, or the minimum norm solution of an underdetermined system, where A is a general rectangular matrix of full rank, using a QR or LQ factorization of A.
SGELSD	**CGELSD**	Computes the minimum norm least squares solution to an over- or underdetermined system of linear equations $AX = B$, using the singular value decomposition of A and a divide and conquer method.
SGELSS	**CGELSS**	Computes the minimum norm least squares solution to an over- or underdetermined system of linear equations $AX = B$, using the singular value decomposition of A.
SGELSY	**CGELSY**	Computes the minimum norm least squares solution to an over- or underdetermined system of linear equations $AX = B$, using a complete orthogonal factorization of A via xGEQP3.
SGEQLF	CGEQLF	Computes a QL factorization of a general rectangular matrix.
SGEQP3	CGEQP3	Computes a QR factorization with column pivoting of a general rectangular matrix using Level 3 BLAS.
SGEQRF	CGEQRF	Computes a QR factorization of a general rectangular matrix.
SGERFS	CGERFS	Improves the computed solution to a general system of linear equations $AX = B$, $A^T X = B$ or $A^H X = B$, and provides forward and backward error bounds for the solution.
SGERQF	CGERQF	Computes an RQ factorization of a general rectangular matrix.
SGESDD	**CGESDD**	Computes the singular value decomposition (SVD) of a general rectangular matrix using a divide and conquer method.
SGESV	**CGESV**	Solves a general system of linear equations $AX = B$.

Routine		Description
real	complex	
SGESVD	**CGESVD**	Computes the singular value decomposition (SVD) of a general rectangular matrix.
SGESVX	**CGESVX**	Solves a general system of linear equations $AX = B$, $A^TX = B$ or $A^HX = B$, and provides an estimate of the condition number and error bounds on the solution.
SGETRF	CGETRF	Computes an LU factorization of a general matrix, using partial pivoting with row interchanges.
SGETRI	CGETRI	Computes the inverse of a general matrix, using the LU factorization computed by SGETRF/CGETRF.
SGETRS	CGETRS	Solves a general system of linear equations $AX = B$, $A^TX = B$ or $A^HX = B$, using the LU factorization computed by SGETRF/CGETRF.
SGGBAK	CGGBAK	Forms the right or left eigenvectors of a real generalized eigenvalue problem $Ax = \lambda Bx$, by backward transformation on the computed eigenvectors of the balanced pair of matrices output by SGGBAL/CGGBAL.
SGGBAL	CGGBAL	Balances a pair of general matrices to improve the accuracy of computed eigenvalues and/or eigenvectors.
SGGES	**CGGES**	Computes the generalized eigenvalues, Schur form, and the left and/or right Schur vectors for a pair of nonsymmetric matrices.
SGGESX	**CGGESX**	Computes the generalized eigenvalues, Schur form, and, optionally, the left and/or right matrices of Schur vectors.
SGGEV	**CGGEV**	Computes the generalized eigenvalues and the left and/or right generalized eigenvectors for a pair of nonsymmetric matrices.
SGGEVX	**CGGEVX**	Computes the generalized eigenvalues and, optionally, the left and/or right generalized eigenvectors.
SGGGLM	**CGGGLM**	Solves a general Gauss-Markov linear model (GLM) problem using a generalized QR factorization.
SGGHRD	CGGHRD	Reduces a pair of matrices to generalized upper Hessenberg form using orthogonal/unitary transformations.
SGGLSE	**CGGLSE**	Solves the linear equality-constrained least squares (LSE) problem using a generalized RQ factorization.
SGGQRF	CGGQRF	Computes a generalized QR factorization of a pair of matrices.
SGGRQF	CGGRQF	Computes a generalized RQ factorization of a pair of matrices.
SGGSVD	**CGGSVD**	Computes the generalized singular value decomposition (GSVD) of a pair of general rectangular matrices.
SGGSVP	CGGSVP	Computes orthogonal/unitary matrices U, V, and Q as the preprocessing step for computing the generalized singular value decomposition (GSVD).

Routine		Description
real	complex	
SGTCON	CGTCON	Estimates the reciprocal of the condition number of a general tridiagonal matrix, in either the 1-norm or the infinity-norm, using the LU factorization computed by SGTTRF/CGTTRF.
SGTRFS	CGTRFS	Improves the computed solution to a general tridiagonal system of linear equations $AX = B$, $A^TX = B$ or $A^HX = B$, and provides forward and backward error bounds for the solution.
SGTSV	**CGTSV**	Solves a general tridiagonal system of linear equations $AX = B$.
SGTSVX	**CGTSVX**	Solves a general tridiagonal system of linear equations $AX = B$, $A^TX = B$ or $A^HX = B$, and provides an estimate of the condition number and error bounds on the solution.
SGTTRF	CGTTRF	Computes an LU factorization of a general tridiagonal matrix, using partial pivoting with row interchanges.
SGTTRS	CGTTRS	Solves a general tridiagonal system of linear equations $AX = B$, $A^TX = B$ or $A^HX = B$, using the LU factorization computed by SGTTRF/CGTTRF.
SHGEQZ	CHGEQZ	Implements a single-/double-shift version of the QZ method for finding the generalized eigenvalues of a pair of general matrices, which can simultaneously be reduced to generalized Schur form using orthogonal/unitary transformations.
SHSEIN	CHSEIN	Computes specified right and/or left eigenvectors of an upper Hessenberg matrix by inverse iteration.
SHSEQR	CHSEQR	Computes the eigenvalues and Schur factorization of an upper Hessenberg matrix, using the multishift QR algorithm.
SOPGTR	CUPGTR	Generates the orthogonal/unitary transformation matrix from a reduction to tridiagonal form determined by SSPTRD/CHPTRD.
SOPMTR	CUPMTR	Multiplies a general matrix by the orthogonal/unitary transformation matrix from a reduction to tridiagonal form determined by SSPTRD/CHPTRD.
SORGBR	CUNGBR	Generates the orthogonal/unitary transformation matrices from a reduction to bidiagonal form determined by SGEBRD/CGEBRD.
SORGHR	CUNGHR	Generates the orthogonal/unitary transformation matrix from a reduction to Hessenberg form determined by SGEHRD/CGEHRD.
SORGLQ	CUNGLQ	Generates all or part of the orthogonal/unitary matrix Q from an LQ factorization determined by SGELQF/CGELQF.
SORGQL	CUNGQL	Generates all or part of the orthogonal/unitary matrix Q from a QL factorization determined by SGEQLF/CGEQLF.
SORGQR	CUNGQR	Generates all or part of the orthogonal/unitary matrix Q from a QR factorization determined by SGEQRF/CGEQRF.
SORGRQ	CUNGRQ	Generates all or part of the orthogonal/unitary matrix Q from an RQ factorization determined by SGERQF/CGERQF.
SORGTR	CUNGTR	Generates the orthogonal/unitary transformation matrix from a reduction to tridiagonal form determined by SSYTRD/CHETRD.

Routine		Description
real	complex	
SORMBR	CUNMBR	Multiplies a general matrix by one of the orthogonal/unitary transformation matrices from a reduction to bidiagonal form determined by SGEBRD/CGEBRD.
SORMHR	CUNMHR	Multiplies a general matrix by the orthogonal/unitary transformation matrix from a reduction to Hessenberg form determined by SGEHRD/CGEHRD.
SORMLQ	CUNMLQ	Multiplies a general matrix by the orthogonal/unitary matrix from an LQ factorization determined by SGELQF/CGELQF.
SORMQL	CUNMQL	Multiplies a general matrix by the orthogonal/unitary matrix from a QL factorization determined by SGEQLF/CGEQLF.
SORMQR	CUNMQR	Multiplies a general matrix by the orthogonal/unitary matrix from a QR factorization determined by SGEQRF/CGEQRF.
SORMRQ	CUNMRQ	Multiplies a general matrix by the orthogonal/unitary matrix from an RQ factorization determined by SGERQF/CGERQF.
SORMRZ	CUNMRZ	Multiplies a general matrix by the orthogonal/unitary matrix from an RZ factorization determined by STZRZF/CTZRZF.
SORMTR	CUNMTR	Multiplies a general matrix by the orthogonal/unitary transformation matrix from a reduction to tridiagonal form determined by SSYTRD/CHETRD.
SPBCON	CPBCON	Estimates the reciprocal of the condition number of a symmetric/Hermitian positive definite band matrix, using the Cholesky factorization computed by SPBTRF/CPBTRF.
SPBEQU	CPBEQU	Computes row and column scalings to equilibrate a symmetric/Hermitian positive definite band matrix and reduce its condition number.
SPBRFS	CPBRFS	Improves the computed solution to a symmetric/Hermitian positive definite banded system of linear equations $AX = B$, and provides forward and backward error bounds for the solution.
SPBSTF	CPBSTF	Computes a split Cholesky factorization of a real/complex symmetric/Hermitian positive definite band matrix.
SPBSV	**CPBSV**	Solves a symmetric/Hermitian positive definite banded system of linear equations $AX = B$.
SPBSVX	**CPBSVX**	Solves a symmetric/Hermitian positive definite banded system of linear equations $AX = B$, and provides an estimate of the condition number and error bounds on the solution.
SPBTRF	CPBTRF	Computes the Cholesky factorization of a symmetric/Hermitian positive definite band matrix.
SPBTRS	CPBTRS	Solves a symmetric/Hermitian positive definite banded system of linear equations $AX = B$, using the Cholesky factorization computed by SPBTRF/CPBTRF.
SPOCON	CPOCON	Estimates the reciprocal of the condition number of a symmetric/Hermitian positive definite matrix, using the Cholesky factorization computed by SPOTRF/CPOTRF.

Routine		Description
real	complex	
SPOEQU	CPOEQU	Computes row and column scalings to equilibrate a symmetric/Hermitian positive definite matrix and reduce its condition number.
SPORFS	CPORFS	Improves the computed solution to a symmetric/Hermitian positive definite system of linear equations $AX = B$, and provides forward and backward error bounds for the solution.
SPOSV	**CPOSV**	Solves a symmetric/Hermitian positive definite system of linear equations $AX = B$.
SPOSVX	**CPOSVX**	Solves a symmetric/Hermitian positive definite system of linear equations $AX = B$, and provides an estimate of the condition number and error bounds on the solution.
SPOTRF	CPOTRF	Computes the Cholesky factorization of a symmetric/Hermitian positive definite matrix.
SPOTRI	CPOTRI	Computes the inverse of a symmetric/Hermitian positive definite matrix, using the Cholesky factorization computed by SPOTRF/CPOTRF.
SPOTRS	CPOTRS	Solves a symmetric/Hermitian positive definite system of linear equations $AX = B$, using the Cholesky factorization computed by SPOTRF/CPOTRF.
SPPCON	CPPCON	Estimates the reciprocal of the condition number of a symmetric/Hermitian positive definite matrix in packed storage, using the Cholesky factorization computed by SPPTRF/CPPTRF.
SPPEQU	CPPEQU	Computes row and column scalings to equilibrate a symmetric/Hermitian positive definite matrix in packed storage and reduce its condition number.
SPPRFS	CPPRFS	Improves the computed solution to a symmetric/Hermitian positive definite system of linear equations $AX = B$, where A is held in packed storage, and provides forward and backward error bounds for the solution.
SPPSV	**CPPSV**	Solves a symmetric/Hermitian positive definite system of linear equations $AX = B$, where A is held in packed storage.
SPPSVX	**CPPSVX**	Solves a symmetric/Hermitian positive definite system of linear equations $AX = B$, where A is held in packed storage, and provides an estimate of the condition number and error bounds on the solution.
SPPTRF	CPPTRF	Computes the Cholesky factorization of a symmetric/Hermitian positive definite matrix in packed storage.
SPPTRI	CPPTRI	Computes the inverse of a symmetric/Hermitian positive definite matrix in packed storage, using the Cholesky factorization computed by SPPTRF/CPPTRF.
SPPTRS	CPPTRS	Solves a symmetric/Hermitian positive definite system of linear equations $AX = B$, where A is held in packed storage, using the Cholesky factorization computed by SPPTRF/CPPTRF.

Routine		Description
real	complex	
SPTCON	CPTCON	Computes the reciprocal of the condition number of a symmetric/Hermitian positive definite tridiagonal matrix, using the LDL^H factorization computed by SPTTRF/CPTTRF.
SPTEQR	CPTEQR	Computes all eigenvalues and eigenvectors of a real symmetric positive definite tridiagonal matrix, by computing the SVD of its bidiagonal Cholesky factor.
SPTRFS	CPTRFS	Improves the computed solution to a symmetric/Hermitian positive definite tridiagonal system of linear equations $AX = B$, and provides forward and backward error bounds for the solution.
SPTSV	**CPTSV**	Solves a symmetric/Hermitian positive definite tridiagonal system of linear equations $AX = B$.
SPTSVX	**CPTSVX**	Solves a symmetric/Hermitian positive definite tridiagonal system of linear equations $AX = B$, and provides an estimate of the condition number and error bounds on the solution.
SPTTRF	CPTTRF	Computes the LDL^H factorization of a symmetric/Hermitian positive definite tridiagonal matrix.
SPTTRS	CPTTRS	Solves a symmetric/Hermitian positive definite tridiagonal system of linear equations, using the LDL^H factorization computed by SPTTRF/CPTTRF.
SSBEV	**CHBEV**	Computes all eigenvalues and, optionally, eigenvectors of a symmetric/Hermitian band matrix.
SSBEVD	**CHBEVD**	Computes all eigenvalues and, optionally, eigenvectors of a symmetric/Hermitian band matrix. If eigenvectors are desired, it uses a divide and conquer algorithm.
SSBEVX	**CHBEVX**	Computes selected eigenvalues and eigenvectors of a symmetric/Hermitian band matrix.
SSBGST	CHBGST	Reduces a real/complex symmetric-/Hermitian-definite banded generalized eigenproblem $Ax = \lambda Bx$ to standard form, where B has been factorized by SPBSTF/CPBSTF (Crawford's algorithm).
SSBGV	**CHBGV**	Computes all of the eigenvalues, and optionally, the eigenvectors of a real/complex generalized symmetric-/Hermitian-definite banded eigenproblem $Ax = \lambda Bx$.
SSBGVD	**CHBGVD**	Computes all eigenvalues, and optionally, the eigenvectors of a real/complex generalized symmetric-/Hermitian-definite banded eigenproblem $Ax = \lambda Bx$. If eigenvectors are desired, it uses a divide and conquer algorithm.
SSBGVX	**CHBGVX**	Computes selected eigenvalues, and optionally, the eigenvectors of a real/complex generalized symmetric-/Hermitian-definite banded eigenproblem $Ax = \lambda Bx$.
SSBTRD	CHBTRD	Reduces a symmetric/Hermitian band matrix to real symmetric tridiagonal form by an orthogonal/unitary similarity transformation.

Routine		Description
real	complex	
SSPCON	CSPCON CHPCON	Estimates the reciprocal of the condition number of a real symmetric/complex symmetric/complex Hermitian indefinite matrix in packed storage, using the factorization computed by SSPTRF/CSPTRF/CHPTRF.
SSPEV	**CHPEV**	Computes all eigenvalues and, optionally, eigenvectors of a symmetric/Hermitian matrix in packed storage.
SSPEVD	**CHPEVD**	Computes all eigenvalues and, optionally, eigenvectors of a symmetric/Hermitian matrix in packed storage. If eigenvectors are desired, it uses a divide and conquer algorithm.
SSPEVX	**CHPEVX**	Computes selected eigenvalues and eigenvectors of a symmetric/Hermitian matrix in packed storage.
SSPGST	CHPGST	Reduces a symmetric/Hermitian definite generalized eigenproblem $Ax = \lambda Bx$, $ABx = \lambda x$, or $BAx = \lambda x$, to standard form, where A and B are held in packed storage, and B has been factorized by SPPTRF/CPPTRF.
SSPGV	**CHPGV**	Computes all eigenvalues and optionally, the eigenvectors of a generalized symmetric/Hermitian definite generalized eigenproblem, $Ax = \lambda Bx$, $ABx = \lambda x$, or $BAx = \lambda x$, where A and B are in packed storage.
SSPGVD	**CHPGVD**	Computes all eigenvalues, and optionally, the eigenvectors of a generalized symmetric/Hermitian definite generalized eigenproblem, $Ax = \lambda Bx$, $ABx = \lambda x$, or $BAx = \lambda x$, where A and B are in packed storage. If eigenvectors are desired, it uses a divide and conquer algorithm.
SSPGVX	**CHPGVX**	Computes selected eigenvalues, and optionally, the eigenvectors of a generalized symmetric/Hermitian definite generalized eigenproblem, $Ax = \lambda Bx$, $ABx = \lambda x$, or $BAx = \lambda x$, where A and B are in packed storage.
SSPRFS	CSPRFS CHPRFS	Improves the computed solution to a real symmetric/complex symmetric/complex Hermitian indefinite system of linear equations $AX = B$, where A is held in packed storage, and provides forward and backward error bounds for the solution.
SSPSV	**CSPSV** **CHPSV**	Solves a real symmetric/complex symmetric/complex Hermitian indefinite system of linear equations $AX = B$, where A is held in packed storage.
SSPSVX	**CSPSVX** **CHPSVX**	Solves a real symmetric/complex symmetric/complex Hermitian indefinite system of linear equations $AX = B$, where A is held in packed storage, and provides an estimate of the condition number and error bounds on the solution.
SSPTRD	CHPTRD	Reduces a symmetric/Hermitian matrix in packed storage to real symmetric tridiagonal form by an orthogonal/unitary similarity transformation.

Routine		Description
real	complex	
SSPTRF	CSPTRF CHPTRF	Computes the factorization of a real symmetric/complex symmetric/complex Hermitian indefinite matrix in packed storage, using the diagonal pivoting method.
SSPTRI	CSPTRI CHPTRI	Computes the inverse of a real symmetric/complex symmetric/complex Hermitian indefinite matrix in packed storage, using the factorization computed by SSPTRF/CSPTRF/CHPTRF.
SSPTRS	CSPTRS CHPTRS	Solves a real symmetric/complex symmetric/complex Hermitian indefinite system of linear equations $AX = B$, where A is held in packed storage, using the factorization computed by SSPTRF/CSPTRF/CHPTRF.
SSTEBZ		Computes selected eigenvalues of a real symmetric tridiagonal matrix by bisection.
SSTEDC	CSTEDC	Computes all eigenvalues and, optionally, eigenvectors of a symmetric tridiagonal matrix using the divide and conquer algorithm.
SSTEGR	CSTEGR	Computes selected eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix using the Relatively Robust Representations.
SSTEIN	CSTEIN	Computes selected eigenvectors of a real symmetric tridiagonal matrix by inverse iteration.
SSTEQR	CSTEQR	Computes all eigenvalues and eigenvectors of a real symmetric tridiagonal matrix, using the implicit QL or QR algorithm.
SSTERF		Computes all eigenvalues of a real symmetric tridiagonal matrix, using a root-free variant of the QL or QR algorithm.
SSTEV		Computes all eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix.
SSTEVD		Computes all eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix. If eigenvectors are desired, it uses a divide and conquer algorithm.
SSTEVR		Computes selected eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix using the Relatively Robust Representations.
SSTEVX		Computes selected eigenvalues and eigenvectors of a real symmetric tridiagonal matrix.
SSYCON	CSYCON CHECON	Estimates the reciprocal of the condition number of a real symmetric/complex symmetric/complex Hermitian indefinite matrix, using the factorization computed by SSYTRF/CSYTRF/CHETRF.
SSYEV	**CHEEV**	Computes all eigenvalues and, optionally, eigenvectors of a symmetric/Hermitian matrix.
SSYEVD	**CHEEVD**	Computes all eigenvalues and, optionally, eigenvectors of a symmetric/Hermitian matrix. If eigenvectors are desired, it uses a divide and conquer algorithm.

Routine		Description
real	complex	
SSYEVR	**CHEEVR**	Computes selected eigenvalues and, optionally, eigenvectors of a real symmetric/Hermitian matrix using the Relatively Robust Representations.
SSYEVX	**CHEEVX**	Computes selected eigenvalues and, optionally, eigenvectors of a symmetric/Hermitian matrix.
SSYGST	CHEGST	Reduces a symmetric/Hermitian definite generalized eigenproblem $Ax = \lambda Bx$, $ABx = \lambda x$, or $BAx = \lambda x$, to standard form, where B has been factorized by SPOTRF/CPOTRF.
SSYGV	**CHEGV**	Computes all eigenvalues, and optionally, the eigenvectors of a generalized symmetric/Hermitian definite generalized eigenproblem, $Ax = \lambda Bx$, $ABx = \lambda x$, or $BAx = \lambda x$.
SSYGVD	**CHEGVD**	Computes all eigenvalues, and optionally, the eigenvectors of a generalized symmetric/Hermitian definite generalized eigenproblem, $Ax = \lambda Bx$, $ABx = \lambda x$, or $BAx = \lambda x$. If eigenvectors are desired, it uses a divide and conquer algorithm.
SSYGVX	**CHEGVX**	Computes selected eigenvalues, and optionally, the eigenvectors of a generalized symmetric/Hermitian definite generalized eigenproblem, $Ax = \lambda Bx$, $ABx = \lambda x$, or $BAx = \lambda x$.
SSYRFS	CSYRFS CHERFS	Improves the computed solution to a real symmetric/complex symmetric/complex Hermitian indefinite system of linear equations $AX = B$, and provides forward and backward error bounds for the solution.
SSYSV	**CSYSV** **CHESV**	Solves a real symmetric/complex symmetric/complex Hermitian indefinite system of linear equations $AX = B$.
SSYSVX	**CSYSVX** **CHESVX**	Solves a real symmetric/complex symmetric/complex Hermitian indefinite system of linear equations $AX = B$, and provides an estimate of the condition number and error bounds on the solution.
SSYTRD	CHETRD	Reduces a symmetric/Hermitian matrix to real symmetric tridiagonal form by an orthogonal/unitary similarity transformation.
SSYTRF	CSYTRF CHETRF	Computes the factorization of a real symmetric/complex symmetric/complex Hermitian indefinite matrix, using the diagonal pivoting method.
SSYTRI	CSYTRI CHETRI	Computes the inverse of a real symmetric/complex symmetric/complex Hermitian indefinite matrix, using the factorization computed by SSYTRF/CSYTRF/CHETRF.
SSYTRS	CSYTRS CHETRS	Solves a real symmetric/complex symmetric/complex Hermitian indefinite system of linear equations $AX = B$, using the factorization computed by SSPTRF/CSPTRF/CHPTRF.
STBCON	CTBCON	Estimates the reciprocal of the condition number of a triangular band matrix, in either the 1-norm or the infinity-norm.

Routine		Description
real	complex	
STBRFS	CTBRFS	Provides forward and backward error bounds for the solution of a triangular banded system of linear equations $AX = B$, $A^TX = B$ or $A^HX = B$.
STBTRS	CTBTRS	Solves a triangular banded system of linear equations $AX = B$, $A^TX = B$ or $A^HX = B$.
STGEVC	CTGEVC	Computes some or all of the right and/or left generalized eigenvectors of a pair of upper triangular matrices.
STGEXC	CTGEXC	Reorders the generalized real-Schur/Schur decomposition of a matrix pair (A, B) using an orthogonal/unitary equivalence transformation so that the diagonal block of (A, B) with row index $IFST$ is moved to row $ILST$.
STGSEN	CTGSEN	Reorders the generalized real-Schur/Schur decomposition of a matrix pair (A, B), computes the generalized eigenvalues of the reordered matrix pair, and, optionally, computes the estimates of reciprocal condition numbers for eigenvalues and eigenspaces.
STGSJA	CTGSJA	Computes the generalized singular value decomposition (GSVD) of a pair of upper triangular (or trapezoidal) matrices, which may be obtained by the preprocessing subroutine SGGSVP/CGGSVP.
STGSNA	CTGSNA	Estimates reciprocal condition numbers for specified eigenvalues and/or eigenvectors of a matrix pair (A, B) in generalized real-Schur/Schur canonical form
STGSYL	CTGSYL	Solves the generalized Sylvester equation
STPCON	CTPCON	Estimates the reciprocal of the condition number of a triangular matrix in packed storage, in either the 1-norm or the infinity-norm.
STPRFS	CTPRFS	Provides forward and backward error bounds for the solution of a triangular system of linear equations $AX = B$, $A^TX = B$ or $A^HX = B$, where A is held in packed storage.
STPTRI	CTPTRI	Computes the inverse of a triangular matrix in packed storage.
STPTRS	CTPTRS	Solves a triangular system of linear equations $AX = B$, $A^TX = B$ or $A^HX = B$, where A is held in packed storage.
STRCON	CTRCON	Estimates the reciprocal of the condition number of a triangular matrix, in either the 1-norm or the infinity-norm.
STREVC	CTREVC	Computes some or all of the right and/or left eigenvectors of an upper quasi-triangular/triangular matrix.
STREXC	CTREXC	Reorders the Schur factorization of a matrix by an orthogonal/unitary similarity transformation.
STRRFS	CTRRFS	Provides forward and backward error bounds for the solution of a triangular system of linear equations $AX = B$, $A^TX = B$ or $A^HX = B$.

Routine		Description
real	complex	
STRSEN	CTRSEN	Reorders the Schur factorization of a matrix in order to find an orthonormal basis of a right invariant subspace corresponding to selected eigenvalues, and returns reciprocal condition numbers (sensitivities) of the average of the cluster of eigenvalues and of the invariant subspace.
STRSNA	CTRSNA	Estimates the reciprocal condition numbers (sensitivities) of selected eigenvalues and eigenvectors of an upper quasi-triangular/triangular matrix.
STRSYL	CTRSYL	Solves the Sylvester matrix equation $AX \pm XB = C$ where A and B are upper quasi-triangular/triangular, and may be transposed.
STRTRI	CTRTRI	Computes the inverse of a triangular matrix.
STRTRS	CTRTRS	Solves a triangular system of linear equations $AX = B$, $A^T X = B$ or $A^H X = B$.
STZRZF	CTZRZF	Computes an RZ factorization of an upper trapezoidal matrix (blocked algorithm).

Appendix B

Index of Auxiliary Routines

Notes

1. This index lists related pairs of real and complex routines together, in the same style as in Appendix A.

2. Routines are listed in alphanumeric order of the real (single precision) routine name (which always begins with S-). (See subsection 2.2.3 for details of the LAPACK naming scheme.)

3. A few complex routines have no real equivalents, and they are listed first; routines listed in italics (for example, *CROT*), have real equivalents in the Level 1 or Level 2 BLAS.

4. Double precision routines are not listed here; they have names beginning with D- instead of S-, or Z- instead of C-. The only exceptions to this simple rule are that the double precision versions of ICMAX1, SCSUM1 and CSRSCL are named IZMAX1, DZSUM1 and ZDRSCL.

5. A few routines in the list have names that are independent of data type: ILAENV, LSAME, LSAMEN and XERBLA.

6. This index gives only a brief description of the purpose of each routine. For a precise description consult the leading comments in the code, which have been written in the same style as for the driver and computational routines.

Routine		Description
real	complex	
	CLACGV	Conjugates a complex vector.
	CLACRM	Performs a matrix multiplication $C = A * B$, where A is complex, B is real, and C is complex.
	CLACRT	Performs the transformation $\begin{pmatrix} c & s \\ -s & c \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$, where c, s, x, and y are complex.
	CLAESY	Computes the eigenvalues and eigenvectors of a 2-by-2 complex symmetric matrix, and checks that the norm of the matrix of eigenvectors is larger than a threshold value.
	CROT	Applies a plane rotation with real cosine and complex sine to a pair of complex vectors.
	CSPMV	Computes the matrix-vector product $y = \alpha Ax + \beta y$, where α and β are complex scalars, x and y are complex vectors and A is a complex symmetric matrix in packed storage.
	CSPR	Performs the symmetric rank-1 update $A = \alpha xx^T + A$, where α is a complex scalar, x is a complex vector and A is a complex symmetric matrix in packed storage.
	CSROT	Applies a plane rotation with real cosine and sine to a pair of complex vectors.
	CSYMV	Computes the matrix-vector product $y = \alpha Ax + \beta y$, where α and β are complex scalars, x and y are complex vectors and A is a complex symmetric matrix.
	CSYR	Performs the symmetric rank-1 update $A = \alpha xx^T + A$, where α is a complex scalar, x is a complex vector and A is a complex symmetric matrix.
	ICMAX1	Finds the index of the element whose real part has maximum absolute value (similar to the Level 1 BLAS ICAMAX, but using the absolute value of the real part).
ILAENV		Environmental enquiry function which returns values for tuning algorithmic performance.
LSAME		Tests two characters for equality regardless of case.
LSAMEN		Tests two character strings for equality regardless of case.
	SCSUM1	Forms the 1-norm of a complex vector (similar to the Level 1 BLAS SCASUM, but using the true absolute value).
SGBTF2	CGBTF2	Computes an *LU* factorization of a general band matrix, using partial pivoting with row interchanges (unblocked algorithm).
SGEBD2	CGEBD2	Reduces a general rectangular matrix to real bidiagonal form by an orthogonal/unitary transformation (unblocked algorithm).
SGEHD2	CGEHD2	Reduces a general matrix to upper Hessenberg form by an orthogonal/unitary similarity transformation (unblocked algorithm).
SGELQ2	CGELQ2	Computes an *LQ* factorization of a general rectangular matrix (unblocked algorithm).

Routine		Description
real	complex	
SGEQL2	CGEQL2	Computes a QL factorization of a general rectangular matrix (unblocked algorithm).
SGEQR2	CGEQR2	Computes a QR factorization of a general rectangular matrix (unblocked algorithm).
SGERQ2	CGERQ2	Computes an RQ factorization of a general rectangular matrix (unblocked algorithm).
SGESC2	CGESC2	Solves a system of linear equations $A * X = scale * RHS$ using the LU factorization with complete pivoting computed by xGETC2.
SGETC2	CGETC2	Computes an LU factorization with complete pivoting of the general n-by-n matrix A
SGETF2	CGETF2	Computes an LU factorization of a general matrix, using partial pivoting with row interchanges (unblocked algorithm).
SGTTS2	CGTTS2	Solves one of the systems of equations $A * X = B$ or $A^H * X = B$, with a tridiagonal matrix A using the LU factorization computed by SGTTRF/CGTTRF.
SLABAD		Returns the square root of the underflow and overflow thresholds if the exponent-range is very large.
SLABRD	CLABRD	Reduces the first nb rows and columns of a general rectangular matrix A to real bidiagonal form by an orthogonal/unitary transformation, and returns auxiliary matrices which are needed to apply the transformation to the unreduced part of A.
SLACON	CLACON	Estimates the 1-norm of a square matrix, using reverse communication for evaluating matrix-vector products.
SLACPY	CLACPY	Copies all or part of one two-dimensional array to another.
SLADIV	CLADIV	Performs complex division in real arithmetic, avoiding unnecessary overflow.
SLAE2		Computes the eigenvalues of a 2-by-2 symmetric matrix.
SLAEBZ		Computes the number of eigenvalues of a real symmetric tridiagonal matrix which are less than or equal to a given value, and performs other tasks required by the routine SSTEBZ.
SLAED0	CLAED0	Used by xSTEDC. Computes all eigenvalues and corresponding eigenvectors of an unreduced symmetric tridiagonal matrix using the divide and conquer method.
SLAED1		Used by SSTEDC. Computes the updated eigensystem of a diagonal matrix after modification by a rank-one symmetric matrix. Used when the original matrix is tridiagonal.
SLAED2		Used by SSTEDC. Merges eigenvalues and deflates secular equation. Used when the original matrix is tridiagonal.
SLAED3		Used by SSTEDC. Finds the roots of the secular equation and updates the eigenvectors. Used when the original matrix is tridiagonal.
SLAED4		Used by SSTEDC. Finds a single root of the secular equation.

Routine		Description
real	complex	
SLAED5		Used by SSTEDC. Solves the 2-by-2 secular equation.
SLAED6		Used by SSTEDC. Computes one Newton step in solution of secular equation.
SLAED7	CLAED7	Used by SSTEDC. Computes the updated eigensystem of a diagonal matrix after modification by a rank-one symmetric matrix. Used when the original matrix is dense.
SLAED8	CLAED8	Used by xSTEDC. Merges eigenvalues and deflates secular equation. Used when the original matrix is dense.
SLAED9		Used by SSTEDC. Finds the roots of the secular equation and updates the eigenvectors. Used when the original matrix is dense.
SLAEDA		Used by SSTEDC. Computes the Z vector determining the rank-one modification of the diagonal matrix. Used when the original matrix is dense.
SLAEIN	CLAEIN	Computes a specified right or left eigenvector of an upper Hessenberg matrix by inverse iteration.
SLAEV2	CLAEV2	Computes the eigenvalues and eigenvectors of a 2-by-2 symmetric/Hermitian matrix.
SLAEXC		Swaps adjacent diagonal blocks of a real upper quasi-triangular matrix in Schur canonical form, by an orthogonal similarity transformation.
SLAG2		Computes the eigenvalues of a 2-by-2 generalized eigenvalue problem $A - w * B$, with scaling as necessary to avoid over-/underflow.
SLAGS2		Computes 2-by-2 orthogonal matrices U, V, and Q, and applies them to matrices A and B such that the rows of the transformed A and B are parallel.
SLAGTF		Computes an LU factorization of a matrix $(T - \lambda I)$, where T is a general tridiagonal matrix, and λ a scalar, using partial pivoting with row interchanges.
SLAGTM	CLAGTM	Performs a matrix-matrix product of the form $C = \alpha AB + \beta C$, where A is a tridiagonal matrix, B and C are rectangular matrices, and α and β are scalars, which may be 0, 1, or -1.
SLAGTS		Solves the system of equations $(T - \lambda I)x = y$ or $(T - \lambda I)^T x = y$, where T is a general tridiagonal matrix and λ a scalar, using the LU factorization computed by SLAGTF.
SLAGV2		Computes the Generalized Schur factorization of a real 2-by-2 matrix pencil (A, B) where B is upper triangular.
SLAHQR	CLAHQR	Computes the eigenvalues and Schur factorization of an upper Hessenberg matrix, using the double-shift/single-shift QR algorithm.
SLAHRD	CLAHRD	Reduces the first nb columns of a general rectangular matrix A so that elements below the kth subdiagonal are zero, by an orthogonal/unitary transformation, and returns auxiliary matrices which are needed to apply the transformation to the unreduced part of A.
SLAIC1	CLAIC1	Applies one step of incremental condition estimation.

Routine		Description
real	complex	
SLALN2		Solves a 1-by-1 or 2-by-2 system of equations of the form $(\gamma A - \lambda D)x = \sigma b$ or $(\gamma A^T - \lambda D)x = \sigma b$, where D is a diagonal matrix, λ, b and x may be complex, and σ is a scale factor set to avoid overflow.
SLALS0	CLALS0	Used by xGELSD. Applies back the multiplying factors of either the left or the right singular vector matrix of a diagonal matrix appended by a row to the right hand side matrix B in solving the least squares problem using the divide and conquer SVD approach.
SLALSA	CLALSA	Used by xGELSD. An intermediate step in solving the least squares problem by computing the SVD of the coefficient matrix in compact form.
SLALSD	CLALSD	Used by xGELSD. Uses the singular value decomposition of A to solve the least squares problem of finding X to minimize the Euclidean norm of each column of $A * X - B$.
SLAMCH		Determines machine parameters for floating-point arithmetic.
SLAMRG		Creates a permutation list which will merge the entries of two independently sorted sets into a single set which is sorted in ascending order.
SLANGB	CLANGB	Returns the value of the 1-norm, Frobenius norm, infinity-norm, or the largest absolute value of any element, of a general band matrix.
SLANGE	CLANGE	Returns the value of the 1-norm, Frobenius norm, infinity-norm, or the largest absolute value of any element, of a general rectangular matrix.
SLANGT	CLANGT	Returns the value of the 1-norm, Frobenius norm, infinity-norm, or the largest absolute value of any element, of a general tridiagonal matrix.
SLANHS	CLANHS	Returns the value of the 1-norm, Frobenius norm, infinity-norm, or the largest absolute value of any element, of an upper Hessenberg matrix.
SLANSB	CLANSB CLANHB	Returns the value of the 1-norm, Frobenius norm, infinity-norm, or the largest absolute value of any element, of a real symmetric/complex symmetric/complex Hermitian band matrix.
SLANSP	CLANSP CLANHP	Returns the value of the 1-norm, Frobenius norm, infinity-norm, or the largest absolute value of any element, of a real symmetric/complex symmetric/complex Hermitian matrix in packed storage.
SLANST	CLANHT	Returns the value of the 1-norm, Frobenius norm, infinity-norm, or the largest absolute value of any element, of a symmetric/Hermitian tridiagonal matrix.
SLANSY	CLANSY CLANHE	Returns the value of the 1-norm, Frobenius norm, infinity-norm, or the largest absolute value of any element, of a real symmetric/complex symmetric/complex Hermitian matrix.
SLANTB	CLANTB	Returns the value of the 1-norm, Frobenius norm, infinity-norm, or the largest absolute value of any element, of a triangular band matrix.

Routine		Description
real	complex	
SLANTP	CLANTP	Returns the value of the 1-norm, Frobenius norm, infinity-norm, or the largest absolute value of any element, of a triangular matrix in packed storage.
SLANTR	CLANTR	Returns the value of the 1-norm, Frobenius norm, infinity-norm, or the largest absolute value of any element, of a triangular matrix.
SLANV2		Computes the Schur factorization of a real 2-by-2 nonsymmetric matrix in Schur canonical form.
SLAPLL	CLAPLL	Measures the linear dependence of two vectors X and Y.
SLAPMT	CLAPMT	Performs a forward or backward permutation of the columns of a matrix.
SLAPY2		Returns $\sqrt{x^2+y^2}$, avoiding unnecessary overflow or harmful underflow.
SLAPY3		Returns $\sqrt{x^2+y^2+z^2}$, avoiding unnecessary overflow or harmful underflow.
SLAQGB	CLAQGB	Scales a general band matrix, using row and column scaling factors computed by SGBEQU/CGBEQU.
SLAQGE	CLAQGE	Scales a general rectangular matrix, using row and column scaling factors computed by SGEEQU/CGEEQU.
SLAQP2	CLAQP2	Computes a QR factorization with column pivoting of the block $A(OFFSET+1:M, 1:N)$. The block $A(1:OFFSET, 1:N)$ is accordingly pivoted, but not factorized.
SLAQPS	CLAQPS	Computes a step of QR factorization with column pivoting of a real M-by-N matrix A by using Level 3 Blas.
SLAQSB	CLAQSB	Scales a symmetric/Hermitian band matrix, using scaling factors computed by SPBEQU/CPBEQU.
SLAQSP	CLAQSP	Scales a symmetric/Hermitian matrix in packed storage, using scaling factors computed by SPPEQU/CPPEQU.
SLAQSY	CLAQSY	Scales a symmetric/Hermitian matrix, using scaling factors computed by SPOEQU/CPOEQU.
SLAQTR		Solves a real quasi-triangular system of equations, or a complex quasi-triangular system of special form, in real arithmetic.
SLAR1V	CLAR1V	Computes the (scaled) r^{th} column of the inverse of the sumbmatrix in rows B1 through BN of the tridiagonal matrix $LDL^T - \sigma I$.
SLAR2V	CLAR2V	Applies a vector of plane rotations with real cosines and real/complex sines from both sides to a sequence of 2-by-2 symmetric/Hermitian matrices.
SLARF	CLARF	Applies an elementary reflector to a general rectangular matrix.
SLARFB	CLARFB	Applies a block reflector or its transpose/conjugate-transpose to a general rectangular matrix.
SLARFG	CLARFG	Generates an elementary reflector (Householder matrix).
SLARFT	CLARFT	Forms the triangular factor T of a block reflector $H = I - VTV^H$.

Routine		Description
real	complex	
SLARFX	CLARFX	Applies an elementary reflector to a general rectangular matrix, with loop unrolling when the reflector has order ≤ 10.
SLARGV	CLARGV	Generates a vector of plane rotations with real cosines and real/complex sines.
SLARNV	CLARNV	Returns a vector of random numbers from a uniform or normal distribution.
SLARRB		Given the relatively robust representation(RRR) LDL^T, SLARRB does "limited" bisection to locate the eigenvalues of LDL^T, W(IFIRST) through W(ILAST), to more accuracy.
SLARRE		Given the tridiagonal matrix T, SLARRE sets "small" off-diagonal elements to zero, and for each unreduced block T_i, it finds the numbers σ_i, the base $T_i - \sigma_i I = L_i D_i L_i^T$ representations and the eigenvalues of each $L_i D_i L_i^T$.
SLARRF		Finds a new relatively robust representation $LDL^T - \Sigma I = L(+)D(+)L(+)^T$ such that at least one of the eigenvalues of $L(+)D(+)L(+)^T$ is relatively isolated.
SLARRV	CLARRV	Computes the eigenvectors of the tridiagonal matrix $T = LDL^T$ given L, D and the eigenvalues of LDL^T.
SLARTG	CLARTG	Generates a plane rotation with real cosine and real/complex sine.
SLARTV	CLARTV	Applies a vector of plane rotations with real cosines and real/complex sines to the elements of a pair of vectors.
SLARUV		Returns a vector of n random real numbers from a uniform (0,1) distribution ($n \leq 128$).
SLARZ	CLARZ	Applies an elementary reflector (as returned by xTZRZF) to a general matrix.
SLARZB	CLARZB	Applies a block reflector or its transpose/conjugate-transpose to a general matrix.
SLARZT	CLARZT	Forms the triangular factor T of a block reflector $H = I - VTV^H$.
SLAS2		Computes the singular values of a 2-by-2 triangular matrix.
SLASCL	CLASCL	Multiplies a general rectangular matrix by a real scalar defined as c_{to}/c_{from}.
SLASD0		Used by SBDSDC. Computes via a divide and conquer method the singular values of a real upper bidiagonal n-by-m matrix with diagonal D and offdiagonal E, where $M = N + SQRE$.
SLASD1		Used by SBDSDC. Computes the SVD of an upper bidiagonal N-by-M matrix, where $N = NL + NR + 1$ and $M = N + SQRE$.
SLASD2		Used by SBDSDC. Merges the two sets of singular values together into a single sorted set, and then it tries to deflate the size of the problem.
SLASD3		Used by SBDSDC. Finds all the square roots of the roots of the secular equation, as defined by the values in D and Z, and then updates the singular vectors by matrix multiplication.

Routine		Description
real	complex	
SLASD4		Used by SBDSDC. Computes the square root of the I-th updated eigenvalue of a positive symmetric rank-one modification to a positive diagonal matrix.
SLASD5		Used by SBDSDC. Computes the square root of the I-th eigenvalue of a positive symmetric rank-one modification of a 2-by-2 diagonal matrix.
SLASD6		Used by SBDSDC. Computes the SVD of an updated upper bidiagonal matrix obtained by merging two smaller ones by appending a row.
SLASD7		Used by SBDSDC. Merges the two sets of singular values together into a single sorted set, and then it tries to deflate the size of the problem.
SLASD8		Used by SBDSDC. Finds the square roots of the roots of the secular equation, and stores, for each element in D, the distance to its two nearest poles (elements in $DSIGMA$).
SLASD9		Used by SBDSDC. Finds the square roots of the roots of the secular equation, and stores, for each element in D, the distance to its two nearest poles (elements in $DSIGMA$).
SLASDA		Used by SBDSDC. Computes the singular value decomposition (SVD) of a real upper bidiagonal N-by-M matrix with diagonal D and offdiagonal E, where $M = N + SQRE$.
SLASDQ		Used by SBDSDC. Computes the singular value decomposition (SVD) of a real (upper or lower) bidiagonal matrix with diagonal D and offdiagonal E, accumulating the transformations if desired.
SLASDT		Used by SBDSDC. Creates a tree of subproblems for bidiagonal divide and conquer.
SLASET	CLASET	Initializes the off-diagonal elements of a matrix to α and the diagonal elements to β.
SLASQ1		Used by SBDSQR. Computes the singular values of a real n-by-n bidiagonal matrix with diagonal D and offdiagonal E.
SLASQ2		Used by SBDSQR and SSTEGR. Computes all the eigenvalues of the symmetric positive definite tridiagonal matrix associated with the qd array Z to high relative accuracy.
SLASQ3		Used by SBDSQR. Checks for deflation, computes a shift (TAU) and calls dqds.
SLASQ4		Used by SBDSQR. Computes an approximation TAU to the smallest eigenvalue using values of d from the previous transform.
SLASQ5		Used by SBDSQR and SSTEGR. Computes one dqds transform in ping-pong form.
SLASQ6		Used by SBDSQR and SSTEGR. computes one dqds transform in ping-pong form.
SLASR	CLASR	Applies a sequence of plane rotations to a general rectangular matrix.

Routine		Description
real	complex	
SLASRT		Sorts numbers in increasing or decreasing order using Quick Sort, reverting to Insertion sort on arrays of size ≤ 20.
SLASSQ	CLASSQ	Updates a sum of squares represented in scaled form.
SLASV2		Computes the singular value decomposition of a 2-by-2 triangular matrix.
SLASWP	CLASWP	Performs a sequence of row interchanges on a general rectangular matrix.
SLASY2		Solves the Sylvester matrix equation $AX \pm XB = \sigma C$ where A and B are of order 1 or 2, and may be transposed, and σ is a scale factor.
SLASYF	CLASYF CLAHEF	Computes a partial factorization of a real symmetric/complex symmetric/complex Hermitian indefinite matrix, using the diagonal pivoting method.
SLATBS	CLATBS	Solves a triangular banded system of equations $Ax = \sigma b$, $A^T x = \sigma b$, or $A^H x = \sigma b$, where σ is a scale factor set to prevent overflow.
SLATDF	CLATDF	Uses the LU factorization of the n-by-n matrix computed by SGETC2 and computes a contribution to the reciprocal Dif-estimate.
SLATPS	CLATPS	Solves a triangular system of equations $Ax = \sigma b$, $A^T x = \sigma b$, or $A^H x = \sigma b$, where A is held in packed storage, and σ is a scale factor set to prevent overflow.
SLATRD	CLATRD	Reduces the first nb rows and columns of a symmetric/Hermitian matrix A to real tridiagonal form by an orthogonal/unitary similarity transformation, and returns auxiliary matrices which are needed to apply the transformation to the unreduced part of A.
SLATRS	CLATRS	Solves a triangular system of equations $Ax = \sigma b$, $A^T x = \sigma b$, or $A^H x = \sigma b$, where σ is a scale factor set to prevent overflow.
SLATRZ	CLATRZ	Factors an upper trapezoidal matrix by means of orthogonal/unitary transformations.
SLAUU2	CLAUU2	Computes the product UU^H or $L^H L$, where U and L are upper or lower triangular matrices (unblocked algorithm).
SLAUUM	CLAUUM	Computes the product UU^H or $L^H L$, where U and L are upper or lower triangular matrices.
SORG2L	CUNG2L	Generates all or part of the orthogonal/unitary matrix Q from a QL factorization determined by SGEQLF/CGEQLF (unblocked algorithm).
SORG2R	CUNG2R	Generates all or part of the orthogonal/unitary matrix Q from a QR factorization determined by SGEQRF/CGEQRF (unblocked algorithm).
SORGL2	CUNGL2	Generates all or part of the orthogonal/unitary matrix Q from an LQ factorization determined by SGELQF/CGELQF (unblocked algorithm).

Routine		Description
real	complex	
SORGR2	CUNGR2	Generates all or part of the orthogonal/unitary matrix Q from an RQ factorization determined by SGERQF/CGERQF (unblocked algorithm).
SORM2L	CUNM2L	Multiplies a general matrix by the orthogonal/unitary matrix from a QL factorization determined by SGEQLF/CGEQLF (unblocked algorithm).
SORM2R	CUNM2R	Multiplies a general matrix by the orthogonal/unitary matrix from a QR factorization determined by SGEQRF/CGEQRF (unblocked algorithm).
SORML2	CUNML2	Multiplies a general matrix by the orthogonal/unitary matrix from an LQ factorization determined by SGELQF/CGELQF (unblocked algorithm).
SORMR2	CUNMR2	Multiplies a general matrix by the orthogonal/unitary matrix from an RQ factorization determined by SGERQF/CGERQF (unblocked algorithm).
SORMR3	CUNMR3	Multiplies a general matrix by the orthogonal/unitary matrix from an RZ factorization determined by STZRZF/CTZRZF (unblocked algorithm).
SPBTF2	CPBTF2	Computes the Cholesky factorization of a symmetric/Hermitian positive definite band matrix (unblocked algorithm).
SPOTF2	CPOTF2	Computes the Cholesky factorization of a symmetric/Hermitian positive definite matrix (unblocked algorithm).
SPTTS2	CPTTS2	Solves a tridiagonal system of the form $A*X = B$ using the $L*D*L^H$ factorization of A computed by SPTTRF/CPTTRF.
SRSCL	CSRSCL	Multiplies a vector by the reciprocal of a real scalar.
SSYGS2	CHEGS2	Reduces a symmetric/Hermitian definite generalized eigenproblem $Ax = \lambda Bx$, $ABx = \lambda x$, or $BAx = \lambda x$, to standard form, where B has been factorized by SPOTRF/CPOTRF (unblocked algorithm).
SSYTD2	CHETD2	Reduces a symmetric/Hermitian matrix to real symmetric tridiagonal form by an orthogonal/unitary similarity transformation (unblocked algorithm).
SSYTF2	CSYTF2 CHETF2	Computes the factorization of a real symmetric/complex symmetric/complex Hermitian indefinite matrix, using the diagonal pivoting method (unblocked algorithm).
STGEX2	CTGEX2	Swaps adjacent diagonal blocks $(A11, B11)$ and $(A22, B22)$ of size 1-by-1 or 2-by-2 in an upper (quasi) triangular matrix pair (A, B) by an orthogonal/unitary equivalence transformation.
STGSY2	CTGSY2	Solves the generalized Sylvester equation (unblocked algorithm).
STRTI2	CTRTI2	Computes the inverse of a triangular matrix (unblocked algorithm).
XERBLA		Error handling routine called by LAPACK routines if an input parameter has an invalid value.

This page intentionally left blank

Appendix C

Quick Reference Guide to the BLAS

Level 1 BLAS

```
                  dim scalar vector   vector   scalars                 5-element prefixes
                                                                       array

SUBROUTINE _ROTG (                                     A, B, C, S )          S, D
SUBROUTINE _ROTMG(                              D1, D2, A, B,        PARAM )  S, D
SUBROUTINE _ROT  ( N,          X, INCX, Y, INCY,               C, S )        S, D
SUBROUTINE _ROTM ( N,          X, INCX, Y, INCY,                    PARAM )  S, D
SUBROUTINE _SWAP ( N,          X, INCX, Y, INCY )                            S, D, C, Z
SUBROUTINE _SCAL ( N,  ALPHA, X, INCX )                                      S, D, C, Z, CS, ZD
SUBROUTINE _COPY ( N,          X, INCX, Y, INCY )                            S, D, C, Z
SUBROUTINE _AXPY ( N,  ALPHA, X, INCX, Y, INCY )                             S, D, C, Z
FUNCTION   _DOT  ( N,          X, INCX, Y, INCY )                            S, D, DS
FUNCTION   _DOTU ( N,          X, INCX, Y, INCY )                            C, Z
FUNCTION   _DOTC ( N,          X, INCX, Y, INCY )                            C, Z
FUNCTION   __DOT ( N,  ALPHA, X, INCX, Y, INCY )                             SDS
FUNCTION   _NRM2 ( N,          X, INCX )                                     S, D, SC, DZ
FUNCTION   _ASUM ( N,          X, INCX )                                     S, D, SC, DZ
FUNCTION   I_AMAX( N,          X, INCX )                                     S, D, C, Z
```

Name	Operation	Prefixes
_ROTG	Generate plane rotation	S, D
_ROTMG	Generate modified plane rotation	S, D
_ROT	Apply plane rotation	S, D
_ROTM	Apply modified plane rotation	S, D
_SWAP	$x \leftrightarrow y$	S, D, C, Z
_SCAL	$x \leftarrow \alpha x$	S, D, C, Z, CS, ZD
_COPY	$y \leftarrow x$	S, D, C, Z
_AXPY	$y \leftarrow \alpha x + y$	S, D, C, Z
_DOT	$dot \leftarrow x^T y$	S, D, DS
_DOTU	$dot \leftarrow x^T y$	C, Z
_DOTC	$dot \leftarrow x^H y$	C, Z
__DOT	$dot \leftarrow \alpha + x^T y$	SDS
_NRM2	$nrm2 \leftarrow \|\|x\|\|_2$	S, D, SC, DZ
_ASUM	$asum \leftarrow \|\|re(x)\|\|_1 + \|\|im(x)\|\|_1$	S, D, SC, DZ
I_AMAX	$amax \leftarrow 1^{st} k \ni \|re(x_k)\| + \|im(x_k)\| = max(\|re(x_i)\| + \|im(x_i)\|)$	S, D, C, Z

Level 2 BLAS

```
        options            dim   b-width scalar matrix  vector   scalar vector   prefixes

_GEMV (        TRANS,      M, N,         ALPHA, A, LDA, X, INCX, BETA,  Y, INCY ) S, D, C, Z
_GBMV (        TRANS,      M, N, KL, KU, ALPHA, A, LDA, X, INCX, BETA,  Y, INCY ) S, D, C, Z
_HEMV ( UPLO,                 N,         ALPHA, A, LDA, X, INCX, BETA,  Y, INCY ) C, Z
_HBMV ( UPLO,                 N, K,      ALPHA, A, LDA, X, INCX, BETA,  Y, INCY ) C, Z
_HPMV ( UPLO,                 N,         ALPHA, AP,     X, INCX, BETA,  Y, INCY ) C, Z
_SYMV ( UPLO,                 N,         ALPHA, A, LDA, X, INCX, BETA,  Y, INCY ) S, D
_SBMV ( UPLO,                 N, K,      ALPHA, A, LDA, X, INCX, BETA,  Y, INCY ) S, D
_SPMV ( UPLO,                 N,         ALPHA, AP,     X, INCX, BETA,  Y, INCY ) S, D
_TRMV ( UPLO, TRANS, DIAG,    N,                A, LDA, X, INCX )                 S, D, C, Z
_TBMV ( UPLO, TRANS, DIAG,    N, K,             A, LDA, X, INCX )                 S, D, C, Z
_TPMV ( UPLO, TRANS, DIAG,    N,                AP,     X, INCX )                 S, D, C, Z
_TRSV ( UPLO, TRANS, DIAG,    N,                A, LDA, X, INCX )                 S, D, C, Z
_TBSV ( UPLO, TRANS, DIAG,    N, K,             A, LDA, X, INCX )                 S, D, C, Z
_TPSV ( UPLO, TRANS, DIAG,    N,                AP,     X, INCX )                 S, D, C, Z

        options            dim   scalar vector   vector   matrix  prefixes

_GER  (                    M, N, ALPHA, X, INCX, Y, INCY, A, LDA ) S, D
_GERU (                    M, N, ALPHA, X, INCX, Y, INCY, A, LDA ) C, Z
_GERC (                    M, N, ALPHA, X, INCX, Y, INCY, A, LDA ) C, Z
_HER  ( UPLO,                 N, ALPHA, X, INCX,          A, LDA ) C, Z
_HPR  ( UPLO,                 N, ALPHA, X, INCX,          AP )     C, Z
_HER2 ( UPLO,                 N, ALPHA, X, INCX, Y, INCY, A, LDA ) C, Z
_HPR2 ( UPLO,                 N, ALPHA, X, INCX, Y, INCY, AP )     C, Z
_SYR  ( UPLO,                 N, ALPHA, X, INCX,          A, LDA ) S, D
_SPR  ( UPLO,                 N, ALPHA, X, INCX,          AP )     S, D
_SYR2 ( UPLO,                 N, ALPHA, X, INCX, Y, INCY, A, LDA ) S, D
_SPR2 ( UPLO,                 N, ALPHA, X, INCX, Y, INCY, AP )     S, D
```

Level 3 BLAS

```
        options                        dim      scalar matrix  matrix  scalar matrix  prefixes

_GEMM (            TRANSA, TRANSB,     M, N, K, ALPHA, A, LDA, B, LDB, BETA,  C, LDC ) S, D, C, Z
_SYMM ( SIDE, UPLO,                    M, N,    ALPHA, A, LDA, B, LDB, BETA,  C, LDC ) S, D, C, Z
_HEMM ( SIDE, UPLO,                    M, N,    ALPHA, A, LDA, B, LDB, BETA,  C, LDC ) C, Z
_SYRK (       UPLO, TRANS,                N, K, ALPHA, A, LDA,         BETA,  C, LDC ) S, D, C, Z
_HERK (       UPLO, TRANS,                N, K, ALPHA, A, LDA,         BETA,  C, LDC ) C, Z
_SYR2K(       UPLO, TRANS,                N, K, ALPHA, A, LDA, B, LDB, BETA,  C, LDC ) S, D, C, Z
_HER2K(       UPLO, TRANS,                N, K, ALPHA, A, LDA, B, LDB, BETA,  C, LDC ) C, Z
_TRMM ( SIDE, UPLO, TRANSA,       DIAG, M, N,    ALPHA, A, LDA, B, LDB )               S, D, C, Z
_TRSM ( SIDE, UPLO, TRANSA,       DIAG, M, N,    ALPHA, A, LDA, B, LDB )               S, D, C, Z
```

Name	Operation	Prefixes
_GEMV	$y \leftarrow \alpha Ax + \beta y, y \leftarrow \alpha A^T x + \beta y, y \leftarrow \alpha A^H x + \beta y, A - m \times n$	S, D, C, Z
_GBMV	$y \leftarrow \alpha Ax + \beta y, y \leftarrow \alpha A^T x + \beta y, y \leftarrow \alpha A^H x + \beta y, A - m \times n$	S, D, C, Z
_HEMV	$y \leftarrow \alpha Ax + \beta y$	C, Z
_HBMV	$y \leftarrow \alpha Ax + \beta y$	C, Z
_HPMV	$y \leftarrow \alpha Ax + \beta y$	C, Z
_SYMV	$y \leftarrow \alpha Ax + \beta y$	S, D
_SBMV	$y \leftarrow \alpha Ax + \beta y$	S, D
_SPMV	$y \leftarrow \alpha Ax + \beta y$	S, D
_TRMV	$x \leftarrow Ax, x \leftarrow A^T x, x \leftarrow A^H x$	S, D, C, Z
_TBMV	$x \leftarrow Ax, x \leftarrow A^T x, x \leftarrow A^H x$	S, D, C, Z
_TPMV	$x \leftarrow Ax, x \leftarrow A^T x, x \leftarrow A^H x$	S, D, C, Z
_TRSV	$x \leftarrow A^{-1}x, x \leftarrow A^{-T}x, x \leftarrow A^{-H}x$	S, D, C, Z
_TBSV	$x \leftarrow A^{-1}x, x \leftarrow A^{-T}x, x \leftarrow A^{-H}x$	S, D, C, Z
_TPSV	$x \leftarrow A^{-1}x, x \leftarrow A^{-T}x, x \leftarrow A^{-H}x$	S, D, C, Z

_GER	$A \leftarrow \alpha xy^T + A, A - m \times n$	S, D
_GERU	$A \leftarrow \alpha xy^T + A, A - m \times n$	C, Z
_GERC	$A \leftarrow \alpha xy^H + A, A - m \times n$	C, Z
_HER	$A \leftarrow \alpha xx^H + A$	C, Z
_HPR	$A \leftarrow \alpha xx^H + A$	C, Z
_HER2	$A \leftarrow \alpha xy^H + y(\alpha x)^H + A$	C, Z
_HPR2	$A \leftarrow \alpha xy^H + y(\alpha x)^H + A$	C, Z
_SYR	$A \leftarrow \alpha xx^T + A$	S, D
_SPR	$A \leftarrow \alpha xx^T + A$	S, D
_SYR2	$A \leftarrow \alpha xy^T + \alpha yx^T + A$	S, D
_SPR2	$A \leftarrow \alpha xy^T + \alpha yx^T + A$	S, D

Name	Operation	Prefixes
_GEMM	$C \leftarrow \alpha op(A)op(B) + \beta C, op(X) = X, X^T, X^H, C - m \times n$	S, D, C, Z
_SYMM	$C \leftarrow \alpha AB + \beta C, C \leftarrow \alpha BA + \beta C, C - m \times n, A = A^T$	S, D, C, Z
_HEMM	$C \leftarrow \alpha AB + \beta C, C \leftarrow \alpha BA + \beta C, C - m \times n, A = A^H$	C, Z
_SYRK	$C \leftarrow \alpha AA^T + \beta C, C \leftarrow \alpha A^T A + \beta C, C - n \times n$	S, D, C, Z
_HERK	$C \leftarrow \alpha AA^H + \beta C, C \leftarrow \alpha A^H A + \beta C, C - n \times n$	C, Z
_SYR2K	$C \leftarrow \alpha AB^T + \alpha BA^T + \beta C, C \leftarrow \alpha A^T B + \alpha B^T A + \beta C, C - n \times n$	S, D, C, Z
_HER2K	$C \leftarrow \alpha AB^H + \bar{\alpha} BA^H + \beta C, C \leftarrow \alpha A^H B + \bar{\alpha} B^H A + \beta C, C - n \times n$	C, Z
_TRMM	$B \leftarrow \alpha op(A)B, B \leftarrow \alpha Bop(A), op(A) = A, A^T, A^H, B - m \times n$	S, D, C, Z
_TRSM	$B \leftarrow \alpha op(A^{-1})B, B \leftarrow \alpha Bop(A^{-1}), op(A) = A, A^T, A^H, B - m \times n$	S, D, C, Z

Notes

Meaning of prefixes

S - REAL	C - COMPLEX	
D - DOUBLE PRECISION	Z - COMPLEX*16	(this may not be supported by all machines)

For the Level 2 BLAS a set of extended-precision routines with the prefixes ES, ED, EC, EZ may also be available.

Level 1 BLAS

In addition to the listed routines there are two further extended-precision dot product routines DQDOTI and DQDOTA.

Level 2 and Level 3 BLAS

Matrix types

GE - GEneral	GB - General Band	
SY - SYmmetric	SB - Symmetric Band	SP - Symmetric Packed
HE - HErmitian	HB - Hermitian Band	HP - Hermitian Packed
TR - TRiangular	TB - Triangular Band	TP - Triangular Packed

Options

Arguments describing options are declared as CHARACTER*1 and may be passed as character strings.

TRANS	= 'No transpose', 'Transpose', 'Conjugate transpose' (X, X^T, X^C)
UPLO	= 'Upper triangular', 'Lower triangular'
DIAG	= 'Non-unit triangular', 'Unit triangular'
SIDE	= 'Left', 'Right' (A or op(A) on the left, or A or op(A) on the right)

For real matrices, TRANS = 'T' and TRANS = 'C' have the same meaning.
For Hermitian matrices, TRANS = 'T' is not allowed.
For complex symmetric matrices, TRANS = 'H' is not allowed.

Appendix D

Converting from LINPACK or EISPACK

This appendix is designed to assist people to convert programs that currently call LINPACK or EISPACK routines, to call LAPACK routines instead.

Notes

1. The appendix consists mainly of indexes giving the nearest LAPACK equivalents of LINPACK and EISPACK routines. These indexes should not be followed blindly or rigidly, especially when two or more LINPACK or EISPACK routines are being used together: in many such cases one of the LAPACK driver routines may be a suitable replacement.

2. When two or more LAPACK routines are given in a single entry, these routines must be combined to achieve the equivalent function.

3. For LINPACK, an index is given for equivalents of the real LINPACK routines; these equivalences apply also to the corresponding complex routines. A separate table is included for equivalences of complex Hermitian routines. For EISPACK, an index is given for all real and complex routines, since there is no direct 1-to-1 correspondence between real and complex routines in EISPACK.

4. A few of the less commonly used routines in LINPACK and EISPACK have no equivalents in Release 1.0 of LAPACK; equivalents for some of these (but not all) are planned for a future release.

5. For some EISPACK routines, there are LAPACK routines providing similar functionality, but using a significantly different method, or LAPACK routines which provide only part of the functionality; such routines are marked by a †. For example, the EISPACK routine ELMHES uses non-orthogonal transformations, whereas the nearest equivalent LAPACK routine, SGEHRD, uses orthogonal transformations.

6. In some cases the LAPACK equivalents require matrices to be stored in a different storage scheme. For example:

   - EISPACK routines BANDR, BANDV, BQR and the driver routine RSB require the lower triangle of a symmetric band matrix to be stored in a different storage scheme to that used in LAPACK, which is illustrated in subsection 5.3.3. The corresponding storage scheme used by the EISPACK routines is:

symmetric band matrix A	EISPACK band storage
$\begin{pmatrix} a_{11} & a_{21} & a_{31} & & \\ a_{21} & a_{22} & a_{32} & a_{42} & \\ a_{31} & a_{32} & a_{33} & a_{43} & a_{53} \\ & a_{42} & a_{43} & a_{44} & a_{54} \\ & & a_{53} & a_{54} & a_{55} \end{pmatrix}$	$\begin{matrix} * & * & a_{11} \\ * & a_{21} & a_{22} \\ a_{31} & a_{32} & a_{33} \\ a_{42} & a_{43} & a_{44} \\ a_{53} & a_{54} & a_{55} \end{matrix}$

   - EISPACK routines TRED1, TRED2, TRED3, HTRID3, HTRIDI, TQL1, TQL2, IMTQL1, IMTQL2, RATQR, TQLRAT and the driver routine RST store the off-diagonal elements of a symmetric tridiagonal matrix in elements $2 : n$ of the array E, whereas LAPACK routines use elements $1 : n - 1$.

7. The EISPACK and LINPACK routines for the singular value decomposition return the matrix of right singular vectors, V, whereas the corresponding LAPACK routines return the transposed matrix V^T.

8. In general, the argument lists of the LAPACK routines are different from those of the corresponding EISPACK and LINPACK routines, and the workspace requirements are often different.

LAPACK equivalents of LINPACK routines for real matrices		
LINPACK	LAPACK	Function of LINPACK routine
SCHDC		Cholesky factorization with diagonal pivoting option
SCHDD		Rank-1 downdate of a Cholesky factorization or the triangular factor of a QR factorization
SCHEX		Modifies a Cholesky factorization under permutations of the original matrix
SCHUD		Rank-1 update of a Cholesky factorization or the triangular factor of a QR factorization
SGBCO	SLANGB SGBTRF SGBCON	LU factorization and condition estimation of a general band matrix
SGBDI		Determinant of a general band matrix, after factorization by SGBCO or SGBFA
SGBFA	SGBTRF	LU factorization of a general band matrix
SGBSL	SGBTRS	Solves a general band system of linear equations, after factorization by SGBCO or SGBFA
SGECO	SLANGE SGETRF SGECON	LU factorization and condition estimation of a general matrix
SGEDI	SGETRI	Determinant and inverse of a general matrix, after factorization by SGECO or SGEFA
SGEFA	SGETRF	LU factorization of a general matrix
SGESL	SGETRS	Solves a general system of linear equations, after factorization by SGECO or SGEFA
SGTSL	SGTSV	Solves a general tridiagonal system of linear equations
SPBCO	SLANSB SPBTRF SPBCON	Cholesky factorization and condition estimation of a symmetric positive definite band matrix
SPBDI		Determinant of a symmetric positive definite band matrix, after factorization by SPBCO or SPBFA
SPBFA	SPBTRF	Cholesky factorization of a symmetric positive definite band matrix
SPBSL	SPBTRS	Solves a symmetric positive definite band system of linear equations, after factorization by SPBCO or SPBFA
SPOCO	SLANSY SPOTRF SPOCON	Cholesky factorization and condition estimation of a symmetric positive definite matrix
SPODI	SPOTRI	Determinant and inverse of a symmetric positive definite matrix, after factorization by SPOCO or SPOFA
SPOFA	SPOTRF	Cholesky factorization of a symmetric positive definite matrix
SPOSL	SPOTRS	Solves a symmetric positive definite system of linear equations, after factorization by SPOCO or SPOFA
SPPCO	SLANSY SPPTRF SPPCON	Cholesky factorization and condition estimation of a symmetric positive definite matrix (packed storage)

LAPACK equivalents of LINPACK routines for real matrices (continued)		
LINPACK	LAPACK	Function of LINPACK routine
SPPDI	SPPTRI	Determinant and inverse of a symmetric positive definite matrix, after factorization by SPPCO or SPPFA (packed storage)
SPPFA	SPPTRF	Cholesky factorization of a symmetric positive definite matrix (packed storage)
SPPSL	SPPTRS	Solves a symmetric positive definite system of linear equations, after factorization by SPPCO or SPPFA (packed storage)
SPTSL	SPTSV	Solves a symmetric positive definite tridiagonal system of linear equations
SQRDC	SGEQPF or SGEQRF	QR factorization with optional column pivoting
SQRSL	SORMQR STRSV[a]	Solves linear least squares problems after factorization by SQRDC
SSICO	SLANSY SSYTRF SSYCON	Symmetric indefinite factorization and condition estimation of a symmetric indefinite matrix
SSIDI	SSYTRI	Determinant, inertia and inverse of a symmetric indefinite matrix, after factorization by SSICO or SSIFA
SSIFA	SSYTRF	Symmetric indefinite factorization of a symmetric indefinite matrix
SSISL	SSYTRS	Solves a symmetric indefinite system of linear equations, after factorization by SSICO or SSIFA
SSPCO	SLANSP SSPTRF SSPCON	Symmetric indefinite factorization and condition estimation of a symmetric indefinite matrix (packed storage)
SSPDI	SSPTRI	Determinant, inertia and inverse of a symmetric indefinite matrix, after factorization by SSPCO or SSPFA (packed storage)
SSPFA	SSPTRF	Symmetric indefinite factorization of a symmetric indefinite matrix (packed storage)
SSPSL	SSPTRS	Solves a symmetric indefinite system of linear equations, after factorization by SSPCO or SSPFA (packed storage)
SSVDC	SGESVD	All or part of the singular value decomposition of a general matrix
STRCO	STRCON	Condition estimation of a triangular matrix
STRDI	STRTRI	Determinant and inverse of a triangular matrix
STRSL	STRTRS	Solves a triangular system of linear equations

[a]STRSV is a Level 2 BLAS routine, not an LAPACK routine.

LAPACK equivalents of LINPACK routines for complex Hermitian matrices		
LINPACK	LAPACK	Function of LINPACK routine
CHICO	CHECON	Factors a complex Hermitian matrix by elimination with symmetric pivoting and estimates the condition number of the matrix
CHIDI	CHETRI	Computes the determinant, inertia and inverse of a complex Hermitian matrix using the factors from CHIFA
CHIFA	CHETRF	Factors a complex Hermitian matrix by elimination with symmetric pivoting
CHISL	CHETRS	Solves the complex Hermitian system Ax=b using the factors computed by CHIFA
CHPCO	CHPCON	Factors a complex Hermitian matrix stored in packed form by elimination with symmetric pivoting and estimates the condition number of the matrix
CHPDI	CHPTRI	Computes the determinant, intertia and inverse of a complex Hermitian matrix using the factors from CHPFA, where the matrix is stored in packed form
CHPFA	CHPTRF	Factors a complex Hermitian matrix stored in packed form by elimination with symmetric pivoting
CHPSL	CHPTRS	Solves the complex Hermitian system Ax=b using the factors computed by CHPFA

LAPACK equivalents of EISPACK routines		
EISPACK	LAPACK	Function of EISPACK routine
BAKVEC		Backtransform eigenvectors after transformation by FIGI
BALANC	SGEBAL	Balance a real matrix
BALBAK	SGEBAK	Backtransform eigenvectors of a real matrix after balancing by BALANC
BANDR	SSBTRD	Reduce a real symmetric band matrix to tridiagonal form
BANDV	SSBEVX SGBSV	Selected eigenvectors of a real band matrix by inverse iteration
BISECT	SSTEBZ	Eigenvalues in a specified interval of a real symmetric tridiagonal matrix
BQR	SSBEVX†	Some eigenvalues of a real symmetric band matrix
CBABK2	CGEBAK	Backtransform eigenvectors of a complex matrix after balancing by CBAL
CBAL	CGEBAL	Balance a complex matrix
CG	CGEEV	All eigenvalues and optionally eigenvectors of a complex general matrix (driver routine)
CH	CHEEV	All eigenvalues and optionally eigenvectors of a complex Hermitian matrix (driver routine)
CINVIT	CHSEIN	Selected eigenvectors of a complex upper Hessenberg matrix by inverse iteration
COMBAK	CUNMHR†	Backtransform eigenvectors of a complex matrix after reduction by COMHES
COMHES	CGEHRD†	Reduce a complex matrix to upper Hessenberg form by a non-unitary transformation
COMLR	CHSEQR†	All eigenvalues of a complex upper Hessenberg matrix, by the LR algorithm
COMLR2	CUNGHR CHSEQR CTREVC†	All eigenvalues/vectors of a complex matrix by the LR algorithm, after reduction by COMHES
COMQR	CHSEQR	All eigenvalues of a complex upper Hessenberg matrix by the QR algorithm
COMQR2	CUNGHR CHSEQR CTREVC	All eigenvalues/vectors of a complex matrix by the QR algorithm, after reduction by CORTH
CORTB	CUNMHR	Backtransform eigenvectors of a complex matrix, after reduction by CORTH
CORTH	CGEHRD	Reduce a complex matrix to upper Hessenberg form by a unitary transformation
ELMBAK	SORMHR†	Backtransform eigenvectors of a real matrix after reduction by ELMHES
ELMHES	SGEHRD†	Reduce a real matrix to upper Hessenberg form by a non-orthogonal transformation
ELTRAN	SORGHR†	Generate transformation matrix used by ELMHES

LAPACK equivalents of EISPACK routines (continued)		
EISPACK	LAPACK	Function of EISPACK routine
FIGI		Transform a nonsymmetric tridiagonal matrix of special form to a symmetric matrix
FIGI2		As FIGI, with generation of the transformation matrix
HQR	SHSEQR	All eigenvalues of a complex upper Hessenberg matrix by the QR algorithm
HQR2	SHSEQR STREVC	All eigenvalues/vectors of a real upper Hessenberg matrix by the QR algorithm
HTRIB3	CUPMTR	Backtransform eigenvectors of a complex Hermitian matrix after reduction by HTRID3
HTRIBK	CUNMTR	Backtransform eigenvectors of a complex Hermitian matrix after reduction by HTRIDI
HTRID3	CHPTRD	Reduce a complex Hermitian matrix to tridiagonal form (packed storage)
HTRIDI	CHETRD	Reduce a complex Hermitian matrix to tridiagonal form
IMTQL1	SSTEQR or SSTERF†	All eigenvalues of a symmetric tridiagonal matrix, by the implicit QL algorithm
IMTQL2	SSTEQR or SSTEDC†	All eigenvalues/vectors of a symmetric tridiagonal matrix, by the implicit QL algorithm
IMTQLV	SSTEQR	As IMTQL1, preserving the input matrix
INVIT	SHSEIN	Selected eigenvectors of a real upper Hessenberg matrix, by inverse iteration
MINFIT	SGELSS	Minimum norm solution of a linear least squares problem, using the singular value decomposition
ORTBAK	SORMHR	Backtransform eigenvectors of a real matrix after reduction to upper Hessenberg form by ORTHES
ORTHES	SGEHRD	Reduce a real matrix to upper Hessenberg form by an orthogonal transformation
ORTRAN	SORGHR	Generate orthogonal transformation matrix used by ORTHES
QZHES	SGGHRD	Reduce a real generalized eigenproblem $Ax = \lambda Bx$ to a form in which A is upper Hessenberg and B is upper triangular
QZIT QZVAL	SHGEQZ	Generalized Schur factorization of a real generalized eigenproblem, after reduction by QZHES
QZVEC	STGEVC	All eigenvectors of a real generalized eigenproblem from generalized Schur factorization
RATQR	SSTEBZ†	Extreme eigenvalues of a symmetric tridiagonal matrix using the rational QR algorithm with Newton corrections
REBAK	STRSM[a]	Backtransform eigenvectors of a symmetric definite generalized eigenproblem $Ax = \lambda Bx$ or $ABx = \lambda x$ after reduction by REDUC or REDUC2
REBAKB	STRMM[b]	Backtransform eigenvectors of a symmetric definite generalized eigenproblem $BAx = \lambda x$ after reduction by REDUC2

[a]STRSM is a Level 3 BLAS routine, not an LAPACK routine.
[b]STRMM is a Level 3 BLAS routine, not an LAPACK routine.

LAPACK equivalents of EISPACK routines (continued)		
EISPACK	LAPACK	Function of EISPACK routine
REDUC	SSYGST	Reduce the symmetric definite generalized eigenproblem $Ax = \lambda Bx$ to a standard symmetric eigenproblem
REDUC2	SSYGST	Reduce the symmetric definite generalized eigenproblem $ABx = \lambda x$ or $BAx = \lambda x$ to a standard symmetric eigenproblem
RG	SGEEV	All eigenvalues and optionally eigenvectors of a real general matrix (driver routine)
RGG	SGEGV	All eigenvalues and optionally eigenvectors or a real generalized eigenproblem (driver routine)
RS	SSYEV or SSYEVD†	All eigenvalues and optionally eigenvectors of a real symmetric matrix (driver routine)
RSB	SSBEV or SSBEVD†	All eigenvalues and optionally eigenvectors of a real symmetric band matrix (driver routine)
RSG	SSYGV	All eigenvalues and optionally eigenvectors of a real symmetric definite generalized eigenproblem $Ax = \lambda Bx$ (driver routine)
RSGAB	SSYGV	All eigenvalues and optionally eigenvectors of a real symmetric definite generalized eigenproblem $ABx = \lambda x$ (driver routine)
RSGBA	SSYGV	All eigenvalues and optionally eigenvectors of a real symmetric definite generalized eigenproblem $BAx = \lambda x$ (driver routine)
RSM	SSYEVX	Selected eigenvalues and optionally eigenvectors of a real symmetric matrix (driver routine)
RSP	SSPEV or SSPEVD†	All eigenvalues and optionally eigenvectors of a real symmetric matrix (packed storage) (driver routine)
RST	SSTEV or SSTEVD†	All eigenvalues and optionally eigenvectors of a real symmetric tridiagonal matrix (driver routine)
RT		All eigenvalues and optionally eigenvectors of a real tridiagonal matrix of special form (driver routine)
SVD	SGESVD	Singular value decomposition of a real matrix
TINVIT	SSTEIN	Selected eigenvectors of a symmetric tridiagonal matrix by inverse iteration
TQL1	SSTEQR† or SSTERF†	All eigenvalues of a symmetric tridiagonal matrix by the explicit QL algorithm
TQL2	SSTEQR† or SSTEDC†	All eigenvalues/vectors of a symmetric tridiagonal matrix by the explicit QL algorithm
TQLRAT	SSTERF	All eigenvalues of a symmetric tridiagonal matrix by a rational variant of the QL algorithm

LAPACK equivalents of EISPACK routines (continued)		
EISPACK	LAPACK	Function of EISPACK routine
TRBAK1	SORMTR	Backtransform eigenvectors of a real symmetric matrix after reduction by TRED1
TRBAK3	SOPMTR	Backtransform eigenvectors of a real symmetric matrix after reduction by TRED3 (packed storage)
TRED1	SSYTRD	Reduce a real symmetric matrix to tridiagonal form
TRED2	SSYTRD SORGTR	As TRED1, but also generating the orthogonal transformation matrix
TRED3	SSPTRD	Reduce a real symmetric matrix to tridiagonal form (packed storage)
TRIDIB	SSTEBZ	Eigenvalues between specified indices of a symmetric tridiagonal matrix
TSTURM	SSTEBZ SSTEIN	Eigenvalues in a specified interval of a symmetric tridiagonal matrix, and corresponding eigenvectors by inverse iteration

Appendix E

LAPACK Working Notes

Most of these working notes are available from *netlib*, where they can be obtained in postscript or pdf form.

```
http://www.netlib.org/lapack/lawns/
http://www.netlib.org/lapack/lawnspdf/
```


1. J. W. DEMMEL, J. J. DONGARRA, J. DU CROZ, A. GREENBAUM, S. HAMMARLING, AND D. SORENSEN, *Prospectus for the Development of a Linear Algebra Library for High-Performance Computers*, ANL, MCS-TM-97, September 1987.

2. J. J. DONGARRA, S. HAMMARLING, AND D. SORENSEN, *Block Reduction of Matrices to Condensed Forms for Eigenvalue Computations*, ANL, MCS-TM-99, September 1987.

3. J. W. DEMMEL AND W. KAHAN, *Computing Small Singular Values of Bidiagonal Matrices with Guaranteed High Relative Accuracy*, ANL, MCS-TM-110, February 1988.

4. J. W. DEMMEL, J. DU CROZ, S. HAMMARLING, AND D. SORENSEN, *Guidelines for the Design of Symmetric Eigenroutines, SVD, and Iterative Refinement and Condition Estimation for Linear Systems*, ANL, MCS-TM-111, March 1988.

5. C. BISCHOF, J. W. DEMMEL, J. J. DONGARRA, J. DU CROZ, A. GREENBAUM, S. HAMMARLING, AND D. SORENSEN, *Provisional Contents*, ANL, MCS-TM-38, September 1988.

6. O. BREWER, J. J. DONGARRA, AND D. SORENSEN, *Tools to Aid in the Analysis of Memory Access Patterns for FORTRAN Programs*, ANL, MCS-TM-120, June 1988.

7. J. BARLOW AND J. W. DEMMEL, *Computing Accurate Eigensystems of Scaled Diagonally Dominant Matrices*, ANL, MCS-TM-126, December 1988.

8. Z. BAI AND J. W. DEMMEL, *On a Block Implementation of Hessenberg Multishift QR Iteration*, ANL, MCS-TM-127, January 1989.

9. J. W. DEMMEL AND A. MCKENNEY, *A Test Matrix Generation Suite*, ANL, MCS-P69-0389, March 1989.

10. E. ANDERSON AND J. J. DONGARRA, *Installing and Testing the Initial Release of LAPACK – Unix and Non-Unix Versions*, ANL, MCS-TM-130, May 1989.

11. P. DEIFT, J. W. DEMMEL, L.-C. LI, AND C. TOMEI, *The Bidiagonal Singular Value Decomposition and Hamiltonian Mechanics*, ANL, MCS-TM-133, August 1989.

12. P. MAYES AND G. RADICATI, *Banded Cholesky Factorization Using Level 3 BLAS*, ANL, MCS-TM-134, August 1989.

13. Z. BAI, J. W. DEMMEL, AND A. MCKENNEY, *On the Conditioning of the Nonsymmetric Eigenproblem: Theory and Software*, UT, CS-89-86, October 1989.

14. J. W. DEMMEL, *On Floating-Point Errors in Cholesky*, UT, CS-89-87, October 1989.

15. J. W. DEMMEL AND K. VESELIĆ, *Jacobi's Method is More Accurate than QR*, UT, CS-89-88, October 1989.

16. E. ANDERSON AND J. J. DONGARRA, *Results from the Initial Release of LAPACK*, UT, CS-89-89, November 1989.

17. A. GREENBAUM AND J. J. DONGARRA, *Experiments with QR/QL Methods for the Symmetric Tridiagonal Eigenproblem*, UT, CS-89-92, November 1989.

18. E. ANDERSON AND J. J. DONGARRA, *Implementation Guide for LAPACK*, UT, CS-90-101, April 1990.

19. E. ANDERSON AND J. J. DONGARRA, *Evaluating Block Algorithm Variants in LAPACK*, UT, CS-90-103, April 1990.

20. E. ANDERSON, Z. BAI, C. BISCHOF, J. W. DEMMEL, J. J. DONGARRA, J. DU CROZ, A. GREENBAUM, S. HAMMARLING, A. MCKENNEY, AND D. SORENSEN, *LAPACK: A Portable Linear Algebra Library for High-Performance Computers*, UT, CS-90-105, May 1990.

21. J. DU CROZ, P. MAYES, AND G. RADICATI, *Factorizations of Band Matrices Using Level 3 BLAS*, UT, CS-90-109, July 1990.

22. J. W. DEMMEL AND N. J. HIGHAM, *Stability of Block Algorithms with Fast Level 3 BLAS*, UT, CS-90-110, July 1990.

23. J. W. DEMMEL AND N. J. HIGHAM, *Improved Error Bounds for Underdetermined System Solvers*, UT, CS-90-113, August 1990.

24. J. J. DONGARRA AND S. OSTROUCHOV, *LAPACK Block Factorization Algorithms on the Intel iPSC/860*, UT, CS-90-115, October, 1990.

25. J. J. DONGARRA, S. HAMMARLING, AND J. H. WILKINSON, *Numerical Considerations in Computing Invariant Subspaces*, UT, CS-90-117, October, 1990.

26. E. ANDERSON, C. BISCHOF, J. W. DEMMEL, J. J. DONGARRA, J. DU CROZ, S. HAMMARLING, AND W. KAHAN, *Prospectus for an Extension to LAPACK: A Portable Linear Algebra Library for High-Performance Computers*, UT, CS-90-118, November 1990.

27. J. DU CROZ AND N. J. HIGHAM, *Stability of Methods for Matrix Inversion*, UT, CS-90-119, October, 1990.

28. J. J. DONGARRA, P. MAYES, AND G. RADICATI, *The IBM RISC System/6000 and Linear Algebra Operations*, UT, CS-90-122, December 1990.

29. R. VAN DE GEIJN, *On Global Combine Operations*, UT, CS-91-129, April 1991.

30. J. J. DONGARRA AND R. VAN DE GEIJN, *Reduction to Condensed Form for the Eigenvalue Problem on Distributed Memory Architectures*, UT, CS-91-130, April 1991.

31. E. ANDERSON, Z. BAI, AND J. J. DONGARRA, *Generalized QR Factorization and its Applications*, UT, CS-91-131, April 1991.

32. C. BISCHOF AND P. TANG, *Generalized Incremental Condition Estimation*, UT, CS-91-132, May 1991.

33. C. BISCHOF AND P. TANG, *Robust Incremental Condition Estimation*, UT, CS-91-133, May 1991.

34. J. J. DONGARRA, *Workshop on the BLACS*, UT, CS-91-134, May 1991.

35. E. ANDERSON, J. J. DONGARRA, AND S. OSTROUCHOV, *Implementation guide for LAPACK*, UT, CS-91-138, August 1991. (*replaced by Working Note 41*)

36. E. ANDERSON, *Robust Triangular Solves for Use in Condition Estimation*, UT, CS-91-142, August 1991.

37. J. J. DONGARRA AND R. VAN DE GEIJN, *Two Dimensional Basic Linear Algebra Communication Subprograms*, UT, CS-91-138, October 1991.

38. Z. BAI AND J. W. DEMMEL, *On a Direct Algorithm for Computing Invariant Subspaces with Specified Eigenvalues*, UT, CS-91-139, November 1991.

39. J. W. DEMMEL, J. J. DONGARRA, AND W. KAHAN, *On Designing Portable High Performance Numerical Libraries*, UT, CS-91-141, July 1991.

40. J. W. DEMMEL, N. J. HIGHAM, AND R. SCHREIBER, *Block LU Factorization*, UT, CS-92-149, February 1992.

41. L. S. BLACKFORD AND J. J. DONGARRA, *Installation Guide for LAPACK*, UT, CS-92-151, February 1992.

42. N. J. HIGHAM, *Perturbation Theory and Backward Error for* $AX - XB = C$, UT, CS-92-153, April, 1992.

43. J. J. DONGARRA, R. VAN DE GEIJN, AND D. W. WALKER, *A Look at Scalable Dense Linear Algebra Libraries*, UT, CS-92-155, April, 1992.

44. E. ANDERSON AND J. J. DONGARRA, *Performance of LAPACK: A Portable Library of Numerical Linear Algebra Routines*, UT, CS-92-156, May 1992.

45. J. W. DEMMEL, *The Inherent Inaccuracy of Implicit Tridiagonal QR*, UT, CS-92-162, May 1992.

46. Z. BAI AND J. W. DEMMEL, *Computing the Generalized Singular Value Decomposition*, UT, CS-92-163, May 1992.

47. J. W. DEMMEL, *Open Problems in Numerical Linear Algebra*, UT, CS-92-164, May 1992.

48. J. W. DEMMEL AND W. GRAGG, *On Computing Accurate Singular Values and Eigenvalues of Matrices with Acyclic Graphs*, UT, CS-92-166, May 1992.

49. J. W. DEMMEL, *A Specification for Floating Point Parallel Prefix*, UT, CS-92-167, May 1992.

50. V. EIJKHOUT, *Distributed Sparse Data Structures for Linear Algebra Operations*, UT, CS-92-169, May 1992.

51. V. EIJKHOUT, *Qualitative Properties of the Conjugate Gradient and Lanczos Methods in a Matrix Framework*, UT, CS-92-170, May 1992.

52. M. T. HEATH AND P. RAGHAVAN, *A Cartesian Parallel Nested Dissection Algorithm*, UT, CS-92-178, June 1992.

53. J. W. DEMMEL, *Trading Off Parallelism and Numerical Stability*, UT, CS-92-179, June 1992.

54. Z. BAI AND J. W. DEMMEL, *On Swapping Diagonal Blocks in Real Schur Form*, UT, CS-92-182, October 1992.

55. J. CHOI, J. J. DONGARRA, R. POZO, AND D. W. WALKER, *ScaLAPACK: A Scalable Linear Algebra for Distributed Memory Concurrent Computers*, UT, CS-92-181, November 1992.

56. E. F. D'AZEVEDO, V. L. EIJKHOUT AND C. H. ROMINE, *Reducing Communication Costs in the Conjugate Gradient Algorithm on Distributed Memory Multiprocessors*, UT, CS-93-185, January 1993.

57. J. CHOI, J. J. DONGARRA, AND D. W. WALKER, *PUMMA: Parallel Universal Matrix Multiplication Algorithms on Distributed Memory Concurrent Computers*, UT, CS-93-187, May 1993.

58. J. J. DONGARRA AND D. W. WALKER, *The Design of Linear Algebra Libraries for High Performance Computer*, UT, CS-93-188, June 1993.

59. J. W. DEMMEL AND X. LI, *Faster Numerical Algorithms via Exception Handling*, UT, CS-93-192, March 1993.

60. J. W. DEMMEL, M. T. HEATH, AND H. A. VAN DER VORST, *Parallel Numerical Linear Algebra*, UT, CS-93-192, March 1993.

61. J. J. DONGARRA, R. POZO, AND D. W. WALKER, *An Object Oriented Design for High Performance Linear Algebra on Distributed Memory Architectures*, UT, CS-93-200, August 1993.

62. M. T. HEATH AND P. RAGHAVAN, *Distributed Solution of Sparse Linear Systems*, UT, CS-93-201, August 1993.

63. M. T. HEATH AND P. RAGHAVAN, *Line and Plane Separators*, UT, CS-93-202, August 1993.

64. P. RAGHAVAN, *Distributed Sparse Gaussian Elimination and Orthogonal Factorization*, UT, CS-93-203, August 1993.

65. J. CHOI, J. J. DONGARRA, AND D. W. WALKER, *Parallel Matrix Transpose Algorithms on Distributed Memory Concurrent Computers*, UT, CS-93-215, November, 1993.

66. V. L. EIJKHOUT, *A Characterization of Polynomial Iterative Methods*, UT, CS-93-216, November, 1993.

67. F. DESPREZ, J. DONGARRA, AND B. TOURANCHEAU, *Performance Complexity of LU Factorization with Efficient Pipelining and Overlap on a Multiprocessor*, UT, CS-93-218, December, 1993.

68. MICHAEL W. BERRY, JACK J. DONGARRA AND YOUNGBAE KIM, *A Highly Parallel Algorithm for the Reduction of a Nonsymmetric Matrix to Block Upper-Hessenberg Form*, UT, CS-94-221, January, 1994.

69. J. RUTTER, *A Serial Implementation of Cuppen's Divide and Conquer Algorithm for the Symmetric Eigenvalue Problem*, UT, CS-94-225, March, 1994.

70. J. W. DEMMEL, INDERJIT DHILLON, AND HUAN REN, *On the Correctness of Parallel Bisection in Floating Point*, UT, CS-94-228, April, 1994.

71. J. DONGARRA AND M. KOLATIS, *IBM RS/6000-550 & -590 Performance for Selected Routines in ESSL*, UT, CS-94-231, April, 1994.

72. R. LEHOUCQ, *The Computation of Elementary Unitary Matrices*, UT, CS-94-233, May, 1994.

73. R. CLINT WHALEY, *Basic Linear Algebra Communication Subprograms: Analysis and Implementation Across Multiple Parallel Architectures*, UT, CS-94-234, May, 1994.

74. J. DONGARRA, A. LUMSDAINE, X. NIU, R. POZO, AND K. REMINGTON, *A Sparse Matrix Library in C++ for High Performance Architectures*, UT, CS-94-236, July, 1994.

75. B. KÅGSTRÖM AND P. POROMAA, *Computing Eigenspaces with Specified Eigenvalues of a Regular Matrix Pair (A,B) and Condition Estimation: Theory, Algorithms and Software*, UT, CS-94-237, July, 1994.

76. R. BARRETT, M. BERRY, J. DONGARRA, V. EIJKHOUT, AND C. ROMINE, *Algorithic Bombardment for the Iterative Solution of Linear Systems: A Poly-Iterative Approach*, UT, CS-94-239, August, 1994.

77. V. EIJKHOUT AND R. POZO, *Basic Concepts for Distributed Sparse Linear Algebra Operations*, UT, CS-94-240, August, 1994.

78. V. EIJKHOUT, *Computational variants of the CGS and BiCGstab methods*, UT, CS-94-241, August, 1994.

79. G. HENRY AND R. VAN DE GEIJN, *Parallelizing the QR Algorithm for the Unsymmetric Algebraic Eigenvalue Problem: Myths and Reality*, UT, CS-94-244, August, 1994.

80. J. CHOI, J. J. DONGARRA, S. OSTROUCHOV, A. P. PETITET, D. W. WALKER, AND R. C. WHALEY, *The Design and Implementation of the ScaLAPACK LU, QR, and Cholesky Factorization Routines*, UT, CS-94-246, September, 1994.

81. J. J. DONGARRA AND L. S. BLACKFORD, *Quick Installation Guide for LAPACK on Unix Systems*, UT, CS-94-249, September, 1994.

82. J. J. DONGARRA AND M. KOLATIS, *Call Conversion Interface (CCI) for LAPACK/ESSL*, UT, CS-94-250, August, 1994.

83. R. C. LI, *Relative Perturbation Bounds for the Unitary Polar Factor*, UT, CS-94-251, September, 1994.

84. R. C. LI, *Relative Perturbation Theory: (I) Eigenvalue Variations*, UT, CS-94-252, September, 1994.

85. R. C. LI, *Relative Perturbation Theory: (II) Eigenspace Variations*, UT, CS-94-253, September, 1994.

86. J. DEMMEL AND K. STANLEY, *The Performance of Finding Eigenvalues and Eigenvectors of Dense Symmetric Matrices on Distributed Memory Computers*, UT, CS-94-254, September, 1994.

87. B. KÅGSTRÖM AND P. POROMAA, *Computing Eigenspaces with Specified Eigenvalues of a Regular Matrix Pair (A,B) and Condition Estimation: Theory, Algorithms and Software*, UT, CS-94-255, September, 1994.

88. MING GU, JAMES DEMMEL, AND INDERJIT DHILLON, *Efficient Computation of the Singular Value Decomposition with Applications to Least Squares Problems*, UT, CS-94-257, October, 1994.

89. REN-CANG LI, *Solving Secular Equations Stably and Efficiently*, UT, CS-94-260, November, 1994.

90. J. S. PLANK, Y. KIM, AND J. J. DONGARRA, *Algorithm-Based Diskless Checkpointing for Fault Tolerant Matrix Operations*, UT, CS-94-268, December, 1994.

91. Z. BAI, J. DEMMEL, J. DONGARRA, A. PETITET, H. ROBINSON, AND K. STANLEY, *The Spectral Decomposition of Nonsymmetric Matrices on Distributed Memory Computers*, UT, CS-95-273, January, 1995.

92. J. CHOI, J. DONGARRA, AND D. WALKER, *The Design of a Parallel Dense Linear Algebra Software Library: Reduction to Hessenberg, Tridiagonal, and Bidiagonal Form*, UT, CS-95-275, February, 1995.

93. J. CHOI, J. DEMMEL, I. DHILLON, J. DONGARRA, S. OSTROUCHOV, A. PETITET, K. STANLEY, D. WALKER, AND R. C. WHALEY, *Installation Guide for ScaLAPACK*, UT, CS-95-280, March, 1995.

94. J. DONGARRA AND R. C. WHALEY, *A User's Guide to the BLACS v1.0*, UT, CS-95-281, March 1995.

95. J. CHOI, J. DEMMEL, I. DHILLON, J. DONGARRA, S. OSTROUCHOV, A. PETITET, K. STANLEY, D. WALKER, AND R. C. WHALEY, *ScaLAPACK: A Portable Linear Algebra Library for Distributed Memory Computers - Design Issues and Performance*, UT, CS-95-283, March, 1995.

96. R. A. VAN DE GEIJN AND J. WATTS, *SUMMA: Scalable Universal Matrix Multiplication Algorithm*, UT, CS-95-286, April, 1995.

97. S. CHAKRABARTI, J. DEMMEL, AND D. YELICK, *Modeling the Benefits of Mixed Data and Task Parallelism*, UT, CS-95-289, May, 1995.

98. J. DONGARRA, R. POZO, AND D. WALKER, *LAPACK++ V. 1.0: High Performance Linear Algebra Users' Guide*, UT, CS-95-290, May, 1995.

99. J. DONGARRA, V. EIJKHOUT, AND A. KALHAN, *Reverse Communication Interface for Linear Algebra Templates for Iterative Methods*, UT, CS-95-291, May, 1995.

100. J. CHOI, J. DONGARRA, S. OSTROUCHOV, A. PETITET, D. WALKER, AND R. C. WHALEY, *A Proposal for a Set of Parallel Basic Linear Algebra Subprograms*, UT, CS-95-292, May, 1995.

101. J. J. DONGARRA, J. DU CROZ, S. HAMMARLING, J. WASNIEWSKI, AND A. ZEMLA, *A Proposal for a Fortran 90 Interface for LAPACK*, UT, CS-95-295, July, 1995.

102. J. DONGARRA, A. LUMSDAINE, R. POZO, AND K. REMINGTON, *IML++ v. 1.2: Iterative Methods Library Reference Guide*, UT, CS-95-303, August, 1995.

103. J. W. DEMMEL, S. C. EISENSTAT, J. R. GILBERT, X. S. LI, AND J. W. H. LIU, *A Supernodal Approach to Sparse Partial Pivoting*, UT, CS-95-304, September, 1995.

104. N. J. HIGHAM, *Iterative Refinement and LAPACK*, UT, CS-95-308, October, 1995.

105. N. J. HIGHAM, *Stability of the Diagonal Pivoting Method with Partial Pivoting*, UT, CS-95-309, October, 1995.

106. Z. BAI, D. DAY, J. DEMMEL, J. DONGARRA, M. GU, A. RUHE, AND H. VAN DER VORST, *Templates for Linear Algebra Problems*, UT, CS-95-311, October, 1995.

107. B. KÅGSTRÖM, P. LING, AND C. VAN LOAN, *GEMM-Based Level 3 BLAS: High-Performance Model Implementations and Performance Evaluation Benchmark*, UT, CS-95-315, November, 1995.

108. B. KÅGSTRÖM, P. LING, AND C. VAN LOAN, *GEMM-Based Level 3 BLAS: Installation, Tuning and Use of the Model Implementations and the Performance Evaluation Benchmark*, UT, CS-95-316, November, 1995.

109. J. DONGARRA, S. HAMMARLING, AND S. OSTROUCHOV, *BLAS Technical Workshop*, UT, CS-95-317, November, 1995.

110. J. J. DONGARRA, S. HAMMARLING, AND D. W. WALKER, *Key Concepts For Parallel Out-Of-Core LU Factorization*, UT, CS-96-324, April, 1996.

111. J. BILMES, K. ASANOVIC, J. DEMMEL, D. LAM, AND C.-W. CHIN,*Optimizing Matrix Multiply using PHiPAC: a Portable, High-Performance, ANSI C Coding Methodology*, UT, CS-96-326, May, 1996.

112. L. S. BLACKFORD, A. CLEARY, J. DEMMEL, I. DHILLON, J. DONGARRA, S. HAMMARLING, A. PETITET, H. REN, K. STANLEY, R. C. WHALEY, *Practical Experience in the Dangers of Heterogeneous Computing*, UT, CS-96-330, July, 1996.

113. G. QUINTANA-ORTÍ, E. S. QUINTANA-ORTÍ, A. PETITET, *Block-Partitioned Algorithms for Solving the Linear Least Squares Problem*, UT, CS-96-333, July, 1996.

114. G. QUINTANA-ORTÍ, X. SUN, C. H. BISCHOF, *A BLAS-3 Version of the QR Factorization with Column Pivoting*, UT, CS-96-334, August, 1996.

115. H. REN, *On the Error Analysis and Implementation of Some Eigenvalue Decomposition and Singular Value Decomposition Algorithms*, UT, CS-96-336, September, 1996.

116. M. SIDANI AND B. HARROD, *Parallel Matrix Distributions: Have we been doing it all right?*, UT, CS-96-340, November 1996.

117. L. S. BLACKFORD, J. DONGARRA, J. DU CROZ, S. HAMMARLING, J. WAŚNIEWSKI, *A Fortran 90 Interface for LAPACK: LAPACK90, version 1.0*, UT, CS-96-341, November 1996.

118. J. J. DONGARRA AND E. F. D'AZEVEDO, *The Design and Implementation of the Parallel Out-of-core ScaLAPACK LU, QR, and Cholesky Factorization Routines*, UT, CS-97-347, January 1997.

119. J. DEMMEL, M. GU, S. EISENSTAT, I. SLAPNICAR, K. VESELIC, AND Z. DRMAC, *Computing the Singular Value Decomposition with High Relative Accuracy*, UT, CS-97-348, February 1997.

120. F. DESPREZ, J. DONGARRA, A. PETITET, C. RANDRIAMARO, AND Y. ROBERT, *Scheduling Block-Cyclic Array Redistribution*, UT, CS-97-349, February 1997.

121. G. HENRY, D. WATKINS, AND J. DONGARRA, *A Parallel Implementation of the Nonsymmetric QR Algorithm for Distributed Memory Architectures*, UT, CS-97-352, March 1997.

122. M. AHUES AND F. TISSEUR, *A New Deflation Criterion for the QR Algorithm*, UT, CS-97-353, March 1997.

123. Z. Bai, D. Day, J. Demmel and J. Dongarra, *A Test Matrix Collection for Non-Hermitian Eigenvalue Problems*, UT, CS-97-355, March 1997.

124. J. Demmel, J. Gilbert, and X. Li, *An Asynchronous Parallel Supernodal Algorithm for Sparse Gaussian Elimination*, UT, CS-97-357, April 1997.

125. A. Cleary and J. Dongarra, *Implementation in ScaLAPACK of Divide-and-Conquer Algorithms for Banded and Tridiagonal Linear Systems*, UT, CS-97-358, April 1997.

126. E. Anderson and M. Fahey, *Performance Improvements to LAPACK for the Cray Scientific Library*, UT, CS-97-359, April 1997.

127. X. Li, *Sparse Gaussian Elimination on High Performance Computers*, UT, CS-97-368, June 1997.

128. A. Petitet, *Algorithmic Redistribution Methods for Block Cyclic Decompositions*, UT, CS-97-371, July 1997.

129. J. Choi, *A New Parallel Matrix Multiplication Algorithm on Distributed-Memory Concurrent Computers*, UT, CS-97-369, September 1997.

130. J. Demmel, *Accurate SVDs of Structured Matrices*, UT, CS-97-375, October 1997.

131. R. Whaley and J. Dongarra, *Automatically Tuned Linear Algebra Software*, UT, CS-97-366, December 1997.

132. F. Tisseur and J. Dongarra, *Parallelizing the Divide and Conquer Algorithm for the Symmetric Tridiagonal Eigenvalue Problem on Distributed Memory Architectures*, UT, CS-98-382, March 1998.

133. A. Petitet and J. Dongarra, *Algorithmic Redistribution Methods for Block Cyclic Distributions*, UT, CS-98-383, March 1998.

134. J. Waśniewski and J. Dongarra, *High Performance Linear Algebra Package – LAPACK90*, UT, CS-98-384, April 1998.

135. E. D'Azevedo and J. Dongarra, *Packed Storage Extensions for ScaLAPACK*, UT, CS-98-385, April 1998.

136. L. S. Blackford and R. C. Whaley, *ScaLAPACK Evaluation and Performance at the DoD MSRCs*, UT, CS-98-388, April 1998.

137. L. S. Blackford, J. J. Dongarra, C. A Papadopoulos, and R. C Whaley, *Installation Guide and Design of the HPF 1.1 interface to ScaLAPACK, SLHPF*, UT, CS-98-396, August 1998.

138. J. J. Dongarra, W. Owczarz, J. Wa, and P. Yalamov, *Testing Software for LAPACK90*, UT, CS-98-401, September 1998.

139. A. Petitet, H. Casanova, J. Dongarra, Y. Robert, and R. C. Whaley, *A Numerical Linear Algebra Problem Solving Environment Designer's Perspective*, UT, CS-98-405, October 1998.

140. H. CASANOVA AND J. DONGARRA, *NetSolve version 1.2: Design and Implementation*, UT, CS-98-406, November 1998.

141. V. EIJKHOUT, *Overview of Iterative Linear System Solver Packages*, UT, CS-98-411, December 1998.

142. P. ARBENZ, A. CLEARY, J. DONGARRA, AND M. HEGLAND, *A Comparison of Parallel Solvers for Diagonally Dominant and General Narrow-Banded Linear Systems*, UT, CS-99-414, February 1999.

143. P. ARBENZ, A. CLEARY, J. DONGARRA, AND M. HEGLAND, *A Comparison of Parallel Solvers for Diagonally Dominant and General Narrow-Banded Linear Systems II*, UT, CS-99-415, May 1999.

This page intentionally left blank

Part 2

Specifications of Routines

This page intentionally left blank

Notes

1. The specifications that follow give the calling sequence, purpose, and descriptions of the arguments of each LAPACK driver and computational routine (but not of auxiliary routines).

2. Specifications of pairs of real and complex routines have been merged (for example SBDSQR/CBDSQR). In a few cases, specifications of three routines have been merged, one for real symmetric, one for complex symmetric, and one for complex Hermitian matrices (for example SSYTRF/CSYTRF/CHETRF). A few routines for real matrices have no complex equivalent (for example SSTEBZ).

3. Specifications are given only for *single precision* routines. To adapt them for the double precision version of the software, simply interpret REAL as DOUBLE PRECISION, COMPLEX as COMPLEX*16 (or DOUBLE COMPLEX), and the initial letters S- and C- of LAPACK routine names as D- and Z-.

4. Specifications are arranged in alphabetical order of the real routine name.

5. The text of the specifications has been derived from the leading comments in the source-text of the routines. It makes only a limited use of mathematical typesetting facilities. To eliminate redundancy, A^H has been used throughout the specifications. Thus, the reader should note that A^H is equivalent to A^T in the real case.

6. If there is a discrepancy between the specifications listed in this section and the actual source code, the source code should be regarded as the most up-to-date.

SBDSDC

```
      SUBROUTINE SBDSDC( UPLO, COMPQ, N, D, E, U, LDU, VT, LDVT, Q, IQ,
     $                   WORK, IWORK, INFO )
      CHARACTER          COMPQ, UPLO
      INTEGER            INFO, LDU, LDVT, N
      INTEGER            IQ( * ), IWORK( * )
      REAL               D( * ), E( * ), Q( * ), U( LDU, * ),
     $                   VT( LDVT, * ), WORK( * )
```

Purpose

SBDSDC computes the singular value decomposition (SVD) of a real n-by-n (upper or lower) bidiagonal matrix B: B = U*S*VT, using a divide and conquer method, where S is a diagonal matrix with non-negative diagonal elements (the singular values of B), and U and VT are orthogonal matrices of left and right singular vectors, respectively. SBDSDC can be used to compute all singular values, and optionally, singular vectors or singular vectors in compact form.

This code makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

The code currently call SLASDQ if singular values only are desired. However, it can be slightly modified to compute singular values using the divide and conquer method.

Arguments

UPLO (input) CHARACTER*1
= 'U': B is upper bidiagonal;
= 'L': B is lower bidiagonal.

COMPQ (input) CHARACTER*1
= 'N': Compute singular values only;
= 'P': Compute singular values and compute singular vectors in compact form;
= 'I': Compute singular values and singular vectors.

N (input) INTEGER
The order of the matrix B. $N \geq 0$.

D (input/output) REAL array, dimension (N)
On entry, the n diagonal elements of the bidiagonal matrix B.
On exit, if INFO=0, the singular values of B.

E (input/output) REAL array, dimension (N)
On entry, the elements of E contain the off-diagonal elements of the bidiagonal matrix whose SVD is desired.
On exit, E has been destroyed.

U (output) REAL array, dimension (LDU, N)
If COMPQ = 'I', then:
On exit, if INFO = 0, U contains the left singular vectors of the bidiagonal matrix.
For other values of COMPQ, U is not referenced.

LDU (input) INTEGER
The leading dimension of the array U. $LDU \geq 1$.
If singular vectors are desired, then $LDU \geq \max(1,N)$.

VT (output) REAL array, dimension (LDVT, N)
If COMPQ = 'I', then:
On exit, if INFO = 0, VT' contains the right singular vectors of the bidiagonal matrix.
For other values of COMPQ, VT is not referenced.

LDVT (input) INTEGER
The leading dimension of the array VT. $LDVT \geq 1$. If singular vectors are desired, then $LDVT \geq \max(1,N)$.

Q (output) REAL array, dimension (LDQ)
If COMPQ = 'P', then: On exit, if INFO = 0, Q and IQ contain the left and right singular vectors in a compact form, requiring O(N log N) space instead of $2*N^2$. In particular, Q contains all the REAL data in $LDQ \geq N*(11 + 2*SMLSIZ + 8*INT(LOG_2(N/(SMLSIZ+1))))$ words of memory, where SMLSIZ is returned by ILAENV and is equal to the maximum size of the subproblems at the bottom of the computation tree (usually about 25). For other values of COMPQ, Q is not referenced.

IQ (output) INTEGER array, dimension (LDIQ)
If COMPQ = 'P', then: On exit, if INFO = 0, Q and IQ contain the left and right singular vectors in a compact form, requiring O(N log N) space instead of $2*N^2$. In particular, IQ contains all INTEGER data in $LDIQ \geq N*(3 + 3*INT(LOG_2(N/(SMLSIZ+1))))$ words of memory, where SMLSIZ is returned by ILAENV and is equal to the maximum size of the subproblems at the bottom of the computation tree (usually about 25). For other values of COMPQ, IQ is not referenced.

WORK (workspace) REAL array, dimension (LWORK)
If COMPQ = 'N' then $LWORK \geq (4*N)$.
If COMPQ = 'P' then $LWORK \geq (6*N)$.
If COMPQ = 'I' then $LWORK \geq (3*N^2 + 2*N)$.

IWORK (workspace) INTEGER array, dimension (7*N)

INFO (output) INTEGER
= 0: successful exit
< 0: If INFO = −i, the ith argument had an illegal value.
> 0: the algorithm failed to compute a singular value. The update process of divide and conquer failed.

SBDSQR/CBDSQR

```
      SUBROUTINE SBDSQR( UPLO, N, NCVT, NRU, NCC, D, E, VT, LDVT, U,
     $                   LDU, C, LDC, WORK, INFO )
      CHARACTER          UPLO
      INTEGER            INFO, LDC, LDU, LDVT, N, NCC, NCVT, NRU
      REAL               C( LDC, * ), D( * ), E( * ), U( LDU, * ),
     $                   VT( LDVT, * ), WORK( * )

      SUBROUTINE CBDSQR( UPLO, N, NCVT, NRU, NCC, D, E, VT, LDVT, U,
     $                   LDU, C, LDC, WORK, INFO )
      CHARACTER          UPLO
      INTEGER            INFO, LDC, LDU, LDVT, N, NCC, NCVT, NRU
      REAL               D( * ), E( * ), WORK( * )
      COMPLEX            C( LDC, * ), U( LDU, * ), VT( LDVT, * )
```

Purpose

SBDSQR/CBDSQR computes the singular value decomposition (SVD) of a real n-by-n (upper or lower) bidiagonal matrix B: $B = Q*S*P^T$, where S is a diagonal matrix with non-negative diagonal elements (the singular values of B), and Q and P are orthogonal matrices.

The routine computes S, and optionally computes $U*Q$, P^T*VT, or Q^T*C, for given real/complex input matrices U, VT, and C.

Arguments

UPLO (input) CHARACTER*1
= 'U': B is upper bidiagonal;
= 'L': B is lower bidiagonal.

N (input) INTEGER
The order of the matrix B. $N \geq 0$.

NCVT (input) INTEGER
The number of columns of the matrix VT. $NCVT \geq 0$.

NRU (input) INTEGER
The number of rows of the matrix U. $NRU \geq 0$.

NCC (input) INTEGER
The number of columns of the matrix C. $NCC \geq 0$.

D (input/output) REAL array, dimension (N)
On entry, the n diagonal elements of the bidiagonal matrix B.
On exit, if INFO=0, the singular values of B in decreasing order.

E (input/output) REAL array, dimension (N)
On entry, the elements of E contain the off-diagonal elements of the bidiagonal matrix whose SVD is desired.
On normal exit (INFO = 0), E is destroyed. If the algorithm does not converge (INFO > 0), D and E will contain the diagonal and superdiagonal elements of a bidiagonal matrix orthogonally equivalent to the one given as input. E(n) is used for workspace.

VT (input/output) REAL/COMPLEX array, dimension (LDVT, NCVT)
On entry, an n-by-ncvt matrix VT.
On exit, VT is overwritten by P^T*VT.
VT is not referenced if NCVT = 0.

LDVT (input) INTEGER
The leading dimension of the array VT.
$LDVT \geq \max(1,N)$ if $NCVT > 0$; $LDVT \geq 1$ if $NCVT = 0$.

U (input/output) REAL/COMPLEX array, dimension (LDU, N)
On entry, an nru-by-n matrix U.
On exit, U is overwritten by $U*Q$.
U is not referenced if NRU = 0.

LDU (input) INTEGER
The leading dimension of the array U. $LDU \geq \max(1,NRU)$.

C (input/output) REAL/COMPLEX array, dimension (LDC, NCC)
On entry, an n-by-ncc matrix C.
On exit, C is overwritten by Q^T*C.
C is not referenced if NCC = 0.

LDC (input) INTEGER
The leading dimension of the array C.
$LDC \geq \max(1,N)$ if $NCC > 0$; $LDC \geq 1$ if $NCC = 0$.

WORK (workspace) REAL array, dimension (4*N)

INFO (output) INTEGER
= 0: successful exit
< 0: If INFO = −i, the ith argument had an illegal value.
> 0: the algorithm did not converge; D and E contain the elements of a bidiagonal matrix which is orthogonally similar to the input matrix B; if INFO = i, i elements of E have not converged to zero.

SDISNA

```
      SUBROUTINE SDISNA( JOB, M, N, D, SEP, INFO )
      CHARACTER          JOB
      INTEGER            INFO, M, N
      REAL               D( * ), SEP( * )
```

Purpose

SDISNA computes the reciprocal condition numbers for the eigenvectors of a real symmetric or complex Hermitian matrix or for the left or right singular vectors of a general m-by-n matrix. The reciprocal condition number is the 'gap' between the corresponding eigenvalue or singular value and the nearest other one.

The bound on the error, measured by angle in radians, in the ith computed vector is given by

SLAMCH('E')*(ANORM / SEP(i))

where ANORM = $\| A \|_2 = \max(\text{abs}(D(j)))$. SEP(i) is not allowed to be smaller than SLAMCH('E')*ANORM in order to limit the size of the error bound.

SDISNA may also be used to compute error bounds for eigenvectors of the generalized symmetric definite eigenproblem.

Arguments

JOB (input) CHARACTER*1
Specifies for which problem the reciprocal condition numbers should be computed:
= 'E': the eigenvectors of a symmetric/Hermitian matrix;
= 'L': the left singular vectors of a general matrix;
= 'R': the right singular vectors of a general matrix.

M (input) INTEGER
The number of rows of the matrix. $M \geq 0$.

N (input) INTEGER
If JOB = 'L' or 'R', the number of columns of the matrix, in which case $N \geq 0$. Ignored if JOB = 'E'.

D (input) REAL array, dimension (M) if JOB = 'E'
dimension (min(M,N)) if JOB = 'L' or 'R'
The eigenvalues (if JOB = 'E') or singular values (if JOB = 'L' or 'R') of the matrix, in either increasing or decreasing order. If singular values, they must be non-negative.

SEP (output) REAL array, dimension (M) if JOB = 'E'
dimension (min(M,N)) if JOB = 'L' or 'R'
The reciprocal condition numbers of the vectors.

INFO (output) INTEGER
= 0: successful exit.
< 0: if INFO = −i, the ith argument had an illegal value.

SGBBRD/CGBBRD

```
      SUBROUTINE SGBBRD( VECT, M, N, NCC, KL, KU, AB, LDAB, D, E, Q,
     $                   LDQ, PT, LDPT, C, LDC, WORK, INFO )
      CHARACTER          VECT
      INTEGER            INFO, KL, KU, LDAB, LDC, LDPT, LDQ, M, N, NCC
      REAL               AB( LDAB, * ), C( LDC, * ), D( * ), E( * ),
     $                   PT( LDPT, * ), Q( LDQ, * ), WORK( * )
```

```
      SUBROUTINE CGBBRD( VECT, M, N, NCC, KL, KU, AB, LDAB, D, E, Q,
     $                   LDQ, PT, LDPT, C, LDC, WORK, RWORK, INFO )
      CHARACTER          VECT
      INTEGER            INFO, KL, KU, LDAB, LDC, LDPT, LDQ, M, N, NCC
      REAL               D( * ), E( * ), RWORK( * )
      COMPLEX            AB( LDAB, * ), C( LDC, * ), PT( LDPT, * ),
     $                   Q( LDQ, * ), WORK( * )
```

Purpose

SGBBRD/CGBBRD reduces a real/complex general m-by-n band matrix A to real upper bidiagonal form B by an orthogonal/unitary transformation: $Q^H*A*P = B$.

The routine computes B, and optionally forms Q or P^H, or computes Q^H*C for a given matrix C.

Arguments

VECT (input) CHARACTER*1
Specifies whether or not the matrices Q and P^H are to be formed.
= 'N': do not form Q or P^H;
= 'Q': form Q only;
= 'P': form P^H only;
= 'B': form both.

M (input) INTEGER
The number of rows of the matrix A. $M \geq 0$.

N (input) INTEGER
The number of columns of the matrix A. $N \geq 0$.

NCC (input) INTEGER
The number of columns of the matrix C. $NCC \geq 0$.

KL (input) INTEGER
The number of subdiagonals of the matrix A. $KL \geq 0$.

KU (input) INTEGER
The number of superdiagonals of the matrix A. $KU \geq 0$.

AB (input/output) REAL/COMPLEX array, dimension (LDAB,N)
On entry, the m-by-n band matrix A, stored in rows 1 to kl+ku+1. The jth column of A is stored in the jth column of the array AB as follows:
$AB(ku+1+i-j,j) = A(i,j)$ for $\max(1,j-ku) \leq i \leq \min(m,j+kl)$.
On exit, A is overwritten by values generated during the reduction.

LDAB (input) INTEGER
The leading dimension of the array A. $LDAB \geq KL+KU+1$.

D (output) REAL array, dimension (min(M,N))
The diagonal elements of the bidiagonal matrix B.

E (output) REAL array, dimension (min(M,N)−1)
The superdiagonal elements of the bidiagonal matrix B.

Q (output) REAL/COMPLEX array, dimension (LDQ,M)
If VECT = 'Q' or 'B', the m-by-m orthogonal/unitary matrix Q.
If VECT = 'N' or 'P', the array Q is not referenced.

LDQ (input) INTEGER
The leading dimension of the array Q.
LDQ $\geq$ max(1,M) if VECT = 'Q' or 'B'; LDQ $\geq$ 1 otherwise.

PT (output) REAL/COMPLEX array, dimension (LDPT,N)
If VECT = 'P' or 'B', the n-by-n orthogonal/unitary matrix P^H.
If VECT = 'N' or 'Q', the array PT is not referenced.

LDPT (input) INTEGER
The leading dimension of the array PT.
LDPT $\geq$ max(1,N) if VECT = 'P' or 'B'; LDPT $\geq$ 1 otherwise.

C (input/output) REAL/COMPLEX array, dimension (LDC,NCC)
On entry, an m-by-ncc matrix C.
On exit, C is overwritten by Q^H*C.
C is not referenced if NCC = 0.

LDC (input) INTEGER
The leading dimension of the array C.
LDC $\geq$ max(1,M) if NCC > 0; LDC $\geq$ 1 if NCC = 0.

WORK *SGBBRD* (workspace) REAL array, dimension (2*max(M,N))
CGBBRD (workspace) COMPLEX array, dimension (max(M,N))

RWORK *CGBBRD only* (workspace) REAL array, dimension (max(M,N))

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SGBCON/CGBCON

```
      SUBROUTINE SGBCON( NORM, N, KL, KU, AB, LDAB, IPIV, ANORM, RCOND,
     $                   WORK, IWORK, INFO )
      CHARACTER          NORM
      INTEGER            INFO, KL, KU, LDAB, N
      REAL               ANORM, RCOND
      INTEGER            IPIV( * ), IWORK( * )
      REAL               AB( LDAB, * ), WORK( * )

      SUBROUTINE CGBCON( NORM, N, KL, KU, AB, LDAB, IPIV, ANORM, RCOND,
     $                   WORK, RWORK, INFO )
      CHARACTER          NORM
      INTEGER            INFO, KL, KU, LDAB, N
      REAL               ANORM, RCOND
      INTEGER            IPIV( * )
      REAL               RWORK( * )
      COMPLEX            AB( LDAB, * ), WORK( * )
```

Purpose

SGBCON/CGBCON estimates the reciprocal of the condition number of a real/complex general band matrix A, in either the 1-norm or the infinity-norm, using the LU factorization computed by SGBTRF/CGBTRF.

An estimate is obtained for $\|A^{-1}\|$, and the reciprocal of the condition number is computed as RCOND = $1/(\|A\| * \|A^{-1}\|)$.

Arguments

NORM (input) CHARACTER*1
Specifies whether the 1-norm condition number or the infinity-norm condition number is required:
= '1' or 'O': 1-norm;
= 'I': Infinity-norm.

N (input) INTEGER
The order of the matrix A. N $\geq$ 0.

KL (input) INTEGER
The number of subdiagonals within the band of A. KL $\geq$ 0.

KU (input) INTEGER
The number of superdiagonals within the band of A. KU $\geq$ 0.

AB (input) REAL/COMPLEX array, dimension (LDAB,N)
Details of the LU factorization of the band matrix A, as computed by SGBTRF/CGBTRF. U is stored as an upper triangular band matrix with kl+ku superdiagonals in rows 1 to kl+ku+1, and the multipliers used during the factorization are stored in rows kl+ku+2 to 2*kl+ku+1.

LDAB (input) INTEGER
The leading dimension of the array AB. LDAB $\geq$ 2*kl+ku+1.

IPIV (input) INTEGER array, dimension (N)
The pivot indices; for 1 $\leq$ i $\leq$ N, row i of the matrix was interchanged with row IPIV(i).

ANORM (input) REAL
If NORM = '1' or 'O', the 1-norm of the original matrix A.
If NORM = 'I', the infinity-norm of the original matrix A.

RCOND (output) REAL
The reciprocal of the condition number of the matrix A, computed as RCOND = $1/(\|A\| * \|A^{-1}\|)$.

WORK *SGBCON* (workspace) REAL array, dimension (3*N)
CGBCON (workspace) COMPLEX array, dimension (2*N)

IWORK *SGBCON only* (workspace) INTEGER array, dimension (N)

RWORK *CGBCON only* (workspace) REAL array, dimension (N)

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SGBEQU/CGBEQU

```
      SUBROUTINE SGBEQU( M, N, KL, KU, AB, LDAB, R, C, ROWCND, COLCND,
     $                   AMAX, INFO )
      INTEGER            INFO, KL, KU, LDAB, M, N
      REAL               AMAX, COLCND, ROWCND
      REAL               AB( LDAB, * ), C( * ), R( * )

      SUBROUTINE CGBEQU( M, N, KL, KU, AB, LDAB, R, C, ROWCND, COLCND,
     $                   AMAX, INFO )
      INTEGER            INFO, KL, KU, LDAB, M, N
      REAL               AMAX, COLCND, ROWCND
      REAL               C( * ), R( * )
      COMPLEX            AB( LDAB, * )
```

Purpose

SGBEQU/CGBEQU computes row and column scalings intended to equilibrate an m-by-n band matrix A and reduce its condition number. R returns the row scale factors and C the column scale factors, chosen to try to make the largest element in each row and column of the matrix B with elements B(i,j) = R(i)*A(i,j)*C(j) have absolute value 1.

R(i) and C(j) are restricted to be between SMLNUM = smallest safe number and BIGNUM = largest safe number. Use of these scaling factors is not guaranteed to reduce the condition number of A but works well in practice.

Arguments

M (input) INTEGER
The number of rows of the matrix A. $M \geq 0$.

N (input) INTEGER
The number of columns of the matrix A. $N \geq 0$.

KL (input) INTEGER
The number of subdiagonals within the band of A. $KL \geq 0$.

KU (input) INTEGER
The number of superdiagonals within the band of A. $KU \geq 0$.

AB (input) REAL/COMPLEX array, dimension (LDAB,N)
The band matrix A, stored in rows 1 to kl+ku+1. The jth column of A is stored in the jth column of the array AB as follows:
$AB(ku+1+i-j,j) = A(i,j)$ for $\max(1,j-ku) \leq i \leq \min(m,j+kl)$.

LDAB (input) INTEGER
The leading dimension of the array AB. $LDAB \geq KL+KU+1$.

R (output) REAL array, dimension (M)
If INFO = 0, or INFO > M, R contains the row scale factors for A.

C (output) REAL array, dimension (N)
If INFO = 0, C contains the column scale factors for A.

ROWCND (output) REAL
If INFO = 0 or INFO > M, ROWCND contains the ratio of the smallest R(i) to the largest R(i). If ROWCND ≥ 0.1 and AMAX is neither too large nor too small, it is not worth scaling by R.

COLCND (output) REAL
If INFO = 0, COLCND contains the ratio of the smallest C(i) to the largest C(i). If COLCND ≥ 0.1, it is not worth scaling by C.

AMAX (output) REAL
Absolute value of largest matrix element. If AMAX is very close to overflow or very close to underflow, the matrix should be scaled.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, and i is
$\leq$ M: the ith row of A is exactly zero
> M: the $(i-M)^{th}$ column of A is exactly zero

SGBRFS/CGBRFS

```
      SUBROUTINE SGBRFS( TRANS, N, KL, KU, NRHS, AB, LDAB, AFB, LDAFB,
     $                   IPIV, B, LDB, X, LDX, FERR, BERR, WORK, IWORK,
     $                   INFO )
      CHARACTER          TRANS
      INTEGER            INFO, KL, KU, LDAB, LDAFB, LDB, LDX, N, NRHS
      INTEGER            IPIV( * ), IWORK( * )
      REAL               AB( LDAB, * ), AFB( LDAFB, * ), B( LDB, * ),
     $                   BERR( * ), FERR( * ), WORK( * ), X( LDX, * )

      SUBROUTINE CGBRFS( TRANS, N, KL, KU, NRHS, AB, LDAB, AFB, LDAFB,
     $                   IPIV, B, LDB, X, LDX, FERR, BERR, WORK, RWORK,
     $                   INFO )
      CHARACTER          TRANS
      INTEGER            INFO, KL, KU, LDAB, LDAFB, LDB, LDX, N, NRHS
      INTEGER            IPIV( * )
      REAL               BERR( * ), FERR( * ), RWORK( * )
      COMPLEX            AB( LDAB, * ), AFB( LDAFB, * ), B( LDB, * ),
     $                   WORK( * ), X( LDX, * )
```

Purpose

SGBRFS/CGBRFS improves the computed solution to a system of linear equations when the coefficient matrix is banded, and provides error bounds and backward error estimates for the solution.

Arguments

TRANS (input) CHARACTER*1
Specifies the form of the system of equations:
= 'N': A*X = B (No transpose)
= 'T': A^T*X = B (Transpose)
= 'C': A^H*X = B (Conjugate transpose)

N (input) INTEGER
The order of the matrix A. N $\geq$ 0.

KL (input) INTEGER
The number of subdiagonals within the band of A. KL $\geq$ 0.

KU (input) INTEGER
The number of superdiagonals within the band of A. KU $\geq$ 0.

NRHS (input) INTEGER
The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS $\geq$ 0.

AB (input) REAL/COMPLEX array, dimension (LDAB,N)
The original band matrix A, stored in rows 1 to kl+ku+1.
The jth column of A is stored in the jth column of the array AB as follows:
AB(ku+1+i−j,j) = A(i,j) for max(1,j−ku)$\leq$i$\leq$min(n,j+kl).

LDAB (input) INTEGER
The leading dimension of the array AB. LDAB $\geq$ KL+KU+1.

AFB (input) REAL/COMPLEX array, dimension (LDAFB,N)
Details of the LU factorization of the band matrix A, as computed by SGBTRF/CGBTRF. U is stored as an upper triangular band matrix with kl+ku superdiagonals in rows 1 to kl+ku+1, and the multipliers used during the factorization are stored in rows kl+ku+2 to 2*kl+ku+1.

LDAFB (input) INTEGER
The leading dimension of the array AFB. LDAFB $\geq$ 2*KL*KU+1.

IPIV (input) INTEGER array, dimension (N)
The pivot indices from SGBTRF/CGBTRF; for 1$\leq$i$\leq$N, row i of the matrix was interchanged with row IPIV(i).

B (input) REAL/COMPLEX array, dimension (LDB,NRHS)
The right hand side matrix B.

LDB (input) INTEGER
The leading dimension of the array B. LDB $\geq$ max(1,N).

X (input/output) REAL/COMPLEX array, dimension (LDX,NRHS)
On entry, the solution matrix X, as computed by SGBTRS/CGBTRS.
On exit, the improved solution matrix X.

LDX (input) INTEGER
The leading dimension of the array X. LDX $\geq$ max(1,N).

FERR (output) REAL array, dimension (NRHS)
The estimated forward error bound for each solution vector X(j) (the jth column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) − XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR (output) REAL array, dimension (NRHS)
The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK *SGBRFS* (workspace) REAL array, dimension (3*N)
CGBRFS (workspace) COMPLEX array, dimension (2*N)

IWORK *SGBRFS only* (workspace) INTEGER array, dimension (N)

RWORK *CGBRFS only* (workspace) REAL array, dimension (N)

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SGBSV/CGBSV

```
SUBROUTINE SGBSV( N, KL, KU, NRHS, AB, LDAB, IPIV, B, LDB, INFO )
INTEGER          INFO, KL, KU, LDAB, LDB, N, NRHS
INTEGER          IPIV( * )
REAL             AB( LDAB, * ), B( LDB, * )

SUBROUTINE CGBSV( N, KL, KU, NRHS, AB, LDAB, IPIV, B, LDB, INFO )
INTEGER          INFO, KL, KU, LDAB, LDB, N, NRHS
INTEGER          IPIV( * )
COMPLEX          AB( LDAB, * ), B( LDB, * )
```

Purpose

SGBSV/CGBSV computes the solution to a real/complex system of linear equations A*X = B, where A is a band matrix of order n with kl subdiagonals and ku superdiagonals, and X and B are n-by-nrhs matrices.

The LU decomposition with partial pivoting and row interchanges is used to factor A as A = L*U, where L is a product of permutation and unit lower triangular matrices with kl subdiagonals, and U is upper triangular with kl+ku superdiagonals. The factored form of A is then used to solve the system of equations A*X = B.

Arguments

N (input) INTEGER
The number of linear equations, i.e., the order of the matrix A. $N \geq 0$.

KL (input) INTEGER
The number of subdiagonals within the band of A. $KL \geq 0$.

KU (input) INTEGER
The number of superdiagonals within the band of A. $KU \geq 0$.

NRHS (input) INTEGER
The number of right hand sides, i.e., the number of columns of the matrix B. $NRHS \geq 0$.

AB (input/output) REAL/COMPLEX array, dimension (LDAB,N)
On entry, the matrix A in band storage, in rows kl+1 to 2*kl+ku+1; rows 1 to kl of the array need not be set. The jth column of A is stored in the jth column of the array AB as follows:
$AB(kl+ku+1+i-j,j) = A(i,j)$ for $\max(1,j-ku) \leq i \leq \min(n,j+kl)$
On exit, details of the factorization: U is stored as an upper triangular band matrix with kl+ku superdiagonals in rows 1 to kl+ku+1, and the multipliers used during the factorization are stored in rows kl+ku+2 to 2*kl+ku+1.

LDAB (input) INTEGER
The leading dimension of the array AB. $LDAB \geq 2*KL+KU+1$.

IPIV (output) INTEGER array, dimension (N)
The pivot indices that define the permutation matrix P; row i of the matrix was interchanged with row IPIV(i).

B (input/output) REAL/COMPLEX array, dimension (LDB,NRHS)
On entry, the n-by-nrhs right hand side matrix B.
On exit, if INFO = 0, the n-by-nrhs solution matrix X.

LDB (input) INTEGER
The leading dimension of the array B. $LDB \geq \max(1,N)$.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, and the solution has not been computed.

SGBSVX/CGBSVX

```
      SUBROUTINE SGBSVX( FACT, TRANS, N, KL, KU, NRHS, AB, LDAB, AFB,
     $                   LDAFB, IPIV, EQUED, R, C, B, LDB, X, LDX,
     $                   RCOND, FERR, BERR, WORK, IWORK, INFO )
      CHARACTER          EQUED, FACT, TRANS
      INTEGER            INFO, KL, KU, LDAB, LDAFB, LDB, LDX, N, NRHS
      REAL               RCOND
      INTEGER            IPIV( * ), IWORK( * )
      REAL               AB( LDAB, * ), AFB( LDAFB, * ), B( LDB, * ),
     $                   BERR( * ), C( * ), FERR( * ), R( * ),
     $                   WORK( * ), X( LDX, * )

      SUBROUTINE CGBSVX( FACT, TRANS, N, KL, KU, NRHS, AB, LDAB, AFB,
     $                   LDAFB, IPIV, EQUED, R, C, B, LDB, X, LDX,
     $                   RCOND, FERR, BERR, WORK, RWORK, INFO )
      CHARACTER          EQUED, FACT, TRANS
      INTEGER            INFO, KL, KU, LDAB, LDAFB, LDB, LDX, N, NRHS
      REAL               RCOND
      INTEGER            IPIV( * )
      REAL               BERR( * ), C( * ), FERR( * ), R( * ),
     $                   RWORK( * )
      COMPLEX            AB( LDAB, * ), AFB( LDAFB, * ), B( LDB, * ),
     $                   WORK( * ), X( LDX, * )
```

Purpose

SGBSVX/CGBSVX uses the LU factorization to compute the solution to a real/complex system of linear equations $A*X = B$, $A^T*X = B$, or $A^H*X = B$, where A is a band matrix of order n with kl subdiagonals and ku superdiagonals, and X and B are n-by-nrhs matrices.

Error bounds on the solution and a condition estimate are also provided.

Description

The following steps are performed:

1. If FACT = 'E', real scaling factors are computed to equilibrate the system:

   TRANS = 'N': $diag(R)*A*diag(C)*diag(C)^{-1}*X = diag(R)*B$
   TRANS = 'T': $(diag(R)*A*diag(C))^T*diag(R)^{-1}*X = diag(C)*B$
   TRANS = 'C': $(diag(R)*A*diag(C))^H*diag(R)^{-1}*X = diag(C)*B$

   Whether or not the system will be equilibrated depends on the scaling of the matrix A, but if equilibration is used, A is overwritten by diag(R)*A*diag(C) and B by diag(R)*B (if TRANS='N') or diag(C)*B (if TRANS = 'T' or 'C').

2. If FACT = 'N' or 'E', the LU decomposition is used to factor the matrix A (after equilibration if FACT = 'E') as A = L*U, where L is a product of permutation and unit lower triangular matrices with kl subdiagonals, and U is upper triangular with kl+ku superdiagonals.

3. If some U(i,i)=0, so that U is exactly singular, then the routine returns with INFO = i. Otherwise, the factored form of A is used to estimate the condition number of the matrix A. If the reciprocal of the condition number is less than machine precision, INFO = N+1 is returned as a warning, but the routine still goes on to solve for X and compute error bounds as described below.

4. The system of equations is solved for X using the factored form of A.

5. Iterative refinement is applied to improve the computed solution matrix and calculate error bounds and backward error estimates for it.

6. If equilibration was used, the matrix X is premultiplied by diag(C) (if TRANS = 'N') or diag(R) (if TRANS = 'T' or 'C') so that it solves the original system before equilibration.

Arguments

FACT (input) CHARACTER*1
Specifies whether or not the factored form of the matrix A is supplied on entry, and if not, whether the matrix A should be equilibrated before it is factored.
= 'F': On entry, AFB and IPIV contain the factored form of A. If EQUED $\neq$ 'N', the matrix A has been equilibrated with scaling factors given by R and C.
AB, AFB, and IPIV are not modified.
= 'N': The matrix A will be copied to AFB and factored.
= 'E': The matrix A will be equilibrated if necessary, then copied to AFB and factored.

TRANS (input) CHARACTER*1
Specifies the form of the system of equations:
= 'N': A*X = B (No transpose)
= 'T': A^T*X = B (Transpose)
= 'C': A^H*X = B (Conjugate Transpose)

N (input) INTEGER
The number of linear equations, i.e., the order of the matrix A. N $\geq$ 0.

KL (input) INTEGER
The number of subdiagonals within the band of A. KL $\geq$ 0.

KU (input) INTEGER
The number of superdiagonals within the band of A. KU $\geq$ 0.

NRHS (input) INTEGER
The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS $\geq$ 0.

AB (input/output) REAL/COMPLEX array, dimension (LDAB,N)
On entry, the matrix A in band storage, in rows 1 to kl+ku+1. The jth column of A is stored in the jth column of the array AB as follows:
AB(ku+1+i−j,j) = A(i,j) for max(1,j−ku)$\leq$i$\leq$min(n,j+kl)
If FACT = 'F' and EQUED is not 'N', then A must have been equilibrated by the scaling factors in R and/or C. AB is not modified if FACT = 'F' or 'N', or if FACT = 'E' and EQUED = 'N' on exit.
On exit, if EQUED $\neq$ 'N', A is scaled as follows:
EQUED = 'R': A := diag(R)*A;
EQUED = 'C': A := A*diag(C);
EQUED = 'B': A := diag(R)*A*diag(C).

LDAB (input) INTEGER
The leading dimension of the array AB. LDAB $\geq$ KL+KU+1.

AFB (input or output) REAL/COMPLEX array, dimension (LDAFB,N)
If FACT = 'F', then AFB is an input argument and on entry contains details of the LU factorization of the band matrix A, as computed by SGBTRF/CGBTRF. U is stored as an upper triangular band matrix with kl+ku superdiagonals in rows 1 to kl+ku+1, and the multipliers used during the factorization are stored in rows kl+ku+2 to 2*kl+ku+1. If EQUED $\neq$ 'N', then AFB is the factored form of the equilibrated matrix A.
If FACT = 'N', then AFB is an output argument and on exit returns details of the LU factorization of A.
If FACT = 'E', then AFB is an output argument and on exit returns details of the LU factorization of the equilibrated matrix A (see the description of AB for the form of the equilibrated matrix).

LDAFB (input) INTEGER
The leading dimension of the array AFB. LDAFB $\geq$ 2*KL+KU+1.

IPIV (input or output) INTEGER array, dimension (N)
If FACT = 'F', then IPIV is an input argument and on entry contains the pivot indices from the factorization A = L*U as computed by SGBTRF/CGBTRF; row i of the matrix was interchanged with row IPIV(i).
If FACT = 'N', then IPIV is an output argument and on exit contains the pivot indices from the factorization A = L*U of the original matrix A.
If FACT = 'E', then IPIV is an output argument and on exit contains the pivot indices from the factorization A = L*U of the equilibrated matrix A.

EQUED (input or output) CHARACTER*1
Specifies the form of equilibration that was done:
= 'N': No equilibration (always true if FACT = 'N').
= 'R': Row equilibration, i.e., A has been premultiplied by diag(R).
= 'C': Column equilibration, i.e., A has been postmultiplied by diag(C).
= 'B': Both row and column equilibration, i.e., A has been replaced by diag(R)*A*diag(C).
EQUED is an input argument if FACT = 'F'; otherwise, it is an output argument.

R (input or output) REAL array, dimension (N)
The row scale factors for A. If EQUED = 'R' or 'B', A is multiplied on the left by diag(R); if EQUED = 'N' or 'C', R is not accessed. R is an input argument if FACT = 'F'; otherwise, R is an output argument.

If FACT = 'F' and EQUED = 'R' or 'B', each element of R must be positive.

C (input or output) REAL array, dimension (N)
The column scale factors for A. If EQUED = 'C' or 'B', A is multiplied on the right by diag(C); if EQUED = 'N' or 'R', C is not accessed. C is an input argument if FACT = 'F'; otherwise, C is an output argument. If FACT = 'F' and EQUED = 'C' or 'B', each element of C must be positive.

B (input/output) REAL/COMPLEX array, dimension (LDB,NRHS)
On entry, the right hand side matrix B.
On exit,
if EQUED = 'N', B is not modified;
if TRANS = 'N' and EQUED = 'R' or 'B', B is overwritten by diag(R)*B;
if TRANS = 'T' or 'C' and EQUED = 'C' or 'B', B is overwritten by diag(C)*B.

LDB (input) INTEGER
The leading dimension of the array B. LDB $\geq$ max(1,N).

X (output) REAL/COMPLEX array, dimension (LDX,NRHS)
If INFO = 0 or INFO = N+1, the N-by-NRHS solution matrix X to the original system of equations. Note that A and B are modified on exit if EQUED .ne. 'N', and the solution to the equilibrated system is $\mathrm{diag}(C)^{-1}*X$ if TRANS = 'N' and EQUED = 'C' or 'B', or $\mathrm{diag}(R)^{-1}*X$ if TRANS = 'T' or 'C' and EQUED = 'R' or 'B'.

LDX (input) INTEGER
The leading dimension of the array X. LDX $\geq$ max(1,N).

RCOND (output) REAL
The estimate of the reciprocal condition number of the matrix A after equilibration (if done). If RCOND is less than the machine precision (in particular, if RCOND = 0), the matrix is singular to working precision. This condition is indicated by a return code of INFO > 0.

FERR (output) REAL array, dimension (NRHS)
The estimated forward error bound for each solution vector X(j) (the jth column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) − XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR (output) REAL array, dimension (NRHS)
The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK *SGBSVX* (workspace/output) REAL array, dimension (3*N)
CGBSVX (workspace) COMPLEX array, dimension (2*N)
SGBSVX only
On exit, WORK(1) contains the reciprocal pivot growth factor $||A||/||U||$. The "max absolute element" norm is used. If WORK(1) is much less than 1, then the stability of the LU factorization of the (equilibrated) matrix A could be poor. This also means that the solution X, condition estimator RCOND, and forward error bound FERR could be unreliable. If factorization fails with 0<INFO$\leq$N, then WORK(1) contains the reciprocal pivot growth factor for the leading INFO columns of A.

IWORK *SGBSVX only* (workspace) INTEGER array, dimension (N)

RWORK *CGBSVX only* (workspace/output) REAL array, dimension (N)
On exit, RWORK(1) contains the reciprocal pivot growth factor $||A||/||U||$. The "max absolute element" norm is used. If RWORK(1) is much less than 1, then the stability of the LU factorization of the (equilibrated) matrix A could be poor. This also means that the solution X, condition estimator RCOND, and forward error bound FERR could be unreliable. If factorization fails with 0<INFO$\leq$N, then RWORK(1) contains the reciprocal pivot growth factor for the leading INFO columns of A.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, and i is
$\leq$ N: U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned.
= N+1: U is nonsingular, but RCOND is less than machine precision, meaning that the matrix is singular to working precision. Nevertheless, the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the value of RCOND would suggest.

SGBTRF/CGBTRF

```
SUBROUTINE SGBTRF( M, N, KL, KU, AB, LDAB, IPIV, INFO )
INTEGER            INFO, KL, KU, LDAB, M, N
INTEGER            IPIV( * )
REAL               AB( LDAB, * )

SUBROUTINE CGBTRF( M, N, KL, KU, AB, LDAB, IPIV, INFO )
INTEGER            INFO, KL, KU, LDAB, M, N
INTEGER            IPIV( * )
COMPLEX            AB( LDAB, * )
```

Purpose

SGBTRF/CGBTRF computes an LU factorization of a real/complex m-by-n band matrix A using partial pivoting with row interchanges.

Arguments

M (input) INTEGER
The number of rows of the matrix A. $M \geq 0$.

N (input) INTEGER
The number of columns of the matrix A. $N \geq 0$.

KL (input) INTEGER
The number of subdiagonals within the band of A. $KL \geq 0$.

KU (input) INTEGER
The number of superdiagonals within the band of A. $KU \geq 0$.

AB (input/output) REAL/COMPLEX array, dimension (LDAB,N)
On entry, the matrix A in band storage, in rows kl+1 to 2*kl+ku+1; rows 1 to kl of the array need not be set. The jth column of A is stored in the jth column of the array AB as follows:
AB(kl+ku+1+i−j,j) = A(i,j) for max(1,j−ku)≤i≤min(m,j+kl)
On exit, details of the factorization: U is stored as an upper triangular band matrix with kl+ku superdiagonals in rows 1 to kl+ku+1, and the multipliers used during the factorization are stored in rows kl+ku+2 to 2*kl+ku+1.

LDAB (input) INTEGER
The leading dimension of the array AB. $LDAB \geq 2*KL+KU+1$.

IPIV (output) INTEGER array, dimension (min(M,N))
The pivot indices; for $1 \leq i \leq \min(M,N)$, row i of the matrix was interchanged with row IPIV(i).

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, and division by zero will occur if it is used to solve a system of equations.

SGBTRS/CGBTRS

```
      SUBROUTINE SGBTRS( TRANS, N, KL, KU, NRHS, AB, LDAB, IPIV, B, LDB,
     $                   INFO )
      CHARACTER          TRANS
      INTEGER            INFO, KL, KU, LDAB, LDB, N, NRHS
      INTEGER            IPIV( * )
      REAL               AB( LDAB, * ), B( LDB, * )
```

```
      SUBROUTINE CGBTRS( TRANS, N, KL, KU, NRHS, AB, LDAB, IPIV, B, LDB,
     $                   INFO )
      CHARACTER          TRANS
      INTEGER            INFO, KL, KU, LDAB, LDB, N, NRHS
      INTEGER            IPIV( * )
      COMPLEX            AB( LDAB, * ), B( LDB, * )
```

Purpose

SGBTRS/CGBTRS solves a system of linear equations A*X = B, A^T*X = B, or A^H*X = B, with a general band matrix A using the LU factorization computed by SGBTRF/CGBTRF.

Arguments

TRANS (input) CHARACTER*1
Specifies the form of the system of equations:
= 'N': A*X = B (No transpose)
= 'T': A^T*X = B (Transpose)
= 'C': A^H*X = B (Conjugate Transpose)

N (input) INTEGER
The order of the matrix A. $N \geq 0$.

KL (input) INTEGER
The number of subdiagonals within the band of A. $KL \geq 0$.

KU (input) INTEGER
The number of superdiagonals within the band of A. $KU \geq 0$.

NRHS (input) INTEGER
The number of right hand sides, i.e., the number of columns of the matrix B. $NRHS \geq 0$.

AB (input) REAL/COMPLEX array, dimension (LDAB,N)
Details of the LU factorization of the band matrix A, as computed by SGBTRF/CGBTRF. U is stored as an upper triangular band matrix with kl+ku superdiagonals in rows 1 to kl+ku+1, and the multipliers used during the factorization are stored in rows kl+ku+2 to 2*kl+ku+1.

LDAB (input) INTEGER
The leading dimension of the array AB. $LDAB \geq 2*KL+KU+1$.

IPIV (input) INTEGER array, dimension (N)
The pivot indices; for $1 \leq i \leq N$, row i of the matrix was interchanged with row IPIV(i).

B (input/output) REAL/COMPLEX array, dimension (LDB,NRHS)
On entry, the right hand side matrix B.
On exit, the solution matrix X.

LDB (input) INTEGER
The leading dimension of the array B. $LDB \geq \max(1,N)$.

INFO (output) INTEGER
= 0: successful exit

< 0: if INFO = −i, the ith argument had an illegal value.

SGEBAK/CGEBAK

```
      SUBROUTINE SGEBAK( JOB, SIDE, N, ILO, IHI, SCALE, M, V, LDV,
     $                   INFO )
      CHARACTER          JOB, SIDE
      INTEGER            IHI, ILO, INFO, LDV, M, N
      REAL               V( LDV, * ), SCALE( * )

      SUBROUTINE CGEBAK( JOB, SIDE, N, ILO, IHI, SCALE, M, V, LDV,
     $                   INFO )
      CHARACTER          JOB, SIDE
      INTEGER            IHI, ILO, INFO, LDV, M, N
      REAL               SCALE( * )
      COMPLEX            V( LDV, * )
```

Purpose

SGEBAK/CGEBAK forms the right or left eigenvectors of a real/complex general matrix by backward transformation on the computed eigenvectors of the balanced matrix output by SGEBAL/CGEBAL.

Arguments

JOB (input) CHARACTER*1
Specifies the type of backward transformation required:
= 'N': do nothing, return immediately;
= 'P': do backward transformation for permutation only;
= 'S': do backward transformation for scaling only;
= 'B': do backward transformations for both permutation and scaling.
JOB must be the same as the argument JOB supplied to SGEBAL/CGEBAL.

SIDE (input) CHARACTER*1
= 'R': V contains right eigenvectors;
= 'L': V contains left eigenvectors.

N (input) INTEGER
The number of rows of the matrix V. $N \geq 0$.

ILO, IHI (input) INTEGER
The integers ILO and IHI determined by SGEBAL/CGEBAL.
$1 \leq ILO \leq IHI \leq N$, if $N > 0$; ILO = 1 and IHI = 0, if N = 0.

SCALE (input) REAL array, dimension (N)
Details of the permutation and scaling factors, as returned by SGEBAL/CGEBAL.

M (input) INTEGER
The number of columns of the matrix V. $M \geq 0$.

V (input/output) REAL/COMPLEX array, dimension (LDV,M)
On entry, the matrix of right or left eigenvectors to be transformed, as returned by SHSEIN/CHSEIN or STREVC/CTREVC.
On exit, V is overwritten by the transformed eigenvectors.

LDV (input) INTEGER
The leading dimension of the array V. $LDV \geq \max(1,N)$.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SGEBAL/CGEBAL

```
      SUBROUTINE SGEBAL( JOB, N, A, LDA, ILO, IHI, SCALE, INFO )
      CHARACTER          JOB
      INTEGER            IHI, ILO, INFO, LDA, N
      REAL               A( LDA, * ), SCALE( * )

      SUBROUTINE CGEBAL( JOB, N, A, LDA, ILO, IHI, SCALE, INFO )
      CHARACTER          JOB
      INTEGER            IHI, ILO, INFO, LDA, N
      REAL               SCALE( * )
      COMPLEX            A( LDA, * )
```

Purpose

SGEBAL/CGEBAL balances a general real/complex matrix A. This involves, first, permuting A by a similarity transformation to isolate eigenvalues in the first 1 to ilo−1 and last ihi+1 to n elements on the diagonal; and second, applying a diagonal similarity transformation to rows and columns ilo to ihi to make the rows and columns as close in norm as possible. Both steps are optional.

Balancing may reduce the 1-norm of the matrix, and improve the accuracy of the computed eigenvalues and/or eigenvectors.

Arguments

JOB (input) CHARACTER*1
Specifies the operations to be performed on A:
= 'N': none: simply set ILO = 1, IHI = N, SCALE(I) = 1.0 for i = 1,...,N;
= 'P': permute only;
= 'S': scale only;
= 'B': both permute and scale.

N (input) INTEGER
The order of the matrix A. $N \geq 0$.

A (input/output) REAL/COMPLEX array, dimension (LDA,N)
On entry, the input matrix A.

On exit, A is overwritten by the balanced matrix.
If JOB = 'N', A is not referenced.

LDA (input) INTEGER
The leading dimension of the array A. LDA $\geq$ max(1,N).

ILO, IHI (output) INTEGER
ILO and IHI are set to integers such that on exit A(i,j) = 0 if i > j and j = 1,...,ilo−1 or i = ihi+1,...,n.
If JOB = 'N' or 'S', ILO = 1 and IHI = N.

SCALE (output) REAL array, dimension (N)
Details of the permutations and scaling factors applied to A. If P(j) is the index of the row and column interchanged with row and column j, and D(j) is the scaling factor applied to row and column j, then

```
SCALE(j)  = P(j)   for j = 1,...,ilo-1
          = D(j)   for j = ilo,...,ihi
          = P(j)   for j = ihi+1,...,n.
```

The order in which the interchanges are made is n to ihi+1, then 1 to ilo−1.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SGEBRD/CGEBRD

```
      SUBROUTINE SGEBRD( M, N, A, LDA, D, E, TAUQ, TAUP, WORK, LWORK,
     $                   INFO )
      INTEGER            INFO, LDA, LWORK, M, N
      REAL               A( LDA, * ), D( * ), E( * ), TAUP( * ),
     $                   TAUQ( * ), WORK( LWORK )

      SUBROUTINE CGEBRD( M, N, A, LDA, D, E, TAUQ, TAUP, WORK, LWORK,
     $                   INFO )
      INTEGER            INFO, LDA, LWORK, M, N
      REAL               D( * ), E( * )
      COMPLEX            A( LDA, * ), TAUP( * ), TAUQ( * ),
     $                   WORK( LWORK )
```

Purpose

SGEBRD/CGEBRD reduces a general real/complex m-by-n matrix A to upper or lower bidiagonal form B by an orthogonal/unitary transformation: $Q^H * A * P = B$.

If m $\geq$ n, B is upper bidiagonal; if m < n, B is lower bidiagonal.

Arguments

M (input) INTEGER
The number of rows in the matrix A. M $\geq$ 0.

N (input) INTEGER
The number of columns in the matrix A. N $\geq$ 0.

A (input/output) REAL/COMPLEX array, dimension (LDA,N)
On entry, the m-by-n general matrix to be reduced.
On exit, if m $\geq$ n, the diagonal and the first superdiagonal are overwritten with the upper bidiagonal matrix B; the elements below the diagonal, with the array TAUQ, represent the orthogonal/unitary matrix Q as a product of elementary reflectors, and the elements above the first superdiagonal, with the array TAUP, represent the orthogonal/unitary matrix P as a product of elementary reflectors; if m < n, the diagonal and the first subdiagonal are overwritten with the lower bidiagonal matrix B; the elements below the first subdiagonal, with the array TAUQ, represent the orthogonal/unitary matrix Q as a product of elementary reflectors, and the elements above the diagonal, with the array TAUP, represent the orthogonal/unitary matrix P as a product of elementary reflectors.

LDA (input) INTEGER
The leading dimension of the array A. LDA $\geq$ max(1,M).

D (output) REAL array, dimension (min(M,N))
The diagonal elements of the bidiagonal matrix B: D(i) = A(i,i).

E (output) REAL array, dimension (min(M,N)−1)
The off-diagonal elements of the bidiagonal matrix B:
if m $\geq$ n, E(i) = A(i,i+1) for i = 1,2,...,n−1;
if m < n, E(i) = A(i+1,i) for i = 1,2,...,m−1.

TAUQ (output) REAL/COMPLEX array dimension (min(M,N))
The scalar factors of the elementary reflectors which represent the orthogonal/unitary matrix Q.

TAUP (output) REAL/COMPLEX array, dimension (min(M,N))
The scalar factors of the elementary reflectors which represent the orthogonal/unitary matrix P.

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The length of the array WORK. LWORK $\geq$ max(1,M,N). For optimum performance LWORK $\geq$ (M+N)*NB, where NB is the optimal blocksize.

If LWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SGECON/CGECON

```
      SUBROUTINE SGECON( NORM, N, A, LDA, ANORM, RCOND, WORK, IWORK,
     $                   INFO )
      CHARACTER          NORM
      INTEGER            INFO, LDA, N
      REAL               ANORM, RCOND
      INTEGER            IWORK( * )
      REAL               A( LDA, * ), WORK( * )

      SUBROUTINE CGECON( NORM, N, A, LDA, ANORM, RCOND, WORK, RWORK,
     $                   INFO )
      CHARACTER          NORM
      INTEGER            INFO, LDA, N
      REAL               ANORM, RCOND
      REAL               RWORK( * )
      COMPLEX            A( LDA, * ), WORK( * )
```

Purpose

SGECON/CGECON estimates the reciprocal of the condition number of a general real/complex matrix A, in either the 1-norm or the infinity-norm, using the LU factorization computed by SGETRF/CGETRF.

An estimate is obtained for $\|A^{-1}\|$, and the reciprocal of the condition number is computed as RCOND = $1/(\|A\| * \|A^{-1}\|)$.

Arguments

NORM (input) CHARACTER*1
Specifies whether the 1−norm condition number or the infinity-norm condition number is required:
= '1' or 'O': 1-norm;
= 'I': Infinity-norm.

N (input) INTEGER
The order of the matrix A. N $\geq$ 0.

A (input) REAL/COMPLEX array, dimension (LDA,N)
The factors L and U from the factorization A = P*L*U as computed by SGETRF/CGETRF.

LDA (input) INTEGER
The leading dimension of the array A. LDA $\geq$ max(1,N).

ANORM (input) REAL
If NORM = '1' or 'O', the 1-norm of the original matrix A.
If NORM = 'I', the infinity-norm of the original matrix A.

RCOND (output) REAL
The reciprocal of the condition number of the matrix A, computed as RCOND = $1/(\|A\| * \|A^{-1}\|)$.

WORK *SGECON* (workspace) REAL array, dimension (4*N)
CGECON (workspace) COMPLEX array, dimension (2*N)

IWORK *SGECON only* (workspace) INTEGER array, dimension (N)

RWORK *CGECON only* (workspace) REAL array, dimension (2*N)

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SGEEQU/CGEEQU

```
      SUBROUTINE SGEEQU( M, N, A, LDA, R, C, ROWCND, COLCND, AMAX,
     $                   INFO )
      INTEGER            INFO, LDA, M, N
      REAL               AMAX, COLCND, ROWCND
      REAL               A( LDA, * ), C( * ), R( * )

      SUBROUTINE CGEEQU( M, N, A, LDA, R, C, ROWCND, COLCND, AMAX,
     $                   INFO )
      INTEGER            INFO, LDA, M, N
      REAL               AMAX, COLCND, ROWCND
      REAL               C( * ), R( * )
      COMPLEX            A( LDA, * )
```

Purpose

SGEEQU/CGEEQU computes row and column scalings intended to equilibrate an m-by-n matrix A and reduce its condition number. R returns the row scale factors and C the column scale factors, chosen to try to make the largest element in each row and column of the matrix B with elements B(i,j) = R(i)*A(i,j)*C(j) have absolute value 1.

R(i) and C(j) are restricted to be between SMLNUM = smallest safe number and BIGNUM = largest safe number. Use of these scaling factors is not guaranteed to reduce the condition number of A but works well in practice.

Arguments

M (input) INTEGER
The number of rows of the matrix A. M $\geq$ 0.

N (input) INTEGER
The number of columns of the matrix A. N $\geq$ 0.

A (input) REAL/COMPLEX array, dimension (LDA,N)
The m-by-n matrix whose equilibration factors are to be computed.

LDA (input) INTEGER
The leading dimension of the array A. LDA $\geq$ max(1,M).

R (output) REAL array, dimension (M)
If INFO = 0 or INFO > M, R contains the row scale factors for A.

C (output) REAL array, dimension (N)
If INFO = 0, C contains the column scale factors for A.

ROWCND (output) REAL
If INFO = 0 or INFO > M, ROWCND contains the ratio of the smallest R(i) to the largest R(i). If ROWCND $\geq$ 0.1 and AMAX is neither too large nor too small, it is not worth scaling by R.

COLCND (output) REAL
If INFO = 0, COLCND contains the ratio of the smallest C(i) to the largest C(i). If COLCND $\geq$ 0.1, it is not worth scaling by C.

AMAX (output) REAL
Absolute value of largest matrix element. If AMAX is very close to overflow or very close to underflow, the matrix should be scaled.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, and i is
$\leq$ M: the ith row of A is exactly zero
> M: the $(i\text{-}M)^{th}$ column of A is exactly zero

SGEES/CGEES

```
      SUBROUTINE SGEES( JOBVS, SORT, SELECT, N, A, LDA, SDIM, WR, WI,
     $                  VS, LDVS, WORK, LWORK, BWORK, INFO )
      CHARACTER          JOBVS, SORT
      INTEGER            INFO, LDA, LDVS, LWORK, N, SDIM
      LOGICAL            BWORK( * )
      REAL               A( LDA, * ), VS( LDVS, * ), WI( * ), WORK( * ),
     $                   WR( * )
      LOGICAL            SELECT
      EXTERNAL           SELECT

      SUBROUTINE CGEES( JOBVS, SORT, SELECT, N, A, LDA, SDIM, W, VS,
     $                  LDVS, WORK, LWORK, RWORK, BWORK, INFO )
      CHARACTER          JOBVS, SORT
      INTEGER            INFO, LDA, LDVS, LWORK, N, SDIM
      LOGICAL            BWORK( * )
      REAL               RWORK( * )
      COMPLEX            A( LDA, * ), VS( LDVS, * ), W( * ), WORK( * )
      LOGICAL            SELECT
      EXTERNAL           SELECT
```

Purpose

SGEES/CGEES computes for an n-by-n real/complex nonsymmetric matrix A, the eigenvalues, the real-Schur/Schur form T, and, optionally, the matrix of Schur vectors Z. This gives the Schur factorization $A = Z*T*Z^H$.

Optionally, it also orders the eigenvalues on the diagonal of the real-Schur/Schur form so that selected eigenvalues are at the top left. The leading columns of Z then form an orthonormal basis for the invariant subspace corresponding to the selected eigenvalues.

A real matrix is in real-Schur form if it is upper quasi-triangular with 1-by-1 and 2-by-2 diagonal blocks. 2-by-2 diagonal blocks will be standardized in the form $\begin{pmatrix} a & b \\ c & a \end{pmatrix}$ where b*c < 0. The eigenvalues of such a block are $a \pm \sqrt{bc}$.

A complex matrix is in Schur form if it is upper triangular.

Arguments

JOBVS (input) CHARACTER*1
= 'N': Schur vectors are not computed;
= 'V': Schur vectors are computed.

SORT (input) CHARACTER*1
Specifies whether or not to order the eigenvalues on the diagonal of the Schur form.
= 'N': Eigenvalues are not ordered;
= 'S': Eigenvalues are ordered (see SELECT).

SELECT *SGEES* (input) LOGICAL FUNCTION of two REAL arguments
CGEES (input) LOGICAL FUNCTION of one COMPLEX argument
SELECT must be declared EXTERNAL in the calling subroutine.
If SORT = 'S', SELECT is used to select eigenvalues to sort to the top left of the Schur form.
If SORT = 'N', SELECT is not referenced.
SGEES
An eigenvalue WR(j) + i*WI(j) is selected if SELECT(WR(j),WI(j)) is true. A complex eigenvalue is selected if either SELECT(WR(j),WI(j)) or SELECT(WR(j),−WI(j)) is true: i.e., if either one of a complex conjugate pair of eigenvalues is selected, then both are. Note that a selected complex eigenvalue may no longer satisfy SELECT(WR(j),WI(j)) = .TRUE. after ordering, since ordering may change the value of complex eigenvalues (especially if the eigenvalue is ill-conditioned); in this case INFO may be set to N+2 (see INFO below).
CGEES
An eigenvalue W(j) is selected if SELECT(W(j)) is true.

N (input) INTEGER
The order of the matrix A. N $\geq$ 0.

A (input/output) REAL/COMPLEX array, dimension (LDA,N)
On entry, the n-by-n matrix A.

On exit, A has been overwritten by its real-Schur/Schur form T.

LDA (input) INTEGER
The leading dimension of the array A. LDA $\geq$ max(1,N).

SDIM (output) INTEGER
If SORT = 'N', SDIM = 0.
If SORT = 'S', SDIM = number of eigenvalues (after sorting) for which SELECT is true.
SGEES only
(Complex conjugate pairs for which SELECT is true for either eigenvalue count as 2.)

WR, WI *SGEES only* (output) REAL/COMPLEX array, dimension (N)
WR and WI contain the real and imaginary parts, respectively, of the computed eigenvalues, in the same order that they appear on the diagonal of the output Schur form T. Complex conjugate pairs of eigenvalues appear consecutively with the eigenvalue having the positive imaginary part first.

W *CGEES only* (output) COMPLEX array, dimension (N)
W contains the computed eigenvalues, in the same order that they appear on the diagonal of the output Schur form T.

VS (output) REAL/COMPLEX array, dimension (LDVS,N)
If JOBVS = 'V', VS contains the orthogonal/unitary matrix Z of Schur vectors.
If JOBVS = 'N', VS is not referenced.

LDVS (input) INTEGER
The leading dimension of the array VS. LDVS $\geq$ 1; if JOBVS = 'V', LDVS $\geq$ N.

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) contains the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK.
LWORK $\geq$ max(1,3*N) (*SGEES*)
LWORK $\geq$ max(1,2*N) (*CGEES*).
For good performance, LWORK must generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK *CGEES only* (workspace) REAL array, dimension (N)

BWORK (workspace) LOGICAL array, dimension (N)
Not referenced if SORT = 'N'.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = $-$i, the ith argument had an illegal value.
> 0: if INFO = i, and i is
$\leq$ N: the QR algorithm failed to compute all the eigenvalues; elements 1:ilo$-$1 and i+1:n of WR and WI (*SGEES*) or W (*CGEES*) contain those eigenvalues which have converged; if JOBVS = 'V', VS contains the matrix which reduces A to its partially converged Schur form.
= N+1: the eigenvalues could not be reordered because some eigenvalues were too close to separate (the problem is very ill-conditioned);
= N+2: after reordering, roundoff changed values of some complex eigenvalues so that leading eigenvalues in the Schur form no longer satisfy SELECT=.TRUE. This could also be caused by underflow due to scaling.

SGEESX/CGEESX

```
      SUBROUTINE SGEESX( JOBVS, SORT, SELECT, SENSE, N, A, LDA, SDIM,
     $                   WR, WI, VS, LDVS, RCONDE, RCONDV, WORK, LWORK,
     $                   IWORK, LIWORK, BWORK, INFO )
      CHARACTER          JOBVS, SENSE, SORT
      INTEGER            INFO, LDA, LDVS, LIWORK, LWORK, N, SDIM
      REAL               RCONDE, RCONDV
      LOGICAL            BWORK( * )
      INTEGER            IWORK( * )
      REAL               A( LDA, * ), VS( LDVS, * ), WI( * ), WORK( * ),
     $                   WR( * )
      LOGICAL            SELECT
      EXTERNAL           SELECT
      SUBROUTINE CGEESX( JOBVS, SORT, SELECT, SENSE, N, A, LDA, SDIM, W,
     $                   VS, LDVS, RCONDE, RCONDV, WORK, LWORK, RWORK,
     $                   BWORK, INFO )
      CHARACTER          JOBVS, SENSE, SORT
      INTEGER            INFO, LDA, LDVS, LWORK, N, SDIM
      REAL               RCONDE, RCONDV
      LOGICAL            BWORK( * )
      REAL               RWORK( * )
      COMPLEX            A( LDA, * ), VS( LDVS, * ), W( * ), WORK( * )
      LOGICAL            SELECT
      EXTERNAL           SELECT
```

Purpose

SGEESX/CGEESX computes for an n-by-n real/complex nonsymmetric matrix A, the eigenvalues, the real-Schur/Schur form T, and, optionally, the matrix of Schur vectors Z. This gives the Schur factorization $A = Z*T*Z^H$.

Optionally, it also orders the eigenvalues on the diagonal of the real-Schur/Schur

form so that selected eigenvalues are at the top left; computes a reciprocal condition number for the average of the selected eigenvalues (RCONDE); and computes a reciprocal condition number for the right invariant subspace corresponding to the selected eigenvalues (RCONDV). The leading columns of Z form an orthonormal basis for this invariant subspace.

For further explanation of the reciprocal condition numbers RCONDE and RCONDV, see Section 4.10 (where these quantities are called s and sep respectively).

A real matrix is in real-Schur form if it is upper quasi-triangular with 1-by-1 and 2-by-2 diagonal blocks. 2-by-2 diagonal blocks will be standardized in the form $\begin{pmatrix} a & b \\ c & a \end{pmatrix}$ where b*c < 0. The eigenvalues of such a block are $a \pm \sqrt{bc}$.

A complex matrix is in Schur form if it is upper triangular.

Arguments

JOBVS (input) CHARACTER*1
= 'N': Schur vectors are not computed;
= 'V': Schur vectors are computed.

SORT (input) CHARACTER*1
Specifies whether or not to order the eigenvalues on the diagonal of the Schur form.
= 'N': Eigenvalues are not ordered;
= 'S': Eigenvalues are ordered (see SELECT).

SELECT *SGEESX* (input) LOGICAL FUNCTION of two REAL arguments
CGEESX (input) LOGICAL FUNCTION of one COMPLEX argument
SELECT must be declared EXTERNAL in the calling subroutine.
If SORT = 'S', SELECT is used to select eigenvalues to sort to the top left of the Schur form.
If SORT = 'N', SELECT is not referenced.
SGEESX
An eigenvalue WR(j) + i*WI(j) is selected if SELECT(WR(j),WI(j)) is true. A complex eigenvalue is selected if either SELECT(WR(j),WI(j)) or SELECT(WR(j),−WI(j)) is true: i.e., if either one of a complex conjugate pair of eigenvalues is selected, then both are. Note that a selected complex eigenvalue may no longer satisfy SELECT(WR(j),WI(j)) = .TRUE. after ordering, since ordering may change the value of complex eigenvalues (especially if the eigenvalue is ill-conditioned); in this case INFO may be set to N+2 (see INFO below).
CGEESX
An eigenvalue W(j) is selected if SELECT(W(j)) is true.

SENSE (input) CHARACTER*1
Determines which reciprocal condition numbers are computed.
= 'N': None are computed;
= 'E': Computed for average of selected eigenvalues only;
= 'V': Computed for selected right invariant subspace only;
= 'B': Computed for both.
If SENSE = 'E', 'V' or 'B', SORT must equal 'S'.

N (input) INTEGER
The order of the matrix A. N ≥ 0.

A (input/output) REAL/COMPLEX array, dimension (LDA, N)
On entry, the n-by-n matrix A.
On exit, A is overwritten by its real-Schur/Schur form T.

LDA (input) INTEGER
The leading dimension of the array A. LDA ≥ max(1,N).

SDIM (output) INTEGER
If SORT = 'N', SDIM = 0.
If SORT = 'S', SDIM = number of eigenvalues (after sorting) for which SELECT is true.
SGEESX only
(Complex conjugate pairs for which SELECT is true for either eigenvalue count as 2.)

WR, WI *SGEESX only* (output) REAL array, dimension (N)
WR and WI contain the real and imaginary parts, respectively, of the computed eigenvalues, in the same order that they appear on the diagonal of the output Schur form T. Complex conjugate pairs of eigenvalues appear consecutively with the eigenvalue having the positive imaginary part first.

W *CGEESX only* (output) COMPLEX array, dimension (N)
W contains the computed eigenvalues, in the same order that they appear on the diagonal of the output Schur form T.

VS (output) REAL/COMPLEX array, dimension (LDVS,N)
If JOBVS = 'V', VS contains the orthogonal/unitary matrix Z of Schur vectors.
If JOBVS = 'N', VS is not referenced.

LDVS (input) INTEGER
The leading dimension of the array VS. LDVS ≥ 1; if JOBVS = 'V', LDVS ≥ N.

RCONDE (output) REAL
If SENSE = 'E' or 'B', RCONDE contains the reciprocal condition number for the average of the selected eigenvalues.
Not referenced if SENSE = 'N' or 'V'.

RCONDV (output) REAL
If SENSE = 'V' or 'B', RCONDV contains the reciprocal condition number for the selected right invariant subspace.
Not referenced if SENSE = 'N' or 'E'.

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK.

LWORK $\geq$ max(1,3*N) (*SGEESX*)
LWORK $\geq$ max(1,2*N) (*CGEESX*).
Also, if SENSE = 'E' or 'V' or 'B',
LWORK $\geq$ N+2*SDIM*(N−SDIM) (*SGEESX*)
LWORK $\geq$ 2*SDIM*(N−SDIM) (*CGEESX*),
where SDIM is the number of selected eigenvalues computed by this routine. Note that 2*SDIM*(N−SDIM) $\leq$ N*N/2.
For good performance, LWORK must generally be larger.

IWORK *SGEESX only* (workspace) INTEGER array, dimension (LIWORK)
Not referenced if SENSE = 'N' or 'E'.

LIWORK *SGEESX only* (input) INTEGER
The dimension of the array IWORK. LIWORK $\geq$ 1; if SENSE = 'V' or 'B', LIWORK $\geq$ SDIM*(N−SDIM).

RWORK *CGEESX only* (workspace) REAL array, dimension (N)

BWORK (workspace) LOGICAL array, dimension (N)
Not referenced if SORT = 'N'.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, and i is
$\leq$ N: the QR algorithm failed to compute all the eigenvalues; elements 1:ilo−1 and i+1:n of WR and WI (*SGEESX*) or W (*CGEESX*) contain those eigenvalues which have converged; if JOBVS = 'V', VS contains the transformation which reduces A to its partially converged Schur form.
= N+1: the eigenvalues could not be reordered because some eigenvalues were too close to separate (the problem is very ill-conditioned);
= N+2: after reordering, roundoff changed values of some complex eigenvalues so that leading eigenvalues in the Schur form no longer satisfy SELECT=.TRUE. This could also be caused by underflow due to scaling.

SGEEV/CGEEV

```
      SUBROUTINE SGEEV( JOBVL, JOBVR, N, A, LDA, WR, WI, VL, LDVL, VR,
     $                  LDVR, WORK, LWORK, INFO )
      CHARACTER          JOBVL, JOBVR
      INTEGER            INFO, LDA, LDVL, LDVR, LWORK, N
      REAL               A( LDA, * ), VL( LDVL, * ), VR( LDVR, * ),
     $                   WI( * ), WORK( * ), WR( * )
```

```
      SUBROUTINE CGEEV( JOBVL, JOBVR, N, A, LDA, W, VL, LDVL, VR, LDVR,
     $                  WORK, LWORK, RWORK, INFO )
      CHARACTER          JOBVL, JOBVR
      INTEGER            INFO, LDA, LDVL, LDVR, LWORK, N
      REAL               RWORK( * )
      COMPLEX            A( LDA, * ), VL( LDVL, * ), VR( LDVR, * ),
     $                   W( * ), WORK( * )
```

Purpose

SGEEV/CGEEV computes for an n-by-n real/complex nonsymmetric matrix A, the eigenvalues and, optionally, the left and/or right eigenvectors.

The right eigenvector v(j) of A satisfies

$$A * v(j) \quad = \quad \lambda(j) * v(j)$$

where λ(j) is its eigenvalue.
The left eigenvector u(j) of A satisfies

$$u(j)^H * A \quad = \quad \lambda(j) * u(j)^H$$

where $u(j)^H$ denotes the conjugate transpose of u(j).

The computed eigenvectors are normalized to have Euclidean norm equal to 1 and largest component real.

Arguments

JOBVL (input) CHARACTER*1
= 'N': left eigenvectors of A are not computed;
= 'V': left eigenvectors of A are computed.

JOBVR (input) CHARACTER*1
= 'N': right eigenvectors of A are not computed;
= 'V': right eigenvectors of A are computed.

N (input) INTEGER
The order of the matrix A. N $\geq$ 0.

A (input/output) REAL/COMPLEX array, dimension (LDA,N)
On entry, the n-by-n matrix A.
On exit, A has been overwritten.

LDA (input) INTEGER
The leading dimension of the array A. LDA $\geq$ max(1,N).

WR, WI *SGEEV only* (output) REAL array, dimension (N)
WR and WI contain the real and imaginary parts, respectively, of the computed eigenvalues. Complex conjugate pairs of eigenvalues appear consecutively with the eigenvalue having the positive imaginary part first.

W *CGEEV only* (output) COMPLEX array, dimension (N)
W contains the computed eigenvalues.

VL (output) REAL/COMPLEX array, dimension (LDVL,N)
If JOBVL = 'V', the left eigenvectors u(j) are stored one after another in the columns of VL, in the same order as their eigenvalues.
If JOBVL = 'N', VL is not referenced.
SGEEV
If the jth eigenvalue is real, then u(j) = VL(:,j), the jth column of VL.
If the jth and $(j+1)^{th}$ eigenvalues form a complex conjugate pair, then u(j) = VL(:,j) + i*VL(:,j+1) and u(j+1) = VL(:,j) − i*VL(:,j+1).
CGEEV
u(j) = VL(:,j), the jth column of VL.

LDVL (input) INTEGER
The leading dimension of the array VL. LDVL ≥ 1; if JOBVL = 'V', LDVL ≥ N.

VR (output) REAL/COMPLEX array, dimension (LDVR,N)
If JOBVR = 'V', the right eigenvectors v(j) are stored one after another in the columns of VR, in the same order as their eigenvalues.
If JOBVR = 'N', VR is not referenced.
SGEEV
If the jth eigenvalue is real, then v(j) = VR(:,j), the jth column of VR.
If the jth and $(j+1)^{th}$ eigenvalues form a complex conjugate pair, then v(j) = VR(:,j) + i*VR(:,j+1) and v(j+1) = VR(:,j) − i*VR(:,j+1).
CGEEV
v(j) = VR(:,j), the jth column of VR.

LDVR (input) INTEGER
The leading dimension of the matrix VR. LDVR ≥ 1; if JOBVR = 'V', LDVR ≥ N.

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK.
LWORK ≥ max(1,3*N), and if JOBVL = 'V' or JOBVR = 'V', LWORK ≥ max(1,4*N) (*SGEEV*)
LWORK ≥ max(1,2*N) (*CGEEV*).
For good performance, LWORK must generally be larger.

If LWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK *CGEEV only* (workspace) REAL array, dimension (2*N)

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, the QR algorithm failed to compute all the eigenvalues, and no eigenvectors have been computed; elements i+1:n of WR and WI (*SGEEV*) or W (*CGEEV*) contain eigenvalues which have converged.

SGEEVX/CGEEVX

```
      SUBROUTINE SGEEVX( BALANC, JOBVL, JOBVR, SENSE, N, A, LDA, WR, WI,
     $                   VL, LDVL, VR, LDVR, ILO, IHI, SCALE, ABNRM,
     $                   RCONDE, RCONDV, WORK, LWORK, IWORK, INFO )
      CHARACTER          BALANC, JOBVL, JOBVR, SENSE
      INTEGER            IHI, ILO, INFO, LDA, LDVL, LDVR, LWORK, N
      REAL               ABNRM
      INTEGER            IWORK( * )
      REAL               A( LDA, * ), RCONDE( * ), RCONDV( * ),
     $                   SCALE( * ), VL( LDVL, * ), VR( LDVR, * ),
     $                   WI( * ), WORK( * ), WR( * )

      SUBROUTINE CGEEVX( BALANC, JOBVL, JOBVR, SENSE, N, A, LDA, W, VL,
     $                   LDVL, VR, LDVR, ILO, IHI, SCALE, ABNRM, RCONDE,
     $                   RCONDV, WORK, LWORK, RWORK, INFO )
      CHARACTER          BALANC, JOBVL, JOBVR, SENSE
      INTEGER            IHI, ILO, INFO, LDA, LDVL, LDVR, LWORK, N
      REAL               ABNRM
      REAL               RCONDE( * ), RCONDV( * ), RWORK( * ),
     $                   SCALE( * )
      COMPLEX            A( LDA, * ), VL( LDVL, * ), VR( LDVR, * ),
     $                   W( * ), WORK( * )
```

Purpose

SGEEVX/CGEEVX computes for an n-by-n real/complex nonsymmetric matrix A, the eigenvalues and, optionally, the left and/or right eigenvectors.

Optionally also, it computes a balancing transformation to improve the conditioning of the eigenvalues and eigenvectors (ILO, IHI, SCALE, and ABNRM), reciprocal condition numbers for the eigenvalues (RCONDE), and reciprocal condition numbers for the right eigenvectors (RCONDV).

The right eigenvector v(j) of A satisfies

$$A * v(j) \quad = \quad \lambda(j) * v(j)$$

where λ(j) is its eigenvalue.
The left eigenvector u(j) of A satisfies

$$u(j)^H * A \quad = \quad \lambda(j) * u(j)^H$$

where $u(j)^H$ denotes the conjugate transpose of u(j).

The computed eigenvectors are normalized to have Euclidean norm equal to 1 and largest component real.

Balancing a matrix means permuting the rows and columns to make it more nearly upper triangular, and applying a diagonal similarity transformation $D*A*D^{-1}$, where D is a diagonal matrix, to make its rows and columns closer in norm and the condition numbers of its eigenvalues and eigenvectors smaller. The computed reciprocal condition numbers correspond to the balanced matrix.

Permuting rows and columns will not change the condition numbers (in exact arithmetic) but diagonal scaling will. For further explanation of balancing, see section 4.8.1.2.

For further explanation of the reciprocal condition numbers RCONDE and RCONDV, see Section 4.8.1.3 (where these quantities are called *s* and sep respectively).

Arguments

BALANC (input) CHARACTER*1
Indicates how the input matrix should be diagonally scaled and/or permuted to improve the conditioning of its eigenvalues.
= 'N': Do not diagonally scale or permute;
= 'P': Perform permutations to make the matrix more nearly upper triangular. Do not diagonally scale;
= 'S': Diagonally scale the matrix, i.e/ie. replace A by $D*A*D^{-1}$, where D is a diagonal matrix chosen to make the rows and columns of A more equal in norm. Do not permute;
= 'B': Both diagonally scale and permute A.
Computed reciprocal condition numbers will be for the matrix after balancing and/or permuting. Permuting does not change condition numbers (in exact arithmetic), but balancing does.

JOBVL (input) CHARACTER*1
= 'N': left eigenvectors of A are not computed;
= 'V': left eigenvectors of A are computed.
If SENSE = 'E' or 'B', JOBVL must = 'V'.

JOBVR (input) CHARACTER*1
= 'N': right eigenvectors of A are not computed;
= 'V': right eigenvectors of A are computed.
If SENSE = 'E' or 'B', JOBVR must = 'V'.

SENSE (input) CHARACTER*1
Determines which reciprocal condition numbers are computed.
= 'N': None are computed;
= 'E': Computed for eigenvalues only;
= 'V': Computed for right eigenvectors only;
= 'B': Computed for eigenvalues and right eigenvectors.
If SENSE = 'E' or 'B', both left and right eigenvectors must also be computed (JOBVL = 'V' and JOBVR = 'V').

N (input) INTEGER
The order of the matrix A. $N \geq 0$.

A (input/output) REAL/COMPLEX array, dimension (LDA,N)
On entry, the n-by-n matrix A.
On exit, A has been overwritten. If JOBVL = 'V' or JOBVR = 'V', A contains the real-Schur/Schur form of the balanced version of the input matrix A.

LDA (input) INTEGER
The leading dimension of the array A. $LDA \geq \max(1,N)$.

WR, WI *SGEEVX only* (output) REAL array, dimension (N)
WR and WI contain the real and imaginary parts, respectively, of the computed eigenvalues. Complex conjugate pairs of eigenvalues appear consecutively with the eigenvalue having the positive imaginary part first.

W *CGEEVX only* (output) COMPLEX array, dimension (N)
W contains the computed eigenvalues.

VL (output) REAL/COMPLEX array, dimension (LDVL,N)
If JOBVL = 'V', the left eigenvectors u(j) are stored one after another in the columns of VL, in the same order as their eigenvalues.
If JOBVL = 'N', VL is not referenced.
SGEEV
If the jth eigenvalue is real, then u(j) = VL(:,j), the jth column of VL.
If the jth and $(j+1)^{th}$ eigenvalues form a complex conjugate pair, then u(j) = VL(:,j) + i*VL(:,j+1) and u(j+1) = VL(:,j) − i*VL(:,j+1).
CGEEV
u(j) = VL(:,j), the jth column of VL.

LDVL (input) INTEGER
The leading dimension of the array VL. $LDVL \geq 1$; if JOBVL = 'V', $LDVL \geq N$.

VR (output) REAL/COMPLEX array, dimension (LDVR,N)
If JOBVR = 'V', the right eigenvectors v(j) are stored one after another in the columns of VR, in the same order as their eigenvalues.
If JOBVR = 'N', VR is not referenced.
SGEEV
If the jth eigenvalue is real, then v(j) = VR(:,j), the jth column of VR.
If the jth and $(j+1)^{th}$ eigenvalues form a complex conjugate pair, then v(j) = VR(:,j) + i*VR(:,j+1) and v(j+1) = VR(:,j) − i*VR(:,j+1).
CGEEV
v(j) = VR(:,j), the jth column of VR.

LDVR (input) INTEGER
The leading dimension of the matrix VR. $LDVR \geq 1$; if JOBVR = 'V', $LDVR \geq N$.

ILO, IHI (output) INTEGER
ILO and IHI are integer values determined when A was balanced. The

balanced A(i,j) = 0 if i > j and j = 1,...,ilo−1 or i = ihi+1,...,n.
If BALANC = 'N' or 'S', ILO = 1 and IHI = N.

SCALE (output) REAL array, dimension (N)
Details of the permutations and scaling factors applied when balancing A. If P(j) is the index of the row and column interchanged with row and column j, and D(j) is the scaling factor applied to row and column j, then

```
SCALE(j)  = P(j)   for j = 1,...,ilo-1
          = D(j)   for j = ilo,...,ihi
          = P(j)   for j = ihi+1,...,n.
```

The order in which the interchanges are made is n to ihi+1, then 1 to ilo−1.

ABNRM (output) REAL
The one-norm of the balanced matrix (the maximum of the sum of absolute values of elements of any column).

RCONDE (output) REAL array, dimension (N)
RCONDE(j) is the reciprocal condition number of the jth eigenvalue.

RCONDV (output) REAL array, dimension (N)
RCONDV(j) is the reciprocal condition number of the jth right eigenvector.

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK.
SGEEVX
If SENSE = 'N' or 'E', LWORK ≥ max(1,2*N), and
if JOBVL = 'V' or JOBVR = 'V', LWORK ≥ 3*N.
If SENSE = 'V' or 'B', LWORK ≥ N*(N+6).
For good performance, LWORK must generally be larger.
CGEEVX
If SENSE = 'N' or 'E', LWORK ≥ max(1,2*N), and
if SENSE = 'V' or 'B', LWORK ≥ N*N+2*N.
For good performance, LWORK must generally be larger.

If LWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK *CGEEVX only* (workspace) REAL array, dimension (2*N)

IWORK *SGEEVX only* (workspace) INTEGER array, dimension (2*N−2)
If SENSE = 'N' or 'E', not referenced.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, the QR algorithm failed to compute all the eigenvalues, and no eigenvectors or condition numbers have been computed; elements 1:ilo−1 and i+1:n of WR and WI (*SGEEVX*) or W (*CGEEVX*) contain eigenvalues which have converged.

SGEHRD/CGEHRD

```
SUBROUTINE SGEHRD( N, ILO, IHI, A, LDA, TAU, WORK, LWORK, INFO )
INTEGER            IHI, ILO, INFO, LDA, LWORK, N
REAL               A( LDA, * ), TAU( * ), WORK( LWORK )

SUBROUTINE CGEHRD( N, ILO, IHI, A, LDA, TAU, WORK, LWORK, INFO )
INTEGER            IHI, ILO, INFO, LDA, LWORK, N
COMPLEX            A( LDA, * ), TAU( * ), WORK( LWORK )
```

Purpose

SGEHRD/CGEHRD reduces a real/complex general matrix A to upper Hessenberg form H by an orthogonal/unitary similarity transformation: $Q^H*A*Q = H$.

Arguments

N (input) INTEGER
The order of the matrix A. N ≥ 0.

ILO, IHI (input) INTEGER
It is assumed that A is already upper triangular in rows and columns 1:ilo−1 and ihi+1:n. ILO and IHI are normally set by a previous call to SGEBAL/CGEBAL; otherwise they should be set to 1 and N respectively.
1 ≤ ILO ≤ IHI ≤ N, if N > 0; ILO = 1 and IHI = 0, if N = 0.

A (input/output) REAL/COMPLEX array, dimension (LDA,N)
On entry, the n-by-n general matrix to be reduced.
On exit, the upper triangle and the first subdiagonal of A are overwritten with the upper Hessenberg matrix H, and the elements below the first subdiagonal, with the array TAU, represent the orthogonal/unitary matrix Q as a product of elementary reflectors.

LDA (input) INTEGER
The leading dimension of the array A. LDA ≥ max(1,N).

TAU (output) REAL/COMPLEX array, dimension (N−1)
The scalar factors of the elementary reflectors. Elements 1:ilo−1 and ihi:n−1 of TAU are set to zero.

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The length of the array WORK. LWORK ≥ max(1,N). For optimum

performance LWORK $\geq$ N*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = $-$i, the ith argument had an illegal value.

SGELQF/CGELQF

```
SUBROUTINE SGELQF( M, N, A, LDA, TAU, WORK, LWORK, INFO )
INTEGER            INFO, LDA, LWORK, M, N
REAL               A( LDA, * ), TAU( * ), WORK( LWORK )

SUBROUTINE CGELQF( M, N, A, LDA, TAU, WORK, LWORK, INFO )
INTEGER            INFO, LDA, LWORK, M, N
COMPLEX            A( LDA, * ), TAU( * ), WORK( LWORK )
```

Purpose

SGELQF/CGELQF computes an LQ factorization of a real/complex m-by-n matrix A: A = L*Q.

Arguments

M (input) INTEGER
The number of rows of the matrix A. M $\geq$ 0.

N (input) INTEGER
The number of columns of the matrix A. N $\geq$ 0.

A (input/output) REAL/COMPLEX array, dimension (LDA,N)
On entry, the m-by-n matrix A.
On exit, the elements on and below the diagonal of the array contain the m-by-min(m,n) lower trapezoidal matrix L (L is lower triangular if m $\leq$ n); the elements above the diagonal, with the array TAU, represent the orthogonal/unitary matrix Q as a product of elementary reflectors.

LDA (input) INTEGER
The leading dimension of the array A. LDA $\geq$ max(1,M).

TAU (output) REAL/COMPLEX array, dimension (min(M,N))
The scalar factors of the elementary reflectors.

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK. LWORK $\geq$ max(1,M). For optimum performance LWORK $\geq$ M*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = $-$i, the ith argument had an illegal value.

SGELS/CGELS

```
SUBROUTINE SGELS( TRANS, M, N, NRHS, A, LDA, B, LDB, WORK, LWORK,
$                 INFO )
CHARACTER          TRANS
INTEGER            INFO, LDA, LDB, LWORK, M, N, NRHS
REAL               A( LDA, * ), B( LDB, * ), WORK( LWORK )

SUBROUTINE CGELS( TRANS, M, N, NRHS, A, LDA, B, LDB, WORK, LWORK,
$                 INFO )
CHARACTER          TRANS
INTEGER            INFO, LDA, LDB, LWORK, M, N, NRHS
COMPLEX            A( LDA, * ), B( LDB, * ), WORK( LWORK )
```

Purpose

SGELS/CGELS solves overdetermined or underdetermined real/complex linear systems involving an m-by-n matrix A, or its transpose/conjugate-transpose, using a QR or LQ factorization of A. It is assumed that A has full rank.

The following options are provided:

1. If TRANS = 'N' and m $\geq$ n: find the least squares solution of an overdetermined system, i.e., solve the least squares problem

$$\text{minimize} \| b - A*x \|_2.$$

2. If TRANS = 'N' and m < n: find the minimum norm solution of an underdetermined system A*x = B.

3. If TRANS = 'T'/'C' and m $\geq$ n: find the minimum norm solution of an undetermined system A^H*X = B.

4. If TRANS = 'T'/'C' and m < n: find the least squares solution of an overdetermined system, i.e., solve the least squares problem

$$\text{minimize} \| b - A^H*x \|_2.$$

Several right hand side vectors b and solution vectors x can be handled in a single

call; they are stored as the columns of the m-by-nrhs right hand side matrix B and the n-by-nrhs solution matrix X.

Arguments

TRANS (input) CHARACTER
= 'N': the linear system involves A;
= 'T': the linear system involves A^T (*SGELS*);
= 'C': the linear system involves A^H (*CGELS*).

M (input) INTEGER
The number of rows of the matrix A. $M \geq 0$.

N (input) INTEGER
The number of columns of the matrix A. $N \geq 0$.

NRHS (input) INTEGER
The number of right hand sides, i.e., the number of columns of the matrices B and X. $NRHS \geq 0$.

A (input/output) REAL/COMPLEX array, dimension (LDA,N)
On entry, the m-by-n matrix A.
On exit,
if $m \geq n$, A is overwritten by details of its QR factorization as returned by SGEQRF/CGEQRF;
if $m < n$, A is overwritten by details of its LQ factorization as returned by SGELQF/CGELQF.

LDA (input) INTEGER
The leading dimension of the array A. $LDA \geq \max(1,M)$.

B (input/output) REAL/COMPLEX array, dimension (LDB,NRHS)
On entry, the matrix B of right hand side vectors, stored columnwise; B is m-by-nrhs if TRANS = 'N', or n-by-nrhs if TRANS = 'T' or 'C'.
On exit, B is overwritten by the solution vectors, stored columnwise:
if TRANS = 'N' and $m \geq n$, rows 1 to n of B contain the least squares solution vectors; the residual sum of squares for the solution in each column is given by the sum of squares of elements n+1 to m in that column;
if TRANS = 'N' and $m < n$, rows 1 to n of B contain the minimum norm solution vectors;
if TRANS = 'T' or 'C' and $m \geq n$, rows 1 to m of B contain the minimum norm solution vectors;
if TRANS = 'T' or 'C' and $m < n$, rows 1 to m of B contain the least squares solution vectors; the residual sum of squares for the solution in each column is given by the sum of squares of elements m+1 to n in that column.

LDB (input) INTEGER
The leading dimension of the array B. $LDB \geq \max(1,M,N)$.

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK. $LWORK \geq \min(M,N) + \max(1,M,N,NRHS)$. For optimal performance, $LWORK \geq \min(M,N) + \max(1,M,N,NRHS)*NB$ where NB is the optimum block size.

If LWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SGELSD/CGELSD

```
      SUBROUTINE SGELSD( M, N, NRHS, A, LDA, B, LDB, S, RCOND, RANK,
     $                   WORK, LWORK, IWORK, INFO )
      INTEGER            INFO, LDA, LDB, LWORK, M, N, NRHS, RANK
      REAL               RCOND
      INTEGER            IWORK( * )
      REAL               A( LDA, * ), B( LDB, * ), S( * ), WORK( * )

      SUBROUTINE CGELSD( M, N, NRHS, A, LDA, B, LDB, S, RCOND, RANK,
     $                   WORK, LWORK, RWORK, IWORK, INFO )
      INTEGER            INFO, LDA, LDB, LWORK, M, N, NRHS, RANK
      REAL               RCOND
      INTEGER            IWORK( * )
      REAL               RWORK( * ), S( * )
      COMPLEX            A( LDA, * ), B( LDB, * ), WORK( * )
```

Purpose

SGELSD/CGELSD computes the minimum norm solution to a real/complex linear least squares problem:

$$\text{minimize} \| b - A*x \|_2.$$

using the singular value decomposition (SVD) of A. A is an m-by-n matrix which may be rank-deficient.

Several right hand side vectors b and solution vectors x can be handled in a single call; they are stored as the columns of the m-by-nrhs right hand side matrix B and the n-by-nrhs solution matrix X.

The problem is solved in three steps:

1. Reduce the coefficient matrix A to bidiagonal form with Householder tranformations, reducing the original problem into a "bidiagonal least squares

problem" (BLS).

2. Solve the BLS using a divide and conquer approach.

3. Apply back all the Householder tranformations to solve the original least squares problem.

The effective rank of A is determined by treating as zero those singular values which are less than RCOND times the largest singular value.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Arguments

M (input) INTEGER
The number of rows of the matrix A. M $\geq$ 0.

N (input) INTEGER
The number of columns of the matrix A. N $\geq$ 0.

NRHS (input) INTEGER
The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS $\geq$ 0.

A (input/output) REAL/COMPLEX array, dimension (LDA,N)
On entry, the m-by-n matrix A.
On exit, the first min(m,n) rows of A are overwritten with its right singular vectors, stored rowwise.

LDA (input) INTEGER
The leading dimension of the array A. LDA $\geq$ max(1,M).

B (input/output) REAL/COMPLEX array, dimension (LDB,NRHS)
On entry, the m-by-nrhs right hand side matrix B.
On exit, B is overwritten by the n-by-nrhs solution matrix X.
If m $\geq$ n and RANK = n, the residual sum-of-squares for the solution in the ith column is given by the sum of squares of elements n+1:m in that column.

LDB (input) INTEGER
The leading dimension of the array B. LDB $\geq$ max(1,M,N).

S (output) REAL array, dimension (min(M,N))
The singular values of A in decreasing order.
The condition number of A in the 2-norm = $S(1)/S(min(m,n))$.

RCOND (input) REAL
RCOND is used to determine the effective rank of A. Singular values $S(i) \leq RCOND * S(1)$ are treated as zero. If RCOND < 0, machine precision is used instead.

RANK (output) INTEGER
The effective rank of A, i.e., the number of singular values which are greater than $RCOND * S(1)$.

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK. LWORK $\geq$ 1. If M $\geq$ N, LWORK $\geq 11*N+2*N*SMLSIZ+8*N*NLVL+N*NRHS$. If M < N, LWORK $\geq 11*M+2*M*SMLSIZ+8*M*NLVL+M*NRHS$. SMLSIZ is returned by ILAENV and is equal to the maximum size of the subproblems at the bottom of the computation tree (usually about 25), and $NLVL = INT(LOG_2(MIN(M,N)/(SMLSIZ+1)))+1$ For good performance, LWORK should generally be larger.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

IWORK (workspace) INTEGER array, dimension (LIWORK)
LIWORK $\geq 3*MINMN*NLVL+11*MINMN$, where MINMN = MIN(M,N).

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = $-$i, the ith argument had an illegal value.
> 0: the algorithm for computing the SVD failed to converge; if INFO = i, i off-diagonal elements of an intermediate bidiagonal form did not converge to zero.

SGELSS/CGELSS

```
      SUBROUTINE SGELSS( M, N, NRHS, A, LDA, B, LDB, S, RCOND, RANK,
     $                   WORK, LWORK, INFO )
      INTEGER            INFO, LDA, LDB, LWORK, M, N, NRHS, RANK
      REAL               RCOND
      REAL               A( LDA, * ), B( LDB, * ), S( * ), WORK( * )

      SUBROUTINE CGELSS( M, N, NRHS, A, LDA, B, LDB, S, RCOND, RANK,
     $                   WORK, LWORK, RWORK, INFO )
      INTEGER            INFO, LDA, LDB, LWORK, M, N, NRHS, RANK
      REAL               RCOND
      REAL               RWORK( * ), S( * )
      COMPLEX            A( LDA, * ), B( LDB, * ), WORK( * )
```

Purpose

SGELSS/CGELSS computes the minimum norm solution to a real/complex linear least squares problem:

$$\text{minimize} \| b - A*x \|_2.$$

using the singular value decomposition (SVD) of A. A is an m-by-n matrix which may be rank-deficient.

Several right hand side vectors b and solution vectors x can be handled in a single call; they are stored as the columns of the m-by-nrhs right hand side matrix B and the n-by-nrhs solution matrix X.

The effective rank of A is determined by treating as zero those singular values which are less than RCOND times the largest singular value.

Arguments

M (input) INTEGER
The number of rows of the matrix A. $M \geq 0$.

N (input) INTEGER
The number of columns of the matrix A. $N \geq 0$.

NRHS (input) INTEGER
The number of right hand sides, i.e., the number of columns of the matrices B and X. $NRHS \geq 0$.

A (input/output) REAL/COMPLEX array, dimension (LDA,N)
On entry, the m-by-n matrix A.
On exit, the first min(m,n) rows of A are overwritten with its right singular vectors, stored rowwise.

LDA (input) INTEGER
The leading dimension of the array A. $LDA \geq \max(1,M)$.

B (input/output) REAL/COMPLEX array, dimension (LDB,NRHS)
On entry, the m-by-nrhs right hand side matrix B.
On exit, B is overwritten by the n-by-nrhs solution matrix X.
If $m \geq n$ and $RANK = n$, the residual sum-of-squares for the solution in the ith column is given by the sum of squares of elements n+1:m in that column.

LDB (input) INTEGER
The leading dimension of the array B. $LDB \geq \max(1,M,N)$.

S (output) REAL array, dimension (min(M,N))
The singular values of A in decreasing order. The condition number of A in the 2-norm is $\kappa_2(A) = S(1)/S(\min(m,n))$.

RCOND (input) REAL
RCOND is used to determine the effective rank of A.
Singular values $S(i) \leq RCOND*S(1)$ are treated as zero.
If $RCOND < 0$, machine precision is used instead.

RANK (output) INTEGER
The effective rank of A, i.e., the number of singular values which are greater than RCOND*S(1).

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if $INFO = 0$, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK. $LWORK \geq 1$, and also:
SGELSS
$LWORK \geq 3*\min(M,N)+\max(2*\min(M,N), \max(M,N), NRHS)$
CGELSS
$LWORK \geq 2*\min(M,N)+\max(M,N,NRHS)$
For good performance, LWORK should generally be larger.

If $LWORK = -1$, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK *CGELSS only* (workspace) REAL array,
dimension (max(1,5*min(M,N)))

INFO (output) INTEGER
$= 0$: successful exit
< 0: if $INFO = -i$, the ith argument had an illegal value.
> 0: the algorithm for computing the SVD failed to converge; if $INFO = i$, i off-diagonal elements of an intermediate bidiagonal form did not converge to zero.

SGELSY/CGELSY

```
      SUBROUTINE SGELSY( M, N, NRHS, A, LDA, B, LDB, JPVT, RCOND, RANK,
     $                   WORK, LWORK, INFO )
      INTEGER            INFO, LDA, LDB, LWORK, M, N, NRHS, RANK
      REAL               RCOND
      INTEGER            JPVT( * )
      REAL               A( LDA, * ), B( LDB, * ), WORK( LWORK )

      SUBROUTINE CGELSY( M, N, NRHS, A, LDA, B, LDB, JPVT, RCOND, RANK,
     $                   WORK, LWORK, RWORK, INFO )
      INTEGER            INFO, LDA, LDB, LWORK, M, N, NRHS, RANK
      REAL               RCOND
      INTEGER            JPVT( * )
      REAL               RWORK( * )
      COMPLEX            A( LDA, * ), B( LDB, * ), WORK( * )
```

Purpose

SGELSY/CGELSY computes the minimum-norm solution to a real/complex linear least squares problem:

minimize || b − A*x ||$_2$

using a complete orthogonal factorization of A. A is an m-by-n matrix which may be rank-deficient.

Several right hand side vectors b and solution vectors x can be handled in a single call; they are stored as the columns of the m-by-nrhs right hand side matrix B and the n-by-nrhs solution matrix X.

The routine first computes a QR factorization with column pivoting:

$$A * P = Q * \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix}$$

with R_{11} defined as the largest leading submatrix whose estimated condition number is less than 1/RCOND. The order of R_{11}, RANK, is the effective rank of A.

Then, R_{22} is considered to be negligible, and R_{12} is annihilated by orthogonal/unitary transformations from the right, arriving at the complete orthogonal factorization:

$$A * P = Q * \begin{pmatrix} T_{11} & 0 \\ 0 & 0 \end{pmatrix} * Z$$

The minimum norm solution is then

$$x = P * Z^H * \begin{pmatrix} T_{11}^{-1} * Q_1^H * b \\ 0 \end{pmatrix}$$

where Q_1 consists of the first RANK columns of Q.

This routine is basically identical to the original xGELSX except three differences:

- The call to the subroutine xGEQPF has been substituted by the the call to the subroutine xGEQP3. This subroutine is a Blas-3 version of the QR factorization with column pivoting.
- Matrix B (the right hand side) is updated with Blas-3.
- The permutation of matrix B (the right hand side) is faster and more simple.

Arguments

M (input) INTEGER
The number of rows of the matrix A. M ≥ 0.

N (input) INTEGER
The number of columns of the matrix A. N ≥ 0.

NRHS (input) INTEGER
The number of right hand sides, i.e., the number of columns of matrices B and X. NRHS ≥ 0.

A (input/output) REAL/COMPLEX array, dimension (LDA,N)
On entry, the m-by-n matrix A.
On exit, A has been overwritten by details of its complete orthogonal factorization.

LDA (input) INTEGER
The leading dimension of the array A. LDA ≥ max(1,M).

B (input/output) REAL/COMPLEX array, dimension (LDB,NRHS)
On entry, the m-by-nrhs right hand side matrix B.
On exit, the n-by-nrhs solution matrix X.

LDB (input) INTEGER
The leading dimension of the array B. LDB ≥ max(1,M,N).

JPVT (input/output) INTEGER array, dimension (N)
On entry, if JPVT(i) ≠ 0, the ith column of A is permuted to the front of AP, otherwise column i is a free column.
On exit, if JPVT(i) = k, then the ith column of AP was the kth column of A.

RCOND (input) REAL
RCOND is used to determine the effective rank of A, which is defined as the order of the largest leading triangular submatrix R_{11} in the QR factorization with pivoting of A, whose estimated condition number < 1/RCOND.

RANK (output) INTEGER
The effective rank of A, i.e., the order of the submatrix R_{11}. This is the same as the order of the submatrix T_{11} in the complete orthogonal factorization of A.

WORK (workspace) REAL/COMPLEX array, dimension (LWORK)

LWORK INTEGER
The dimension of the array WORK.
SGELSY
The unblocked strategy requires that:
LWORK ≥ MAX(MN+3*N+1, 2*MN+NRHS),
where MN = min(M,N). The block algorithm requires that:
LWORK ≥ MAX(MN+2*N+NB*(N+1), 2*MN+NB*NRHS), where NB is an upper bound on the blocksize returned by ILAENV for the routines SGEQP3, STZRZF, STZRQF, SORMQR, and SORMRZ.
CGELSY
The unblocked strategy requires that:
LWORK ≥ MN + MAX(2*MN, N+1, MN+NRHS)
where MN = min(M,N). The block algorithm requires that:

LWORK $\geq$ MN + MAX(2*MN, NB*(N+1), MN+MN*NB, MN+NB*NRHS)
where NB is an upper bound on the blocksize returned by ILAENV for the routines CGEQP3, CTZRZF, CTZRQF, CUNMQR, and CUNMRZ.

RWORK *CGELSX only* (workspace) REAL array, dimension (2*N)

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SGEQLF/CGEQLF

```
SUBROUTINE SGEQLF( M, N, A, LDA, TAU, WORK, LWORK, INFO )
INTEGER            INFO, LDA, LWORK, M, N
REAL               A( LDA, * ), TAU( * ), WORK( LWORK )

SUBROUTINE CGEQLF( M, N, A, LDA, TAU, WORK, LWORK, INFO )
INTEGER            INFO, LDA, LWORK, M, N
COMPLEX            A( LDA, * ), TAU( * ), WORK( LWORK )
```

Purpose

SGEQLF/CGEQLF computes a QL factorization of a real/complex m-by-n matrix A: A = Q*L.

Arguments

M (input) INTEGER
The number of rows of the matrix A. M $\geq$ 0.

N (input) INTEGER
The number of columns of the matrix A. N $\geq$ 0.

A (input/output) REAL/COMPLEX array, dimension (LDA,N)
On entry, the m-by-n matrix A.
On exit,
if m $\geq$ n, the lower triangle of the subarray A(m-n+1:m,1:n) contains the n-by-n lower triangular matrix L;
if m $\leq$ n, the elements on and below the $(n-m)^{th}$ superdiagonal contain the m-by-n lower trapezoidal matrix L;
the remaining elements, with the array TAU, represent the orthogonal/unitary matrix Q as a product of elementary reflectors.

LDA (input) INTEGER
The leading dimension of the array A. LDA $\geq$ max(1,M).

TAU (output) REAL/COMPLEX array, dimension (min(M,N))
The scalar factors of the elementary reflectors.

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK. LWORK $\geq$ max(1,N). For optimum performance LWORK $\geq$ N*NB, where NB is the optimal blocksize.

If LWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SGEQP3/CGEQP3

```
SUBROUTINE SGEQP3( M, N, A, LDA, JPVT, TAU, WORK, LWORK, INFO )
INTEGER            INFO, LDA, LWORK, M, N
INTEGER            JPVT( * )
REAL               A( LDA, * ), TAU( * ), WORK( * )

SUBROUTINE CGEQP3( M, N, A, LDA, JPVT, TAU, WORK, LWORK, RWORK,
$                  INFO )
INTEGER            INFO, LDA, LWORK, M, N
INTEGER            JPVT( * )
REAL               RWORK( * )
COMPLEX            A( LDA, * ), TAU( * ), WORK( * )
```

Purpose

SGEQP3/CGEQP3 computes a QR factorization with column pivoting of a real/-complex m-by-n matrix A: A*P = Q*R using Level 3 BLAS.

Arguments

M (input) INTEGER
The number of rows of the matrix A. M $\geq$ 0.

N (input) INTEGER
The number of columns of the matrix A. N $\geq$ 0

A (input/output) REAL/COMPLEX array, dimension (LDA,N)
On entry, the m-by-n matrix A.
On exit, the upper triangle of the array contains the min(m,n)-by-n upper trapezoidal matrix R; the elements below the diagonal, together with the array TAU, represent the orthogonal/unitary matrix Q as a product of min(m,n) elementary reflectors.

LDA (input) INTEGER
The leading dimension of the array A. LDA $\geq$ max(1,M).

JPVT (input/output) INTEGER array, dimension (N)
On entry, if JPVT(i) $\neq 0$, the ith column of A is permuted to the front of A*P (a leading column); if JPVT(i) = 0, the ith column of A is a free column.
On exit, if JPVT(i) = k, then the ith column of A*P was the kth column of A.

TAU (output) REAL/COMPLEX array, dimension (min(M,N))
The scalar factors of the elementary reflectors.

WORK *SGEQP3* (workspace) REAL array, dimension (3*N+1)
CGEQP3 (workspace) COMPLEX array, dimension (N+1)

RWORK *CGEQP3 only* (workspace) REAL array, dimension (2*N)

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SGEQRF/CGEQRF

```
      SUBROUTINE SGEQRF( M, N, A, LDA, TAU, WORK, LWORK, INFO )
      INTEGER            INFO, LDA, LWORK, M, N
      REAL               A( LDA, * ), TAU( * ), WORK( LWORK )

      SUBROUTINE CGEQRF( M, N, A, LDA, TAU, WORK, LWORK, INFO )
      INTEGER            INFO, LDA, LWORK, M, N
      COMPLEX            A( LDA, * ), TAU( * ), WORK( LWORK )
```

Purpose

SGEQRF/CGEQRF computes a QR factorization of a real/complex m-by-n matrix A: A = Q*R.

Arguments

M (input) INTEGER
The number of rows of the matrix A. M $\geq$ 0.

N (input) INTEGER
The number of columns of the matrix A. N $\geq$ 0.

A (input/output) REAL/COMPLEX array, dimension (LDA,N)
On entry, the m-by-n matrix A.
On exit, the elements on and above the diagonal of the array contain the min(m,n)-by-n upper trapezoidal matrix R (R is upper triangular if m $\geq$ n); the elements below the diagonal, with the array TAU, represent the orthogonal/unitary matrix Q as a product of min(m,n) elementary reflectors.

LDA (input) INTEGER
The leading dimension of the array A. LDA $\geq$ max(1,M).

TAU (output) REAL/COMPLEX array, dimension (min(M,N))
The scalar factors of the elementary reflectors.

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK. LWORK $\geq$ max(1,N). For optimum performance LWORK $\geq$ N*NB, where NB is the optimal blocksize.

If LWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SGERFS/CGERFS

```
      SUBROUTINE SGERFS( TRANS, N, NRHS, A, LDA, AF, LDAF, IPIV, B, LDB,
     $                   X, LDX, FERR, BERR, WORK, IWORK, INFO )
      CHARACTER          TRANS
      INTEGER            INFO, LDA, LDAF, LDB, LDX, N, NRHS
      INTEGER            IPIV( * ), IWORK( * )
      REAL               A( LDA, * ), AF( LDAF, * ), B( LDB, * ),
     $                   BERR( * ), FERR( * ), WORK( * ), X( LDX, * )

      SUBROUTINE CGERFS( TRANS, N, NRHS, A, LDA, AF, LDAF, IPIV, B, LDB,
     $                   X, LDX, FERR, BERR, WORK, RWORK, INFO )
      CHARACTER          TRANS
      INTEGER            INFO, LDA, LDAF, LDB, LDX, N, NRHS
      INTEGER            IPIV( * )
      REAL               BERR( * ), FERR( * ), RWORK( * )
      COMPLEX            A( LDA, * ), AF( LDAF, * ), B( LDB, * ),
     $                   WORK( * ), X( LDX, * )
```

Purpose

SGERFS/CGERFS improves the computed solution to a system of linear equations and provides error bounds and backward error estimates for the solution.

Arguments

TRANS (input) CHARACTER*1
Specifies the form of the system of equations:
= 'N': A*X = B (No transpose)
= 'T': A^T*X = B (Transpose)

= 'C': $A^H*X = B$ (Conjugate transpose)

N (input) INTEGER
The order of the matrix A. $N \geq 0$.

NRHS (input) INTEGER
The number of right hand sides, i.e., the number of columns of the matrices B and X. $NRHS \geq 0$.

A (input) REAL/COMPLEX array, dimension (LDA,N)
The original n-by-n matrix A.

LDA (input) INTEGER
The leading dimension of the array A. $LDA \geq \max(1,N)$.

AF (input) REAL/COMPLEX array, dimension (LDAF,N)
The factors L and U from the factorization A = P*L*U as computed by SGETRF/CGETRF.

LDAF (input) INTEGER
The leading dimension of the array AF. $LDAF \geq \max(1,N)$.

IPIV (input) INTEGER array, dimension (N)
The pivot indices from SGETRF/CGETRF; for $1 \leq i \leq N$, row i of the matrix was interchanged with row IPIV(i).

B (input) REAL/COMPLEX array, dimension (LDB,NRHS)
The right hand side matrix B.

LDB (input) INTEGER
The leading dimension of the array B. $LDB \geq \max(1,N)$.

X (input/output) REAL/COMPLEX array, dimension (LDX,NRHS)
On entry, the solution matrix X, as computed by SGETRS/CGETRS.
On exit, the improved solution matrix X.

LDX (input) INTEGER
The leading dimension of the array X. $LDX \geq \max(1,N)$.

FERR (output) REAL array, dimension (NRHS)
The estimated forward error bound for each solution vector X(j) (the jth column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) − XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR (output) REAL array, dimension (NRHS)
The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK *SGERFS* (workspace) REAL array, dimension (3*N)
CGERFS (workspace) COMPLEX array, dimension (2*N)

IWORK *SGERFS only* (workspace) INTEGER array, dimension (N)

RWORK *CGERFS only* (workspace) REAL array, dimension (N)

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SGERQF/CGERQF

```
SUBROUTINE SGERQF( M, N, A, LDA, TAU, WORK, LWORK, INFO )
INTEGER            INFO, LDA, LWORK, M, N
REAL               A( LDA, * ), TAU( * ), WORK( LWORK )

SUBROUTINE CGERQF( M, N, A, LDA, TAU, WORK, LWORK, INFO )
INTEGER            INFO, LDA, LWORK, M, N
COMPLEX            A( LDA, * ), TAU( * ), WORK( LWORK )
```

Purpose

SGERQF/CGERQF computes an RQ factorization of a real/complex m-by-n matrix A: A = R*Q.

Arguments

M (input) INTEGER
The number of rows of the matrix A. $M \geq 0$.

N (input) INTEGER
The number of columns of the matrix A. $N \geq 0$.

A (input/output) REAL/COMPLEX array, dimension (LDA,N)
On entry, the m-by-n matrix A.
On exit,
if $m \leq n$, the upper triangle of the subarray A(1:m,n-m+1:n) contains the m-by-m upper triangular matrix R;
if $m \geq n$, the elements on and above the $(m-n)^{th}$ subdiagonal contain the m-by-n upper trapezoidal matrix R;
the remaining elements, with the array TAU, represent the orthogonal/unitary matrix Q as a product of min(m,n) elementary reflectors.

LDA (input) INTEGER
The leading dimension of the array A. $LDA \geq \max(1,M)$.

TAU (output) REAL/COMPLEX array, dimension (min(M,N))
The scalar factors of the elementary reflectors.

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK. $LWORK \geq \max(1,M)$. For optimum performance $LWORK \geq M*NB$, where NB is the optimal blocksize.

If LWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SGESDD/CGESDD

```
      SUBROUTINE SGESDD( JOBZ, M, N, A, LDA, S, U, LDU, VT, LDVT, WORK,
     $                   LWORK, IWORK, INFO )
      CHARACTER          JOBZ
      INTEGER            INFO, LDA, LDU, LDVT, LWORK, M, N
      INTEGER            IWORK( * )
      REAL               A( LDA, * ), S( * ), U( LDU, * ),
     $                   VT( LDVT, * ), WORK( * )

      SUBROUTINE CGESDD( JOBZ, M, N, A, LDA, S, U, LDU, VT, LDVT, WORK,
     $                   LWORK, RWORK, IWORK, INFO )
      CHARACTER          JOBZ
      INTEGER            INFO, LDA, LDU, LDVT, LWORK, M, N
      INTEGER            IWORK( * )
      REAL               RWORK( * ), S( * )
      COMPLEX            A( LDA, * ), U( LDU, * ), VT( LDVT, * ),
     $                   WORK( * )
```

Purpose

SGESDD/CGESDD computes the singular value decomposition (SVD) of a real/complex m-by-n matrix A, optionally computing the left and right singular vectors. If singular vectors are desired, it uses a divide-and-conquer algorithm.

The SVD is written

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

where Σ is an m-by-n matrix which is zero except for its min(m,n) diagonal elements, U is an m-by-m orthogonal/unitary matrix, and V is an n-by-n orthogonal/unitary matrix. The diagonal elements of Σ are the singular values of A; they are real and non-negative, and are returned in descending order. The first min(m,n) columns of U and V are the left and right singular vectors of A.

Note that the routine returns V^H, not V.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Arguments

JOBZ (input) CHARACTER*1
Specifies options for computing all or part of the matrix U:
= 'A': all M columns of U and all N rows of V^T are returned in the arrays U and VT;
= 'S': the first min(M,N) columns of U and the first min(M,N) rows of V^T are returned in the arrays U and VT;
= 'O': If M ≥ N, the first N columns of U are overwritten on the array A and all rows of V^T are returned in the array VT;
otherwise, all columns of U are returned in the array U and the first M rows of V^T are overwritten in the array VT;
= 'N': no columns of U or rows of V^T are computed.

M (input) INTEGER
The number of rows of the matrix A. M ≥ 0.

N (input) INTEGER
The number of columns of the matrix A. N ≥ 0.

A (input/output) REAL/COMPLEX array, dimension (LDA,N)
On entry, the m-by-n matrix A.
On exit,
if JOBZ = 'O', A is overwritten with the first N columns of U (the left singular vectors, stored columnwise) if M ≥ N;
A is overwritten with the first M rows of V^T (the right singular vectors, stored rowwise) otherwise.
if JOBZ ≠ 'O', the contents of A are destroyed.

LDA (input) INTEGER
The leading dimension of the array A. LDA ≥ max(1,M).

S (output) REAL array, dimension (min(M,N))
The singular values of A, sorted so that S(i) ≥ S(i+1).

U (output) REAL/COMPLEX array, dimension (LDU,UCOL)
UCOL = M if JOBZ = 'A' or JOBZ = 'O' and M < N;
UCOL = min(M,N) if JOBZ = 'S'.
If JOBZ = 'A' or JOBZ = 'O' and M < N, U contains the M-by-M orthogonal/unitary matrix U;
if JOBZ = 'S', U contains the first min(M,N) columns of U (the left singular vectors, stored columnwise);
if JOBZ = 'O' and M ≥ N, or JOBZ = 'N', U is not referenced.

LDU (input) INTEGER
The leading dimension of the array U. LDU ≥ 1; if JOBZ = 'S' or 'A' or JOBZ = 'O' and M < N, LDU ≥ M.

VT (output) REAL/COMPLEX array, dimension (LDVT,N)
If JOBZ = 'A' or JOBZ = 'O' and M ≥ N, VT contains the N-by-N

unitary matrix V^T;
if JOBZ = 'S', VT contains the first min(M,N) rows of V^T (the right singular vectors, stored rowwise);
if JOBZ = 'O' and M < N, or JOBZ = 'N', VT is not referenced.

LDVT (input) INTEGER
The leading dimension of the array VT. LDVT $\geq$ 1; if JOBZ = 'A' or JOBZ = 'O' and M $\geq$ N, LDVT $\geq$ N; if JOBZ = 'S', LDVT $\geq$ min(M,N).

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK. LWORK $\geq$ 1.
SGESDD
If JOBZ = 'N', LWORK $\geq$ 5*min(M,N).
If JOBZ = 'O',
LWORK $\geq$ 5*min(M,N)*min(M,N) + max(M,N) + 9*min(M,N).
If JOBZ = 'S' or 'A'
LWORK $\geq$ 4*min(M,N)*min(M,N) + max(M,N) + 9*min(M,N).
CGESDD
If JOBZ = 'N', LWORK $\geq$ 2*min(M,N)+max(M,N).
If JOBZ = 'O',
LWORK $\geq$ 2*min(M,N)*min(M,N)+2*min(M,N)+max(M,N).
If JOBZ = 'S' or 'A',
LWORK $\geq$ min(M,N)*min(M,N)+2*min(M,N)+max(M,N).
For good performance, LWORK should generally be larger.

If LWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK *CGESDD only* (workspace) REAL array, dimension (LRWORK)
If JOBZ = 'N', LRWORK $\geq$ 5*min(M,N).
Otherwise, LRWORK $\geq$ 5*min(M,N)*min(M,N) + 7*min(M,N).

IWORK (workspace) INTEGER array, dimension (8*min(M,N))

INFO (output) INTEGER
= 0: successful exit.
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: SBDSDC did not converge, updating process failed.

SGESV/CGESV

```
SUBROUTINE SGESV( N, NRHS, A, LDA, IPIV, B, LDB, INFO )
INTEGER            INFO, LDA, LDB, N, NRHS
INTEGER            IPIV( * )
REAL               A( LDA, * ), B( LDB, * )

SUBROUTINE CGESV( N, NRHS, A, LDA, IPIV, B, LDB, INFO )
INTEGER            INFO, LDA, LDB, N, NRHS
INTEGER            IPIV( * )
COMPLEX            A( LDA, * ), B( LDB, * )
```

Purpose

SGESV/CGESV computes the solution to a real/complex system of linear equations A*X = B, where A is an n-by-n matrix and X and B are n-by-nrhs matrices.

The LU decomposition with partial pivoting and row interchanges is used to factor A as A = P*L*U, where P is a permutation matrix, L is unit lower triangular, and U is upper triangular. The factored form of A is then used to solve the system of equations A*X = B.

Arguments

N (input) INTEGER
The number of linear equations, i.e., the order of the matrix A. N $\geq$ 0.

NRHS (input) INTEGER
The number of right hand sides, i.e., the number of columns of the matrix B. NRHS $\geq$ 0.

A (input/output) REAL/COMPLEX array, dimension (LDA,N)
On entry, the n-by-n coefficient matrix A.
On exit, the factors L and U from the factorization A = P*L*U; the unit diagonal elements of L are not stored.

LDA (input) INTEGER
The leading dimension of the array A. LDA $\geq$ max(1,N).

IPIV (output) INTEGER array, dimension (N)
The pivot indices that define the permutation matrix P; row i of the matrix was interchanged with row IPIV(i).

B (input/output) REAL/COMPLEX array, dimension (LDB,NRHS)
On entry, the n-by-nrhs matrix of right hand side matrix B.
On exit, if INFO = 0, the n-by-nrhs solution matrix X.

LDB (input) INTEGER
The leading dimension of the array B. LDB $\geq$ max(1,N).

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

> 0: if INFO = i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution could not be computed.

SGESVD/CGESVD

```
      SUBROUTINE SGESVD( JOBU, JOBVT, M, N, A, LDA, S, U, LDU, VT, LDVT,
     $                   WORK, LWORK, INFO )
      CHARACTER          JOBU, JOBVT
      INTEGER            INFO, LDA, LDU, LDVT, LWORK, M, N
      REAL               A( LDA, * ), S( * ), U( LDU, * ),
     $                   VT( LDVT, * ), WORK( * )

      SUBROUTINE CGESVD( JOBU, JOBVT, M, N, A, LDA, S, U, LDU, VT, LDVT,
     $                   WORK, LWORK, RWORK, INFO )
      CHARACTER          JOBU, JOBVT
      INTEGER            INFO, LDA, LDU, LDVT, LWORK, M, N
      REAL               RWORK( * ), S( * )
      COMPLEX            A( LDA, * ), U( LDU, * ), VT( LDVT, * ),
     $                   WORK( * )
```

Purpose

SGESVD/CGESVD computes the singular value decomposition (SVD) of a real/complex m-by-n matrix A, optionally computing the left and/or right singular vectors. The SVD is written

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

where Σ is an m-by-n matrix which is zero except for its min(m,n) diagonal elements, U is an m-by-m orthogonal/unitary matrix, and V is an n-by-n orthogonal/unitary matrix. The diagonal elements of Σ are the singular values of A; they are real and non-negative, and are returned in descending order. The first min(m,n) columns of U and V are the left and right singular vectors of A.

Note that the routine returns V^H, not V.

Arguments

JOBU (input) CHARACTER*1
Specifies options for computing all or part of the matrix U:
= 'A': all M columns of U are returned in array U;
= 'S': the first min(m,n) columns of U (the left singular vectors) are returned in the array U;
= 'O': the first min(m,n) columns of U (the left singular vectors) are overwritten on the array A;
= 'N': no columns of U (no left singular vectors) are computed.

JOBVT (input) CHARACTER*1
Specifies options for computing all or part of the matrix V^H:
= 'A': all N rows of V^H are returned in the array VT;
= 'S': the first min(m,n) rows of V^H (the right singular vectors) are returned in the array VT;
= 'O': the first min(m,n) rows of V^H (the right singular vectors) are overwritten on the array A;
= 'N': no rows of V^H (no right singular vectors) are computed.
JOBVT and JOBU cannot both be 'O'.

M (input) INTEGER
The number of rows of the matrix A. $M \geq 0$.

N (input) INTEGER
The number of columns of the matrix A. $N \geq 0$.

A (input/output) REAL/COMPLEX array, dimension (LDA,N)
On entry, the m-by-n matrix A.
On exit,
if JOBU = 'O', A is overwritten with the first min(m,n) columns of U (the left singular vectors, stored columnwise);
if JOBVT = 'O', A is overwritten with the first min(m,n) rows of V^H (the right singular vectors, stored rowwise);
if JOBU $\neq$ 'O' and JOBVT $\neq$ 'O', the contents of A are destroyed.

LDA (input) INTEGER
The leading dimension of the array A. LDA $\geq$ max(1,M).

S (output) REAL array, dimension (min(M,N))
The singular values of A, sorted so that S(i) $\geq$ S(i+1).

U (output) REAL/COMPLEX array, dimension
(LDU,M) if JOBU = 'A' or (LDU,min(M,N)) if JOBU = 'S'.
If JOBU = 'A', U contains the m-by-m orthogonal/unitary matrix U;
if JOBU = 'S', U contains the first min(m,n) columns of U (the left singular vectors, stored columnwise);
if JOBU = 'N' or 'O', U is not referenced.

LDU (input) INTEGER
The leading dimension of the array U. LDU $\geq$ 1; if JOBU = 'S' or 'A', LDU $\geq$ M.

VT (output) REAL/COMPLEX array, dimension (LDVT,N)
If JOBVT = 'A', VT contains the n-by-n orthogonal/unitary matrix V^H;
if JOBVT = 'S', VT contains the first min(m,n) rows of V^H (the right singular vectors, stored rowwise);
if JOBVT = 'N' or 'O', VT is not referenced.

LDVT (input) INTEGER
The leading dimension of the array VT. LDVT $\geq$ 1; if JOBVT = 'A', LDVT $\geq$ N; if JOBVT = 'S', LDVT $\geq$ min(M,N).

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK;

SGESVD
if INFO > 0, WORK(2:min(M,N)) contains the unconverged superdiagonal elements of an upper bidiagonal matrix B whose diagonal is in S (not necessarily sorted). B satisfies A = U*B*VT, so it has the same singular values as A, and singular vectors related by U and VT.

LWORK (input) INTEGER
The dimension of the array WORK. LWORK $\geq$ 1.
LWORK $\geq$ max(3*min(M,N)+max(M,N),5*min(M,N)). (*SGESVD*)
LWORK $\geq$ 2*min(M,N)+max(M,N). (*CGESVD*)
For good performance, LWORK should generally be larger.

If LWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK *CGESVD only* (workspace) REAL array, dimension (5*min(M,N))
On exit, if INFO > 0, RWORK(1:min(M,N)−1) contains the unconverged superdiagonal elements of an upper bidiagonal matrix B whose diagonal is in S (not necessarily sorted). B satisfies A = U*B*VT, so it has the same singular values as A, and singular vectors related by U and VT.

INFO (output) INTEGER
= 0: successful exit.
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if SBDSQR/CBDSQR did not converge, INFO specifies how many superdiagonals of an intermediate bidiagonal form B did not converge to zero.

SGESVX/CGESVX

```
      SUBROUTINE SGESVX( FACT, TRANS, N, NRHS, A, LDA, AF, LDAF, IPIV,
     $                   EQUED, R, C, B, LDB, X, LDX, RCOND, FERR, BERR,
     $                   WORK, IWORK, INFO )
      CHARACTER          EQUED, FACT, TRANS
      INTEGER            INFO, LDA, LDAF, LDB, LDX, N, NRHS
      REAL               RCOND
      INTEGER            IPIV( * ), IWORK( * )
      REAL               A( LDA, * ), AF( LDAF, * ), B( LDB, * ),
     $                   BERR( * ), C( * ), FERR( * ), R( * ),
     $                   WORK( * ), X( LDX, * )
```

```
      SUBROUTINE CGESVX( FACT, TRANS, N, NRHS, A, LDA, AF, LDAF, IPIV,
     $                   EQUED, R, C, B, LDB, X, LDX, RCOND, FERR, BERR,
     $                   WORK, RWORK, INFO )
      CHARACTER          EQUED, FACT, TRANS
      INTEGER            INFO, LDA, LDAF, LDB, LDX, N, NRHS
      REAL               RCOND
      INTEGER            IPIV( * )
      REAL               BERR( * ), C( * ), FERR( * ), R( * ),
     $                   RWORK( * )
      COMPLEX            A( LDA, * ), AF( LDAF, * ), B( LDB, * ),
     $                   WORK( * ), X( LDX, * )
```

Purpose

SGESVX/CGESVX uses the LU factorization to compute the solution to a real/complex system of linear equations A*X = B, where A is an n-by-n matrix and X and B are n-by-nrhs matrices.

Error bounds on the solution and a condition estimate are also provided.

Description

The following steps are performed:

1. If FACT = 'E', real scaling factors are computed to equilibrate the system:

   TRANS = 'N': diag(R)*A*diag(C)*diag(C)$^{-1}$*X = diag(R)*B
   TRANS = 'T': (diag(R)*A*diag(C))^{T}*diag(R)$^{-1}$*X = diag(C)*B
   TRANS = 'C': (diag(R)*A*diag(C))^{H}*diag(R)$^{-1}$*X = diag(C)*B

   Whether or not the system will be equilibrated depends on the scaling of the matrix A, but if equilibration is used, A is overwritten by diag(R)*A*diag(C) and B by diag(R)*B (if TRANS='N') or diag(C)*B (if TRANS = 'T' or 'C').

2. If FACT = 'N' or 'E', the LU decomposition is used to factor the matrix A (after equilibration if FACT = 'E') as A = P*L*U, where P is a permutation matrix, L is a unit lower triangular matrix, and U is upper triangular.

3. If some U(i,i)=0, so that U is exactly singular, then the routine returns with INFO = i. Otherwise, the factored form of A is used to estimate the condition number of the matrix A. If the reciprocal of the condition number is less than machine precision, INFO = N+1 is returned as a warning, but the routine still goes on to solve for X and compute error bounds as described below.

4. The system of equations is solved for X using the factored form of A.

5. Iterative refinement is applied to improve the computed solution matrix and calculate error bounds and backward error estimates for it.

6. If equilibration was used, the matrix X is premultiplied by diag(C) (if TRANS = 'N') or diag(R) (if TRANS = 'T' or 'C') so that it solves the original system before equilibration.

Arguments

FACT (input) CHARACTER*1
Specifies whether or not the factored form of the matrix A is supplied on entry, and if not, whether the matrix A should be equilibrated before it is factored.
= 'F': On entry, AF and IPIV contain the factored form of A.
If EQUED $\neq$ 'N', the matrix A has been equilibrated with scaling factors given by R and C.
A, AF, and IPIV are not modified.
= 'N': The matrix A will be copied to AF and factored.
= 'E': The matrix A will be equilibrated if necessary, then copied to AF and factored.

TRANS (input) CHARACTER*1
Specifies the form of the system of equations:
= 'N': A*X = B (No transpose)
= 'T': A^T*X = B (Transpose)
= 'C': A^H*X = B (Conjugate transpose)

N (input) INTEGER
The number of linear equations, i.e., the order of the matrix A. N $\geq$ 0.

NRHS (input) INTEGER
The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS $\geq$ 0.

A (input/output) REAL/COMPLEX array, dimension (LDA,N)
On entry, the n-by-n matrix A. If FACT = 'F' and EQUED $\neq$ 'N', then A must have been equilibrated by the scaling factors in R and/or C. A is not modified if FACT = 'F' or 'N', or if FACT = 'E' and EQUED = 'N' on exit.
On exit, if EQUED $\neq$ 'N', A is scaled as follows:
EQUED = 'R': A := diag(R)*A;
EQUED = 'C': A := A*diag(C);
EQUED = 'B': A := diag(R)*A*diag(C).

LDA (input) INTEGER
The leading dimension of the array A. LDA $\geq$ max(1,N).

AF (input or output) REAL/COMPLEX array, dimension (LDAF,N)
If FACT = 'F', then AF is an input argument and on entry contains the factors L and U from the factorization A = P*L*U as computed by SGETRF/CGETRF. If EQUED $\neq$ 'N', then AF is the factored form of the equilibrated matrix A.
If FACT = 'N', then AF is an output argument and on exit returns the factors L and U from the factorization A = P*L*U of the original matrix A.
If FACT = 'E', then AF is an output argument and on exit returns the factors L and U from the factorization A = P*L*U of the equilibrated matrix A (see the description of A for the form of the equilibrated matrix).

LDAF (input) INTEGER
The leading dimension of the array AF. LDAF $\geq$ max(1,N).

IPIV (input or output) INTEGER array, dimension (N)
If FACT = 'F', then IPIV is an input argument and on entry contains the pivot indices from the factorization A = P*L*U as computed by SGETRF/CGETRF; row i of the matrix was interchanged with row IPIV(i).
If FACT = 'N', then IPIV is an output argument and on exit contains the pivot indices from the factorization A = P*L*U of the original matrix A.
If FACT = 'E', then IPIV is an output argument and on exit contains the pivot indices from the factorization A = P*L*U of the equilibrated matrix A.

EQUED (input or output) CHARACTER*1
Specifies the form of equilibration that was done.
= 'N': No equilibration (always true if FACT = 'N').
= 'R': Row equilibration, i.e., A has been premultiplied by diag(R).
= 'C': Column equilibration, i.e., A has been postmultiplied by diag(C).
= 'B': Both row and column equilibration, i.e., A has been replaced by diag(R)*A*diag(C).
EQUED is an input argument if FACT = 'F'; otherwise, it is an output argument.

R (input or output) REAL array, dimension (N)
The row scale factors for A. If EQUED = 'R' or 'B', A is multiplied on the left by diag(R); if EQUED = 'N' or 'C', R is not accessed. R is an input argument if FACT = 'F'; otherwise, R is an output argument. If FACT = 'F' and EQUED = 'R' or 'B', each element of R must be positive.

C (input or output) REAL array, dimension (N)
The column scale factors for A. If EQUED = 'C' or 'B', A is multiplied on the right by diag(C); if EQUED = 'N' or 'R', C is not accessed. C is an input argument if FACT = 'F'; otherwise, C is an output argument. If FACT = 'F' and EQUED = 'C' or 'B', each element of C must be positive.

B (input/output) REAL/COMPLEX array, dimension (LDB,NRHS)
On entry, the n-by-nrhs right hand side matrix B.
On exit,
if EQUED = 'N', B is not modified;
if TRANS = 'N' and EQUED = 'R' or 'B', B is overwritten by diag(R)*B;
if TRANS = 'T' or 'C' or EQUED = 'C' or 'B', B is overwritten by diag(C)*B.

LDB (input) INTEGER
The leading dimension of the array B. LDB $\geq$ max(1,N).

X (output) REAL/COMPLEX array, dimension (LDX,NRHS)
If INFO = 0 or INFO = N+1, the N-by-NRHS solution matrix X to the original system of equations. Note that A and B are modified on exit if EQUED .ne. 'N', and the solution to the equilibrated system is $diag(C)^{-1}*X$ if TRANS = 'N' and EQUED = 'C' or 'B', or $diag(R)^{-1}*X$ if TRANS = 'T' or 'C' and EQUED = 'R' or 'B'.

LDX (input) INTEGER
The leading dimension of the array X. LDX $\geq$ max(1,N).

RCOND (output) REAL
The estimate of the reciprocal condition number of the matrix A after equilibration (if done). If RCOND is less than the machine precision (in particular, if RCOND = 0), the matrix is singular to working precision. This condition is indicated by a return code of INFO > 0.

FERR (output) REAL array, dimension (NRHS)
The estimated forward error bound for each solution vector X(j) (the jth column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) - XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR (output) REAL array, dimension (NRHS)
The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK *SGESVX* (workspace/output) REAL array, dimension (4*N)
CGESVX (workspace) COMPLEX array, dimension (2*N)
SGESVX only
On exit, WORK(1) contains the reciprocal pivot growth factor $||A||/||U||$. The "max absolute element" norm is used. If WORK(1) is much less than 1, then the stability of the LU factorization of the (equilibrated) matrix A could be poor. This also means that the solution X, condition estimator RCOND, and forward error bound FERR could be unreliable. If factorization fails with 0<INFO$\leq$N, then WORK(1) contains the reciprocal pivot growth factor for the leading INFO columns of A.

IWORK *SGESVX only* (workspace) INTEGER array, dimension (N)

RWORK *CGESVX only* (workspace/output) REAL array, dimension (2*N)
On exit, RWORK(1) contains the reciprocal pivot growth factor $||A||/||U||$. The "max absolute element" norm is used. If RWORK(1) is much less than 1, then the stability of the LU factorization of the (equilibrated) matrix A could be poor. This also means that the solution X, condition estimator RCOND, and forward error bound FERR could be unreliable. If factorization fails with 0<INFO$\leq$N, then RWORK(1) contains the reciprocal pivot growth factor for the leading INFO columns of A.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, and i is
$\leq$ N: U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned.
= N+1: U is nonsingular, but RCOND is less than machine precision, meaning that the matrix is singular to working precision. Nevertheless, the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the value of RCOND would suggest.

SGETRF/CGETRF

```
SUBROUTINE SGETRF( M, N, A, LDA, IPIV, INFO )
INTEGER            INFO, LDA, M, N
INTEGER            IPIV( * )
REAL               A( LDA, * )

SUBROUTINE CGETRF( M, N, A, LDA, IPIV, INFO )
INTEGER            INFO, LDA, M, N
INTEGER            IPIV( * )
COMPLEX            A( LDA, * )
```

Purpose

SGETRF/CGETRF computes an LU factorization of a general m-by-n matrix A using partial pivoting with row interchanges.

The factorization has the form A = P*L*U where P is a permutation matrix, L is lower triangular with unit diagonal elements (lower trapezoidal if m > n), and U is upper triangular (upper trapezoidal if m < n).

Arguments

M (input) INTEGER
The number of rows of the matrix A. M $\geq$ 0.

N (input) INTEGER
The number of columns of the matrix A. N $\geq$ 0.

A (input/output) REAL/COMPLEX array, dimension (LDA,N)
On entry, the m-by-n matrix to be factored.
On exit, the factors L and U from the factorization A = P*L*U; the unit diagonal elements of L are not stored.

LDA (input) INTEGER
The leading dimension of the array A. LDA $\geq$ max(1,M).

IPIV (output) INTEGER array, dimension (min(M,N))
The pivot indices; for $1 \leq i \leq$ min(M,N), row i of the matrix was interchanged with row IPIV(i).

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, and division by zero will occur if it is used to solve a system of equations.

SGETRI/CGETRI

```
SUBROUTINE SGETRI( N, A, LDA, IPIV, WORK, LWORK, INFO )
INTEGER            INFO, LDA, LWORK, N
INTEGER            IPIV( * )
REAL               A( LDA, * ), WORK( LWORK )

SUBROUTINE CGETRI( N, A, LDA, IPIV, WORK, LWORK, INFO )
INTEGER            INFO, LDA, LWORK, N
INTEGER            IPIV( * )
COMPLEX            A( LDA, * ), WORK( LWORK )
```

Purpose

SGETRI/CGETRI computes the inverse of a matrix using the LU factorization computed by SGETRF/CGETRF.

This method inverts U and then computes A^{-1} by solving the system $A^{-1}*L = U^{-1}$ for A^{-1}.

Arguments

N (input) INTEGER
The order of the matrix A. N $\geq$ 0.

A (input/output) REAL/COMPLEX array, dimension (LDA,N)
On entry, the factors L and U from the factorization A = P*L*U as computed by SGETRF/CGETRF.
On exit, if INFO = 0, the inverse of the original matrix A.

LDA (input) INTEGER
The leading dimension of the array A. LDA $\geq$ max(1,N).

IPIV (input) INTEGER array, dimension (N)
The pivot indices from SGETRF/CGETRF; for $1 \leq i \leq N$, row i of the matrix was interchanged with row IPIV(i).

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, then WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK. LWORK $\geq$ max(1,N). For optimal performance LWORK $\geq$ N*NB, where NB is the optimal blocksize returned by ILAENV.

If LWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, U(i,i) is exactly zero; the matrix is singular and its inverse could not be computed.

SGETRS/CGETRS

```
SUBROUTINE SGETRS( TRANS, N, NRHS, A, LDA, IPIV, B, LDB, INFO )
CHARACTER          TRANS
INTEGER            INFO, LDA, LDB, N, NRHS
INTEGER            IPIV( * )
REAL               A( LDA, * ), B( LDB, * )

SUBROUTINE CGETRS( TRANS, N, NRHS, A, LDA, IPIV, B, LDB, INFO )
CHARACTER          TRANS
INTEGER            INFO, LDA, LDB, N, NRHS
INTEGER            IPIV( * )
COMPLEX            A( LDA, * ), B( LDB, * )
```

Purpose

SGETRS/CGETRS solves a system of linear equations A*X = B, A^T*X = B, or A^H*X = B, with a general n-by-n matrix A using the LU factorization computed by SGETRF/CGETRF.

Arguments

TRANS (input) CHARACTER*1
Specifies the form of the system of equations:
= 'N': A*X = B (No transpose)
= 'T': A^T*X = B (Transpose)
= 'C': A^H*X = B (Conjugate transpose)

N (input) INTEGER
The order of the matrix A. N $\geq$ 0.

NRHS (input) INTEGER
The number of right hand sides, i.e., the number of columns of the matrix B. NRHS $\geq$ 0.

A (input) REAL/COMPLEX array, dimension (LDA,N)
The factors L and U from the factorization A = P*L*U as computed by SGETRF/CGETRF.

LDA (input) INTEGER
The leading dimension of the array A. LDA $\geq$ max(1,N).

IPIV (input) INTEGER array, dimension (N)
The pivot indices from SGETRF/CGETRF; for $1 \leq i \leq N$, row i of the matrix was interchanged with row IPIV(i).

B (input/output) REAL/COMPLEX array, dimension (LDB,NRHS)
On entry, the right hand side matrix B.
On exit, the solution matrix X.

LDB (input) INTEGER
The leading dimension of the array B. LDB $\geq$ max(1,N).

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SGGBAK/CGGBAK

```
      SUBROUTINE SGGBAK( JOB, SIDE, N, ILO, IHI, LSCALE, RSCALE, M, V,
     $                   LDV, INFO )
      CHARACTER          JOB, SIDE
      INTEGER            IHI, ILO, INFO, LDV, M, N
      REAL               LSCALE( * ), RSCALE( * ), V( LDV, * )

      SUBROUTINE CGGBAK( JOB, SIDE, N, ILO, IHI, LSCALE, RSCALE, M, V,
     $                   LDV, INFO )
      CHARACTER          JOB, SIDE
      INTEGER            IHI, ILO, INFO, LDV, M, N
      REAL               LSCALE( * ), RSCALE( * )
      COMPLEX            V( LDV, * )
```

Purpose

SGGBAK/CGGBAK forms the right or left eigenvectors of a real/complex generalized eigenvalue problem $A*x = \lambda*B*x$, by backward transformation on the computed eigenvectors of the balanced pair of matrices output by SGGBAL/CGGBAL.

Arguments

JOB (input) CHARACTER*1
Specifies the type of backward transformation required:
= 'N': do nothing, return immediately;
= 'P': do backward transformation for permutation only;
= 'S': do backward transformation for scaling only;
= 'B': do backward transformations for both permutation and scaling.
JOB must be the same as the argument JOB supplied to SGGBAL/CGGBAL.

SIDE (input) CHARACTER*1
= 'R': V contains right eigenvectors;
= 'L': V contains left eigenvectors.

N (input) INTEGER
The number of rows of the matrix V. N $\geq$ 0.

ILO, IHI (input) INTEGER
The integers ILO and IHI determined by SGGBAL/CGGBAL.
$1 \leq ILO \leq IHI \leq N$, if N > 0; ILO = 1 and IHI = 0, if N = 0.

LSCALE (input) REAL array, dimension (N)
Details of the permutations and/or scaling factors applied to the left side of A and B, as returned by SGGBAL/CGGBAL.

RSCALE (input) REAL array, dimension (N)
Details of the permutations and/or scaling factors applied to the right side of A and B, as returned by SGGBAL/CGGBAL.

M (input) INTEGER
The number of columns of the matrix V. M $\geq$ 0.

V (input/output) REAL/COMPLEX array, dimension (LDV,M)
On entry, the matrix of right or left eigenvectors to be transformed, as returned by STGEVC/CTGEVC.
On exit, V is overwritten by the transformed eigenvectors.

LDV (input) INTEGER
The leading dimension of the matrix V. LDV $\geq$ max(1,N).

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SGGBAL/CGGBAL

```
      SUBROUTINE SGGBAL( JOB, N, A, LDA, B, LDB, ILO, IHI, LSCALE,
     $                   RSCALE, WORK, INFO )
      CHARACTER          JOB
      INTEGER            IHI, ILO, INFO, LDA, LDB, N
      REAL               A( LDA, * ), B( LDB, * ), LSCALE( * ),
     $                   RSCALE( * ), WORK( * )
```

```
      SUBROUTINE CGGBAL( JOB, N, A, LDA, B, LDB, ILO, IHI, LSCALE,
     $                   RSCALE, WORK, INFO )
      CHARACTER          JOB
      INTEGER            IHI, ILO, INFO, LDA, LDB, N
      REAL               LSCALE( * ), RSCALE( * ), WORK( * )
      COMPLEX            A( LDA, * ), B( LDB, * )
```

Purpose

SGGBAL/CGGBAL balances a pair of general real/complex matrices (A,B). This involves, first, permuting A and B by similarity transformations to isolate eigenvalues in the first 1 to ilo−1 and last ihi+1 to n elements on the diagonal; and second, applying a diagonal similarity transformation to rows and columns ilo to ihi to make the rows and columns as close in norm as possible. Both steps are optional.

Balancing may reduce the 1-norm of the matrices, and improve the accuracy of the computed eigenvalues and/or eigenvectors in the generalized eigenvalue problem $A*x = \lambda*B*x$.

Arguments

JOB (input) CHARACTER*1
Specifies the operations to be performed on A and B:
= 'N': none: simply set ILO = 1, IHI = N, LSCALE(i) = 1.0 and RSCALE(i) = 1.0 for i=1,...,n;
= 'P': permute only;
= 'S': scale only;
= 'B': both permute and scale.

N (input) INTEGER
The order of the matrices A and B. N $\geq$ 0.

A (input/output) REAL/COMPLEX array, dimension (LDA,N)
On entry, the input matrix A.
On exit, A is overwritten by the balanced matrix.
If JOB = 'N', A is not referenced.

LDA (input) INTEGER
The leading dimension of the array A. LDA $\geq$ max(1,N).

B (input/output) REAL/COMPLEX array, dimension (LDB,N)
On entry, the input matrix B.
On exit, B is overwritten by the balanced matrix.
If JOB = 'N', B is not referenced.

LDB (input) INTEGER
The leading dimension of the array B. LDB $\geq$ max(1,N).

ILO, IHI (output) INTEGER
ILO and IHI are set to integers such that on exit A(i,j) = 0 and B(i,j) = 0 if i > j and j = 1,...,ilo−1 or i = ihi+1,...,n.
If JOB = 'N' or 'S', ILO = 1 and IHI = N.

LSCALE (output) REAL array, dimension (N)
Details of the permutations and scaling factors applied to the left side of A and B. If P(j) is the index of the row interchanged with row j, and D(j) is the scaling factor applied to row j, then

```
LSCALE(j)  = P(j)   for j = 1,...,ilo-1
           = D(j)   for j = ilo,...,ihi
           = P(j)   for j = ihi+1,...,n.
```

The order in which the interchanges are made is n to ihi+1, then 1 to ilo−1.

RSCALE (output) REAL array, dimension (N)
Details of the permutations and scaling factors applied to the right side of A and B. If P(j) is the index of the column interchanged with column j, and D(j) is the scaling factor applied to column j, then

```
RSCALE(j)  = P(j)   for j = 1,...,ilo-1
           = D(j)   for j = ilo,...,ihi
           = P(j)   for j = ihi+1,...,n.
```

The order in which the interchanges are made is n to ihi+1, then 1 to ilo−1.

WORK (workspace) REAL array, dimension (6*N)

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SGGES/CGGES

```
      SUBROUTINE SGGES( JOBVSL, JOBVSR, SORT, SELCTG, N, A, LDA, B, LDB,
     $                  SDIM, ALPHAR, ALPHAI, BETA, VSL, LDVSL, VSR,
     $                  LDVSR, WORK, LWORK, BWORK, INFO )
      CHARACTER          JOBVSL, JOBVSR, SORT
      INTEGER            INFO, LDA, LDB, LDVSL, LDVSR, LWORK, N, SDIM
      LOGICAL            BWORK( * )
      REAL               A( LDA, * ), ALPHAI( * ), ALPHAR( * ),
     $                   B( LDB, * ), BETA( * ), VSL( LDVSL, * ),
     $                   VSR( LDVSR, * ), WORK( * )
      LOGICAL            SELCTG
      EXTERNAL           SELCTG
```

```
      SUBROUTINE CGGES( JOBVSL, JOBVSR, SORT, SELCTG, N, A, LDA, B, LDB,
     $                  SDIM, ALPHA, BETA, VSL, LDVSL, VSR, LDVSR, WORK,
     $                  LWORK, RWORK, BWORK, INFO )
      CHARACTER          JOBVSL, JOBVSR, SORT
      INTEGER            INFO, LDA, LDB, LDVSL, LDVSR, LWORK, N, SDIM
      LOGICAL            BWORK( * )
      REAL               RWORK( * )
      COMPLEX            A( LDA, * ), ALPHA( * ), B( LDB, * ),
     $                   BETA( * ), VSL( LDVSL, * ), VSR( LDVSR, * ),
     $                   WORK( * )
      LOGICAL            SELCTG
      EXTERNAL           SELCTG
```

Purpose

SGGES/CGGES computes for a pair of N-by-N real/complex nonsymmetric matrices (A,B), the generalized eigenvalues, the generalized real/complex Schur form (S,T), optionally, the left and/or right matrices of Schur vectors (VSL and VSR). This gives the generalized Schur factorization

$$(A,B) = ((VSL)*S*(VSR)^H, (VSL)*T*(VSR)^H)$$

Optionally, it also orders the eigenvalues so that a selected cluster of eigenvalues appears in the leading diagonal blocks of the upper quasi-triangular matrix S and the upper triangular matrix T. The leading columns of VSL and VSR then form a(n) orthonormal/unitary basis for the corresponding left and right eigenspaces (deflating subspaces).

(If only the generalized eigenvalues are needed, use the driver SGGEV/CGGEV instead, which is faster.)

A generalized eigenvalue for a pair of matrices (A,B) is a scalar w or a ratio alpha/beta = w, such that A - w*B is singular. It is usually represented as the pair (alpha,beta), as there is a reasonable interpretation for beta=0 or both being zero.

A pair of matrices (S,T) is in generalized real Schur form if T is upper triangular with non-negative diagonal and S is block upper triangular with 1-by-1 and 2-by-2 blocks. 1-by-1 blocks correspond to real generalized eigenvalues, while 2-by-2 blocks of S will be "standardized" by making the corresponding elements of T have the form: $\begin{pmatrix} a & 0 \\ 0 & b \end{pmatrix}$ and the pair of corresponding 2-by-2 blocks in S and T will have a complex conjugate pair of generalized eigenvalues.

A pair of matrices (S,T) is in generalized complex Schur form if S and T are upper triangular and, in addition, the diagonal elements of T are non-negative real numbers.

Arguments

JOBVSL (input) CHARACTER*1
= 'N': do not compute the left Schur vectors;
= 'V': compute the left Schur vectors.

JOBVSR (input) CHARACTER*1
= 'N': do not compute the right Schur vectors;
= 'V': compute the right Schur vectors.

SORT (input) CHARACTER*1
Specifies whether or not to order the eigenvalues on the diagonal of the generalized Schur form.
= 'N': Eigenvalues are not ordered;
= 'S': Eigenvalues are ordered (see SELCTG).

SELCTG *SGGES* (input) LOGICAL FUNCTION of three REAL arguments
CGGES (input) LOGICAL FUNCTION of two COMPLEX arguments
SELCTG must be declared EXTERNAL in the calling subroutine.
If SORT = 'N', SELCTG is not referenced.
If SORT = 'S', SELCTG is used to select eigenvalues to sort to the top left of the Schur form.
SGGES
An eigenvalue (ALPHAR(j)+ALPHAI(j))/BETA(j) is selected if SELCTG(ALPHAR(j),ALPHAI(j),BETA(j)) is true; i.e. if either one of a complex conjugate pair of eigenvalues is selected, then both complex eigenvalues are selected.
Note that in the ill-conditioned case, a selected complex eigenvalue may no longer satisfy SELCTG(ALPHAR(j),ALPHAI(j), BETA(j)) = .TRUE. after ordering. INFO is to be set to N+2 in this case.
CGGES
An eigenvalue ALPHA(j)/BETA(j) is selected if SELCTG(ALPHA(j),BETA(j)) is true.
Note that a selected complex eigenvalue may no longer satisfy SELCTG(ALPHA(j),BETA(j)) = .TRUE. after ordering, since ordering may change the value of complex eigenvalues (especially if the eigenvalue is ill-conditioned), in this case INFO is set to N+2 (See INFO below).

N (input) INTEGER
The order of the matrices A, B, VSL, and VSR. N $\geq$ 0.

A (input/output) REAL/COMPLEX array, dimension (LDA, N)
On entry, the first of the pair of matrices.
On exit, A has been overwritten by its generalized Schur form S.

LDA (input) INTEGER
The leading dimension of A. LDA $\geq$ max(1,N).

B (input/output) REAL/COMPLEX array, dimension (LDB, N)
On entry, the second of the pair of matrices.
On exit, B has been overwritten by its generalized Schur form T.

LDB (input) INTEGER
The leading dimension of B. LDB $\geq$ max(1,N).

SDIM (output) INTEGER
If SORT = 'N', SDIM = 0. If SORT = 'S', SDIM = number of eigenvalues (after sorting) for which SELCTG is true. (Complex conjugate pairs for which SELCTG is true for either eigenvalue count as 2.)

ALPHAR *SGGES only* (output) REAL array, dimension (N)

ALPHAI *SGGES only* (output) REAL array, dimension (N)

ALPHA *CGGES only* (output) COMPLEX array, dimension (N)

BETA (output) REAL/COMPLEX array, dimension (N)
Note: the quotients ALPHAR(j)/BETA(j) and ALPHAI(j)/BETA(j) (*SGGES*) or ALPHA(j)/BETA(j) (*CGGES*) may easily over- or underflow, and BETA(j) may even be zero. Thus, the user should avoid naively computing the ratio alpha/beta. However, ALPHAR and ALPHAI (*SGGES*) or ALPHA (*CGGES*) will be always less than and usually comparable with norm(A) in magnitude, and BETA always less than and usually comparable with norm(B).
SGGES only
On exit, (ALPHAR(j) + ALPHAI(j)*i)/BETA(j), j=1,...,n, will be the generalized eigenvalues. ALPHAR(j) + ALPHAI(j)*i, j=1,...,n and BETA(j),j=1,...,n are the diagonals of the complex Schur form (A,B) that would result if the 2-by-2 diagonal blocks of the real Schur form of (A,B) were further reduced to triangular form using 2-by-2 complex unitary transformations. If ALPHAI(j) is zero, then the jth eigenvalue is real; if positive, then the jth and $(j+1)^{st}$ eigenvalues are a complex conjugate pair, with ALPHAI(j+1) negative.
CGGES only
On exit, ALPHA(j)/BETA(j), j=1,...,n, will be the generalized eigenvalues. ALPHA(j), j=1,...,n and BETA(j), j=1,...,n are the diagonals of the complex Schur form (A,B) output by CGGES. The BETA(j) will be non-negative real.

VSL (output) REAL/COMPLEX array, dimension (LDVSL,N)
If JOBVSL = 'V', VSL will contain the left Schur vectors.
Not referenced if JOBVSL = 'N'.

LDVSL (input) INTEGER
The leading dimension of the matrix VSL.
LDVSL $\geq$ 1, and if JOBVSL = 'V', LDVSL $\geq$ N.

VSR (output) REAL/COMPLEX array, dimension (LDVSR,N)
If JOBVSR = 'V', VSR will contain the right Schur vectors.
Not referenced if JOBVSR = 'N'.

LDVSR (input) INTEGER
The leading dimension of the matrix VSR.
LDVSR $\geq$ 1, and if JOBVSR = 'V', LDVSR $\geq$ N.

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK.
LWORK $\geq$ max(1,8*N+16). (*SGGES*)
LWORK $\geq$ max(1,2*N). (*CGGES*)
For good performance, LWORK must generally be larger.
If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK *CGGES only* (workspace) REAL array, dimension (8*N)

BWORK (workspace) LOGICAL array, dimension (N)
Not referenced if SORT = 'N'.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = $-$i, the ith argument had an illegal value.
=1,...,N: The QZ iteration failed. (A,B) are not in Schur form, but ALPHAR(j), ALPHAI(j) (*SGGES*) or ALPHA(j) (*CGGES*), and BETA(j) should be correct for j=info+1,...,n.
>N: errors that usually indicate LAPACK problems:
=N+1: other than QZ iteration failed in SHGEQZ/CHGEQZ
=N+2: after reordering, roundoff changed values of some complex eigenvalues so that leading eigenvalues in the Generalized Schur form no longer satisfy SELCTG=.TRUE. This could also be caused due to scaling.
=N+3: reordering failed in STGSEN/CTGSEN.

SGGESX/CGGESX

```
      SUBROUTINE SGGESX( JOBVSL, JOBVSR, SORT, SELCTG, SENSE, N, A, LDA,
     $                   B, LDB, SDIM, ALPHAR, ALPHAI, BETA, VSL, LDVSL,
     $                   VSR, LDVSR, RCONDE, RCONDV, WORK, LWORK, IWORK,
     $                   LIWORK, BWORK, INFO )
      CHARACTER          JOBVSL, JOBVSR, SENSE, SORT
      INTEGER            INFO, LDA, LDB, LDVSL, LDVSR, LIWORK, LWORK, N,
     $                   SDIM
      LOGICAL            BWORK( * )
      INTEGER            IWORK( * )
      REAL               A( LDA, * ), ALPHAI( * ), ALPHAR( * ),
     $                   B( LDB, * ), BETA( * ), RCONDE( 2 ),
     $                   RCONDV( 2 ), VSL( LDVSL, * ), VSR( LDVSR, * ),
     $                   WORK( * )
      LOGICAL            SELCTG
      EXTERNAL           SELCTG
```

```
      SUBROUTINE CGGESX( JOBVSL, JOBVSR, SORT, SELCTG, SENSE, N, A, LDA,
     $                   B, LDB, SDIM, ALPHA, BETA, VSL, LDVSL, VSR,
     $                   LDVSR, RCONDE, RCONDV, WORK, LWORK, RWORK,
     $                   IWORK, LIWORK, BWORK, INFO )
      CHARACTER          JOBVSL, JOBVSR, SENSE, SORT
      INTEGER            INFO, LDA, LDB, LDVSL, LDVSR, LIWORK, LWORK, N,
     $                   SDIM
      LOGICAL            BWORK( * )
      INTEGER            IWORK( * )
      REAL               RCONDE( 2 ), RCONDV( 2 ), RWORK( * )
      COMPLEX            A( LDA, * ), ALPHA( * ), B( LDB, * ),
     $                   BETA( * ), VSL( LDVSL, * ), VSR( LDVSR, * ),
     $                   WORK( * )
      LOGICAL            SELCTG
      EXTERNAL           SELCTG
```

Purpose

SGGESX/CGGESX computes for a pair of N-by-N real/complex nonsymmetric matrices (A,B), the generalized eigenvalues, the real/complex Schur form (S,T), and, optionally, the left and/or right matrices of Schur vectors (VSL and VSR). This gives the generalized Schur factorization

$$(A,B) = ((VSL)*S*(VSR)^H, (VSL)*T*(VSR)^H)$$

Optionally, it also orders the eigenvalues so that a selected cluster of eigenvalues appears in the leading diagonal blocks of the upper triangular matrix S and the upper triangular matrix T; computes a reciprocal condition number for the average of the selected eigenvalues (RCONDE); and computes a reciprocal condition number for the right and left deflating subspaces corresponding to the selected eigenvalues (RCONDV). The leading columns of VSL and VSR then form a(n) orthonormal/unitary basis for the corresponding left and right eigenspaces (deflating subspaces).

A generalized eigenvalue for a pair of matrices (A,B) is a scalar w or a ratio alpha/beta = w, such that A - w*B is singular. It is usually represented as the pair (alpha,beta), as there is a reasonable interpretation for beta=0 or for both being zero.

A pair of matrices (S,T) is in generalized real/complex Schur form if T is upper triangular with non-negative diagonal and S is upper triangular.

Arguments

JOBVSL (input) CHARACTER*1
= 'N': do not compute the left Schur vectors;
= 'V': compute the left Schur vectors.

JOBVSR (input) CHARACTER*1
= 'N': do not compute the right Schur vectors;
= 'V': compute the right Schur vectors.

SORT (input) CHARACTER*1
Specifies whether or not to order the eigenvalues on the diagonal of the generalized Schur form.
= 'N': Eigenvalues are not ordered;
= 'S': Eigenvalues are ordered (see SELCTG).

SELCTG *SGGESX* (input) LOGICAL FUNCTION of three REAL arguments
CGGESX (input) LOGICAL FUNCTION of two COMPLEX arguments
SELCTG must be declared EXTERNAL in the calling subroutine.
If SORT = 'N', SELCTG is not referenced.
If SORT = 'S', SELCTG is used to select eigenvalues to sort to the top left of the Schur form.
SGGESX
An eigenvalue (ALPHAR(j)+ALPHAI(j))/BETA(j) is selected if SELCTG(ALPHAR(j),ALPHAI(j),BETA(j)) is true; i.e. if either one of a complex conjugate pair of eigenvalues is selected, then both complex eigenvalues are selected.
Note that a selected complex eigenvalue may no longer satisfy SELCTG(ALPHAR(j),ALPHAI(j),BETA(j)) = .TRUE. after ordering, since ordering may change the value of complex eigenvalues (especially if the eigenvalue is ill-conditioned), in this case INFO is set to N+3.
CGGESX
An eigenvalue ALPHA(j)/BETA(j) is selected if SELCTG(ALPHA(j),BETA(j)) is true.
Note that a selected complex eigenvalue may no longer satisfy SELCTG(ALPHA(j),BETA(j)) = .TRUE. after ordering, since ordering may change the value of complex eigenvalues (especially if the eigenvalue is ill-conditioned), in this case INFO is set to N+3 (See INFO below).

SENSE (input) CHARACTER*1
Determines which reciprocal condition numbers are computed.
= 'N': None are computed;
= 'E': Computed for average of selected eigenvalues only;
= 'V': Computed for selected deflating subspaces only;
= 'B': Computed for both.
If SENSE = 'E', 'V', or 'B', SORT must equal 'S'.

N (input) INTEGER
The order of the matrices A, B, VSL, and VSR. N $\geq$ 0.

A (input/output) REAL/COMPLEX array, dimension (LDA, N)
On entry, the first of the pair of matrices.
On exit, A has been overwritten by its generalized Schur form S.

LDA (input) INTEGER
The leading dimension of A. LDA $\geq$ max(1,N).

B (input/output) REAL/COMPLEX array, dimension (LDB, N)
On entry, the second of the pair of matrices.
On exit, B has been overwritten by its generalized Schur form T.

LDB (input) INTEGER
The leading dimension of B. LDB $\geq$ max(1,N).

SDIM (output) INTEGER
If SORT = 'N', SDIM = 0. If SORT = 'S', SDIM = number of eigenvalues (after sorting) for which SELCTG is true. (Complex conjugate pairs for which SELCTG is true for either eigenvalue count as 2.)

ALPHAR *SGGESX only* (output) REAL array, dimension (N)

ALPHAI *SGGESX only* (output) REAL array, dimension (N)

ALPHA *CGGESX only* (output) COMPLEX array, dimension (N)

BETA (output) REAL/COMPLEX array, dimension (N)
Note: the quotients ALPHAR(j)/BETA(j) and ALPHAI(j)/BETA(j) (*SGGESX*) or ALPHA(j)/BETA(j) (*CGGESX*) may easily over- or underflow, and BETA(j) may even be zero. Thus, the user should avoid naively computing the ratio alpha/beta. However, ALPHAR and ALPHAI (*SGGESX*) or ALPHA (*CGGESX*) will be always less than and usually comparable with norm(A) in magnitude, and BETA always less than and usually comparable with norm(B).
SGGESX only
On exit, (ALPHAR(j) + ALPHAI(j)*i)/BETA(j), j=1,...,n, will be the generalized eigenvalues. ALPHAR(j) + ALPHAI(j)*i, j=1,...,n and BETA(j),j=1,...,n are the diagonals of the complex Schur form (A,B) that would result if the 2-by-2 diagonal blocks of the real Schur form of (A,B) were further reduced to triangular form using 2-by-2 complex unitary transformations. If ALPHAI(j) is zero, then the jth eigenvalue is real; if positive, then the jth and $(j+1)^{st}$ eigenvalues are a complex conjugate pair, with ALPHAI(j+1) negative.
CGGESX only
On exit, ALPHA(j)/BETA(j), j=1,...,n, will be the generalized eigenvalues. ALPHA(j), j=1,...,n and BETA(j), j=1,...,n are the diagonals of the complex Schur form (A,B) output by CGGESX. The BETA(j) will be non-negative real.

VSL (output) REAL/COMPLEX array, dimension (LDVSL,N)
If JOBVSL = 'V', VSL will contain the left Schur vectors.
Not referenced if JOBVSL = 'N'.

LDVSL (input) INTEGER
The leading dimension of the matrix VSL. LDVSL $\geq$ 1, and if JOBVSL = 'V', LDVSL $\geq$ N.

VSR (output) REAL/COMPLEX array, dimension (LDVSR,N)
If JOBVSR = 'V', VSR will contain the right Schur vectors.
Not referenced if JOBVSR = 'N'.

LDVSR (input) INTEGER
The leading dimension of the matrix VSR. LDVSR $\geq$ 1, and if JOBVSR = 'V', LDVSR $\geq$ N.

RCONDE (output) REAL array, dimension (2)
If SENSE = 'E' or 'B', RCONDE(1) and RCONDE(2) contain the reciprocal condition numbers for the average of the selected eigenvalues.
Not referenced if SENSE = 'N' or 'V'.

RCONDV (output) REAL array, dimension (2)
If SENSE = 'V' or 'B', RCONDV(1) and RCONDV(2) contain the reciprocal condition numbers for the selected deflating subspaces.
Not referenced if SENSE = 'N' or 'E'.

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK.
SGGESX
LWORK $\geq$ max(1,8*(N+1)+16). If SENSE = 'E', 'V', or 'B', LWORK $\geq$ MAX(8*(N+1)+16, 2*SDIM*(N-SDIM)).
CGGESX
LWORK $\geq$ max(1,2*N). If SENSE = 'E', 'V', or 'B', LWORK $\geq$ MAX(2*N, 2*SDIM*(N-SDIM)).
For good performance, LWORK must generally be larger.

RWORK *CGGESX only* (workspace) REAL array, dimension (8*N)

IWORK (workspace) INTEGER array, dimension (LIWORK)
Not referenced if SENSE = 'N'.

LIWORK (input) INTEGER
The dimension of the array WORK. LIWORK $\geq$ N+6. (*SGGESX*)
The dimension of the array WORK. LIWORK $\geq$ N+2. (*CGGESX*)

BWORK (workspace) LOGICAL array, dimension (N)
Not referenced if SORT = 'N'.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = $-i$, the ith argument had an illegal value.
=1,...,N: The QZ iteration failed. (A,B) are not in Schur form, but ALPHAR(j), ALPHAI(j) (*SGGESX*) or ALPHA(j) (*CGGESX*), and BETA(j) should be correct for j=info+1,...,n.
>N: errors that usually indicate LAPACK problems:
=N+1: other than QZ iteration failed in SHGEQZ/CHGEQZ
=N+2: after reordering, roundoff changed values of some complex eigenvalues so that leading eigenvalues in the Generalized Schur form no longer satisfy SELCTG=.TRUE. This could also be caused due to scaling.
=N+3: reordering failed in STGSEN/CTGSEN.

SGGEV/CGGEV

```
      SUBROUTINE SGGEV( JOBVL, JOBVR, N, A, LDA, B, LDB, ALPHAR, ALPHAI,
     $                  BETA, VL, LDVL, VR, LDVR, WORK, LWORK, INFO )
      CHARACTER          JOBVL, JOBVR
      INTEGER            INFO, LDA, LDB, LDVL, LDVR, LWORK, N
      REAL               A( LDA, * ), ALPHAI( * ), ALPHAR( * ),
     $                   B( LDB, * ), BETA( * ), VL( LDVL, * ),
     $                   VR( LDVR, * ), WORK( * )

      SUBROUTINE CGGEV( JOBVL, JOBVR, N, A, LDA, B, LDB, ALPHA, BETA,
     $                  VL, LDVL, VR, LDVR, WORK, LWORK, RWORK, INFO )
      CHARACTER          JOBVL, JOBVR
      INTEGER            INFO, LDA, LDB, LDVL, LDVR, LWORK, N
      REAL               RWORK( * )
      COMPLEX            A( LDA, * ), ALPHA( * ), B( LDB, * ),
     $                   BETA( * ), VL( LDVL, * ), VR( LDVR, * ),
     $                   WORK( * )
```

Purpose

SGGEV/CGGEV computes for a pair of N-by-N real/complex nonsymmetric matrices (A,B), the generalized eigenvalues, and optionally, the left and/or right generalized eigenvectors.

A generalized eigenvalue for a pair of matrices (A,B) is a scalar λ or a ratio alpha/beta = λ, such that A - λ*B is singular. It is usually represented as the pair (alpha,beta), as there is a reasonable interpretation for beta=0, and even for both being zero.

The right generalized eigenvector $v(j)$ corresponding to the generalized eigenvalue $\lambda(j)$ of (A,B) satisfies

$$A * v(j) \quad = \quad \lambda(j) * B * v(j).$$

The left generalized eigenvector $u(j)$ corresponding to the generalized eigenvalues $\lambda(j)$ of (A,B) satisfies

$$u(j)^H * A \quad = \quad \lambda(j) * u(j)^H * B$$

where $u(j)^H$ denotes the conjugate-transpose of $u(j)$.

Arguments

JOBVL (input) CHARACTER*1
= 'N': do not compute the left generalized eigenvectors;
= 'V': compute the left generalized eigenvectors.

JOBVR (input) CHARACTER*1
= 'N': do not compute the right generalized eigenvectors;
= 'V': compute the right generalized eigenvectors.

N (input) INTEGER
The order of the matrices A, B, VL, and VR. N $\geq$ 0.

A (input/output) REAL/COMPLEX array, dimension (LDA, N)
On entry, the matrix A in the pair (A,B).
On exit, A has been overwritten.

LDA (input) INTEGER
The leading dimension of A. LDA $\geq$ max(1,N).

B (input/output) REAL/COMPLEX array, dimension (LDB, N)
On entry, the matrix B in the pair (A,B).
On exit, B has been overwritten.

LDB (input) INTEGER
The leading dimension of B. LDB $\geq$ max(1,N).

ALPHAR *SGGEV only* (output) REAL array, dimension (N)

ALPHAI *SGGEV only* (output) REAL array, dimension (N)

ALPHA *CGGEV only* (output) COMPLEX array, dimension (N)

BETA (output) REAL/COMPLEX array, dimension (N)
Note: the quotients ALPHAR(j)/BETA(j) and ALPHAI(j)/BETA(j) (*SGGEV*) or ALPHA(j)/BETA(j) (*CGGEV*) may easily over- or underflow, and BETA(j) may even be zero. Thus, the user should avoid naively computing the ratio alpha/beta. However, ALPHAR and ALPHAI (*SGGEV*) or ALPHA (*CGGEV*) will be always less than and usually comparable with norm(A) in magnitude, and BETA always less than and usually comparable with norm(B).
SGGES only
On exit, (ALPHAR(j) + ALPHAI(j)*i)/BETA(j), j=1,...,n, will be the generalized eigenvalues. If ALPHAI(j) is zero, then the jth eigenvalue is real; if positive, then the jth and $(j+1)^{st}$ eigenvalues are a complex conjugate pair, with ALPHAI(j+1) negative.
CGGES only
On exit, ALPHA(j)/BETA(j), j=1,...,n, will be the generalized eigenvalues.

VL (output) REAL/COMPLEX array, dimension (LDVL,N)
SGGEV
If JOBVL = 'V', the left eigenvectors u(j) are stored one after another in the columns of VL, in the same order as their eigenvalues. If the jth eigenvalue is real, then u(j) = VL(:,j), the jth column of VL. If the jth and $(j+1)^{th}$ eigenvalues form a complex conjugate pair, then u(j) = VL(:,j)+i*VL(:,j+1) and u(j+1) = VL(:,j)−i*VL(:,j+1). Each eigenvector will be scaled so the largest component have abs(real part)+abs(imag. part)=1. Not referenced if JOBVL = 'N'.
CGGEV
If JOBVL = 'V', the left generalized eigenvectors u(j) are stored one after another in the columns of VL, in the same order as their eigenvalues. Each eigenvector will be scaled so the largest component will have abs(real part) + abs(imag. part) = 1. Not referenced if JOBVL = 'N'.

LDVL (input) INTEGER
The leading dimension of the matrix VL.
LDVL $\geq$ 1, and if JOBVL = 'V', LDVL $\geq$ N.

VR (output) REAL/COMPLEX array, dimension (LDVR,N)
Not referenced if JOBVR = 'N'.
SGGEV
If JOBVR = 'V', the right eigenvectors v(j) are stored one after another in the columns of VR, in the same order as their eigenvalues. If the jth eigenvalue is real, then v(j) = VR(:,j), the j-th column of VR. If the jth and $(j+1)^{th}$ eigenvalues form a complex conjugate pair, then v(j) = VR(:,j)+i*VR(:,j+1) and v(j+1) = VR(:,j)−i*VR(:,j+1). Each eigenvector will be scaled so the largest component have abs(real part)+abs(imag. part)=1. *CGGEV*
If JOBVR = 'V', the right generalized eigenvectors v(j) are stored one after another in the columns of VR, in the same order as their eigenvalues. Each eigenvector will be scaled so the largest component will have abs(real part) + abs(imag. part) = 1.

LDVR (input) INTEGER
The leading dimension of the matrix VR.
LDVR $\geq$ 1, and if JOBVR = 'V', LDVR $\geq$ N.

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK.
LWORK $\geq$ max(1,8*N+16). (*SGGEV*)
LWORK $\geq$ max(1,2*N). (*CGGEV*)
For good performance, LWORK must generally be larger.

If LWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK *CGGEV only* (workspace) REAL array, dimension (8*N)

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
=1,...,N: The QZ iteration failed. No eigenvectors have been calculated, but ALPHAR(j), ALPHAI(j) (*SGGEV*) or ALPHA(j) (*CGGEV*), and BETA(j) should be correct for j=info+1,...,n.
>N: errors that usually indicate LAPACK problems:
=N+1: other than QZ iteration failed in SHGEQZ/CHGEQZ.
=N+2: error return from STGEVC/CTGEVC.

SGGEVX/CGGEVX

```
      SUBROUTINE SGGEVX( BALANC, JOBVL, JOBVR, SENSE, N, A, LDA, B, LDB,
     $                   ALPHAR, ALPHAI, BETA, VL, LDVL, VR, LDVR, ILO,
     $                   IHI, LSCALE, RSCALE, ABNRM, BBNRM, RCONDE,
     $                   RCONDV, WORK, LWORK, IWORK, BWORK, INFO )
      CHARACTER          BALANC, JOBVL, JOBVR, SENSE
      INTEGER            IHI, ILO, INFO, LDA, LDB, LDVL, LDVR, LWORK, N
      REAL               ABNRM, BBNRM
      LOGICAL            BWORK( * )
      INTEGER            IWORK( * )
      REAL               A( LDA, * ), ALPHAI( * ), ALPHAR( * ),
     $                   B( LDB, * ), BETA( * ), LSCALE( * ),
     $                   RCONDE( * ), RCONDV( * ), RSCALE( * ),
     $                   VL( LDVL, * ), VR( LDVR, * ), WORK( * )

      SUBROUTINE CGGEVX( BALANC, JOBVL, JOBVR, SENSE, N, A, LDA, B, LDB,
     $                   ALPHA, BETA, VL, LDVL, VR, LDVR, ILO, IHI,
     $                   LSCALE, RSCALE, ABNRM, BBNRM, RCONDE, RCONDV,
     $                   WORK, LWORK, RWORK, IWORK, BWORK, INFO )
      CHARACTER          BALANC, JOBVL, JOBVR, SENSE
      INTEGER            IHI, ILO, INFO, LDA, LDB, LDVL, LDVR, LWORK, N
      REAL               ABNRM, BBNRM
      LOGICAL            BWORK( * )
      INTEGER            IWORK( * )
      REAL               LSCALE( * ), RCONDE( * ), RCONDV( * ),
     $                   RSCALE( * ), RWORK( * )
      COMPLEX            A( LDA, * ), ALPHA( * ), B( LDB, * ),
     $                   BETA( * ), VL( LDVL, * ), VR( LDVR, * ),
     $                   WORK( * )
```

Purpose

SGGEVX computes for a pair of N-by-N real nonsymmetric matrices (A,B) the generalized eigenvalues, and optionally, the left and/or right generalized eigenvectors.

Optionally also, it computes a balancing transformation to improve the conditioning of the eigenvalues and eigenvectors (ILO, IHI, LSCALE, RSCALE, ABNRM, and BBNRM), reciprocal condition numbers for the eigenvalues (RCONDE), and reciprocal condition numbers for the right eigenvectors (RCONDV).

A generalized eigenvalue for a pair of matrices (A,B) is a scalar λ or a ratio alpha/beta = λ, such that A $-\lambda$*B is singular. It is usually represented as the pair (alpha,beta), as there is a reasonable interpretation for beta=0, and even for both being zero.

The right eigenvector $v(j)$ corresponding to the eigenvalue $\lambda(j)$ of (A,B) satisfies

$$A * v(j) \quad = \quad \lambda(j) * B * v(j).$$

The left eigenvector $u(j)$ corresponding to the eigenvalue $\lambda(j)$ of (A,B) satisfies

$$u(j)^H * A \quad = \quad \lambda(j) * u(j)^H * B$$

where $u(j)^H$ denotes the conjugate-transpose of $u(j)$.

Arguments

BALANC (input) CHARACTER*1
Specifies the balance option to be performed.
= 'N': do not diagonally scale or permute;
= 'P': permute only;
= 'S': scale only;
= 'B': both permute and scale.
Computed reciprocal condition numbers will be for the matrices after permuting and/or balancing. Permuting does not change condition numbers (in exact arithmetic), but balancing does.

JOBVL (input) CHARACTER*1
= 'N': do not compute the left generalized eigenvectors;
= 'V': compute the left generalized eigenvectors.

JOBVR (input) CHARACTER*1
= 'N': do not compute the right generalized eigenvectors;
= 'V': compute the right generalized eigenvectors.

SENSE (input) CHARACTER*1
Determines which reciprocal condition numbers are computed.
= 'N': none are computed;
= 'E': computed for eigenvalues only;
= 'V': computed for eigenvectors only;
= 'B': computed for eigenvalues and eigenvectors.

N (input) INTEGER
The order of the matrices A, B, VL, and VR. N $\geq$ 0.

A (input/output) REAL/COMPLEX array, dimension (LDA, N)
On entry, the matrix A in the pair (A,B).
On exit, A has been overwritten. If JOBVL='V' or JOBVR='V' or both, then A contains the first part of the real Schur form of the "balanced" versions of the input A and B.

LDA (input) INTEGER
The leading dimension of A. LDA $\geq$ max(1,N).

B (input/output) REAL/COMPLEX array, dimension (LDB, N)
On entry, the matrix B in the pair (A,B).
On exit, B has been overwritten. If JOBVL='V' or JOBVR='V' or both, then B contains the second part of the real Schur form of the "balanced" versions of the input A and B.

LDB (input) INTEGER
The leading dimension of B. LDB $\geq$ max(1,N).

ALPHAR *SGGEVX only* (output) REAL array, dimension (N)

ALPHAI *SGGEVX only* (output) REAL array, dimension (N)

ALPHA *CGGEVX only* (output) COMPLEX array, dimension (N)

BETA (output) REAL/COMPLEX array, dimension (N)
Note: the quotients ALPHAR(j)/BETA(j) and ALPHAI(j)/BETA(j) (*SGGEVX*) or ALPHA(j)/BETA(j) (*CGGEVX*) may easily over- or underflow, and BETA(j) may even be zero. Thus, the user should avoid naively computing the ratio alpha/beta. However, ALPHAR and ALPHAI (*SGGEVX*) or ALPHA (*CGGEVX*) will be always less than and usually comparable with norm(A) in magnitude, and BETA always less than and usually comparable with norm(B).
SGGEVX only
On exit, (ALPHAR(j) + ALPHAI(j)*i)/BETA(j), j=1,...,n, will be the generalized eigenvalues. If ALPHAI(j) is zero, then the jth eigenvalue is real; if positive, then the jth and $(j+1)^{st}$ eigenvalues are a complex conjugate pair, with ALPHAI(j+1) negative.
CGGEVX only
On exit, ALPHA(j)/BETA(j), j=1,...,n, will be the generalized eigenvalues.

VL (output) REAL/COMPLEX array, dimension (LDVL,N)
Not referenced if JOBVL = 'N'.
SGGEVX
If JOBVL = 'V', the left eigenvectors u(j) are stored one after another in the columns of VL, in the same order as their eigenvalues. If the jth eigenvalue is real, then u(j) = VL(:,j), the jth column of VL. If the jth and $(j+1)^{st}$ eigenvalues form a complex conjugate pair, then u(j) = VL(:,j)+i*VL(:,j+1) and u(j+1) = VL(:,j)−i*VL(:,j+1). Each eigenvector will be scaled so the largest component have abs(real part) + abs(imag. part) = 1.
CGGEVX
If JOBVL = 'V', the left generalized eigenvectors u(j) are stored one after another in the columns of VL, in the same order as their eigenvalues. Each eigenvector will be scaled so the largest component will have abs(real part) + abs(imag. part) = 1.

LDVL (input) INTEGER
The leading dimension of the matrix VL.
LDVL $\geq$ 1, and if JOBVL = 'V', LDVL $\geq$ N.

VR (output) REAL/COMPLEX array, dimension (LDVR,N)
Not referenced if JOBVR = 'N'.
SGGEVX
If JOBVR = 'V', the right eigenvectors v(j) are stored one after another in the columns of VR, in the same order as their eigenvalues. If the jth eigenvalue is real, then v(j) = VR(:,j), the jth column of VR. If the jth and $(j+1)^{st}$ eigenvalues form a complex conjugate pair, then v(j) = VR(:,j)+i*VR(:,j+1) and v(j+1) = VR(:,j)−i*VR(:,j+1). Each eigenvector will be scaled so the largest component have abs(real part)

+ abs(imag. part) = 1.
CGGEVX
If JOBVL = 'V', the right generalized eigenvectors v(j) are stored one after another in the columns of VR, in the same order as their eigenvalues. Each eigenvector will be scaled so the largest component will have abs(real part) + abs(imag. part) = 1.

LDVR (input) INTEGER
The leading dimension of the matrix VR.
LDVR $\geq$ 1, and if JOBVR = 'V', LDVR $\geq$ N.

ILO, IHI (output) INTEGER
ILO and IHI are integer values such that on exit A(i,j) = 0 and B(i,j) = 0 if i > j and
j = 1,...,ilo−1 or i = ihi+1,...,n.
If BALANC = 'N' or 'S', ilo = 1 and ihi = N.

LSCALE (output) REAL array, dimension (N)
Details of the permutations and scaling factors applied to the left side of A and B. If PL(j) is the index of the row interchanged with row j, and DL(j) is the scaling factor applied to row j, then

```
LSCALE(j)  = PL(j)   for j = 1,...,ilo-1
           = DL(j)   for j = ilo,...,ihi
           = PL(j)   for j = ihi+1,...,n.
```

The order in which the interchanges are made is n to ihi+1, then 1 to ilo−1.

RSCALE (output) REAL array, dimension (N)
Details of the permutations and scaling factors applied to the right side of A and B. If PR(j) is the index of the column interchanged with column j, and DR(j) is the scaling factor applied to column j, then

```
RSCALE(j)  = PR(j)   for j = 1,...,ilo-1
           = DR(j)   for j = ilo,...,ihi
           = PR(j)   for j = ihi+1,...,n.
```

The order in which the interchanges are made is n to ihi+1, then 1 to ilo−1.

ABNRM (output) REAL
The one-norm of the balanced matrix A.

BBNRM (output) REAL
The one-norm of the balanced matrix B.

RCONDE (output) REAL array, dimension (N)
If SENSE = 'V', RCONDE is not referenced.
SGGEVX
If SENSE = 'E' or 'B', the reciprocal condition numbers of the selected eigenvalues, stored in consecutive elements of the array. For a complex conjugate pair of eigenvalues two consecutive elements of RCONDE are set to the same value. Thus RCONDE(j), RCONDV(j), and the jth columns of VL and VR all correspond to the same eigenpair (but not in general the jth eigenpair, unless all eigenpairs are selected). If SENSE = 'V', RCONDE is not referenced.
CGGEVX
If SENSE = 'E' or 'B', the reciprocal condition numbers of the selected eigenvalues, stored in consecutive elements of the array.

RCONDV (output) REAL array, dimension (N)
If SENSE = 'E', RCONDV is not referenced.
SGGEVX
If SENSE = 'V' or 'B', the estimated reciprocal condition numbers of the selected eigenvectors, stored in consecutive elements of the array. For a complex eigenvector two consecutive elements of RCONDV are set to the same value. If the eigenvalues cannot be reordered to compute RCONDV(j), RCONDV(j) is set to 0; this can only occur when the true value would be very small anyway.
CGGEVX
If SENSE = 'V' or 'B', the estimated reciprocal condition numbers of the selected eigenvectors, stored in consecutive elements of the array. If the eigenvalues cannot be reordered to compute RCONDV(j), RCONDV(j) is set to 0; this can only occur when the true value would be very small anyway.

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK.
SGGEVX
The dimension of the array WORK. LWORK $\geq$ max(1,6*N).
If SENSE = 'E', LWORK $\geq$ 12*N.
If SENSE = 'V' or 'B', LWORK $\geq$ 2*N*N+12*N+16.
CGGEVX
The dimension of the array WORK. LWORK $\geq$ max(1,2*N).
If SENSE = 'N' or 'E', LWORK $\geq$ 2*N.
If SENSE = 'V' or 'B', LWORK $\geq$ 2*N*N+2*N.

If LWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK *CGGEVX only* (workspace) REAL array, dimension (6*N)

IWORK *SGGEVX* (workspace) INTEGER array, dimension (N+6)
CGGEVX (workspace) INTEGER array, dimension (N+2)
If SENSE = 'E', IWORK is not referenced.

BWORK (workspace) LOGICAL array, dimension (N)
If SENSE = 'N', BWORK is not referenced.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
=1,...,N: The QZ iteration failed. No eigenvectors have been calculated, but ALPHAR(j), ALPHAI(j) (*SGGEVX*) or ALPHA(j)

(*CGGEVX*), and BETA(j) should be correct for j=info+1,...,n.
>N: errors that usually indicate LAPACK problems:
=N+1: other than QZ iteration failed in SHGEQZ/CHGEQZ
=N+2: error return from STGEVC/CTGEVC.

SGGGLM/CGGGLM

```
      SUBROUTINE SGGGLM( N, M, P, A, LDA, B, LDB, D, X, Y, WORK, LWORK,
     $                   INFO )
      INTEGER            INFO, LDA, LDB, LWORK, M, N, P
      REAL               A( LDA, * ), B( LDB, * ), D( * ), WORK( * ),
     $                   X( * ), Y( * )

      SUBROUTINE CGGGLM( N, M, P, A, LDA, B, LDB, D, X, Y, WORK, LWORK,
     $                   INFO )
      INTEGER            INFO, LDA, LDB, LWORK, M, N, P
      COMPLEX            A( LDA, * ), B( LDB, * ), D( * ), WORK( * ),
     $                   X( * ), Y( * )
```

Purpose

SGGGLM/CGGGLM solves a general Gauss-Markov linear model (GLM) problem:

$$\min_x \| y \|_2 \text{ subject to } d = A*x + B*y$$

where A is an n-by-m matrix, B is an n-by-p matrix, and d is a given n-vector. It is assumed that $m \leq n \leq m+p$ and

$$\text{rank}(A) = m, \quad \text{and} \quad \text{rank}(A\ B) = n.$$

Under these assumptions, the constrained equation is always consistent, and there is a unique solution x and a minimal 2-norm solution y, which is obtained using a generalized QR factorization of A and B.

In particular, if matrix B is square nonsingular, then the problem GLM is equivalent to the following weighted linear least squares problem

$$\min_x \| B^{-1}*(d - A*x) \|_2.$$

Arguments

N (input) INTEGER
The number of rows of the matrices A and B. $N \geq 0$.

M (input) INTEGER
The number of columns of the matrix A. $0 \leq M \leq N$.

P (input) INTEGER
The number of columns of the matrix B. $P \geq N-M$.

A (input/output) REAL/COMPLEX array, dimension (LDA,M)
On entry, the n-by-m matrix A.
On exit, A is destroyed.

LDA (input) INTEGER
The leading dimension of the array A. $LDA \geq \max(1,N)$.

B (input/output) REAL/COMPLEX array, dimension (LDB,P)
On entry, the n-by-p matrix B.
On exit, B is destroyed.

LDB (input) INTEGER
The leading dimension of the array B. $LDB \geq \max(1,N)$.

D (input/output) REAL/COMPLEX array, dimension (N)
On entry, D is the left hand side of the GLM equation.
On exit, D is destroyed.

X (output) REAL/COMPLEX array, dimension (M)

Y (output) REAL/COMPLEX array, dimension (P)
On exit, X and Y are the solutions of the GLM problem.

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK. $LWORK \geq \max(1,N+M+P)$.
For optimum performance $LWORK \geq M+\min(N,P)+\max(N,P)*NB$, where NB is an upper bound for the optimal blocksizes for SGEQRF/CGEQRF, SGERQF/CGERQF, SORMQR/CUNMQR, and SORMRQ/CUNMRQ.

If LWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SGGHRD/CGGHRD

```
      SUBROUTINE SGGHRD( COMPQ, COMPZ, N, ILO, IHI, A, LDA, B, LDB, Q,
     $                   LDQ, Z, LDZ, INFO )
      CHARACTER          COMPQ, COMPZ
      INTEGER            IHI, ILO, INFO, LDA, LDB, LDQ, LDZ, N
      REAL               A( LDA, * ), B( LDB, * ), Q( LDQ, * ),
     $                   Z( LDZ, * )
```

```
      SUBROUTINE CGGHRD( COMPQ, COMPZ, N, ILO, IHI, A, LDA, B, LDB, Q,
     $                   LDQ, Z, LDZ, INFO )
      CHARACTER          COMPQ, COMPZ
      INTEGER            IHI, ILO, INFO, LDA, LDB, LDQ, LDZ, N
      COMPLEX            A( LDA, * ), B( LDB, * ), Q( LDQ, * ),
     $                   Z( LDZ, * )
```

Purpose

SGGHRD/CGGHRD reduces a pair of real/complex matrices (A,B) to generalized upper Hessenberg form using orthogonal/unitary transformations, where A is a general matrix and B is upper triangular: $Q^H*A*Z = H$ and $Q^H*B*Z = T$, where H is upper Hessenberg, T is upper triangular, and Q and Z are orthogonal/unitary.

The orthogonal/unitary matrices Q and Z are determined as products of Givens rotations. They may either be formed explicitly, or they may be postmultiplied into input matrices Q_1 and Z_1, so that

$$Q_1*A*Z_1^H = (Q_1Q)*H*(Z_1Z)^H$$

$$Q_1*B*Z_1^H = (Q_1Q)*T*(Z_1Z)^H$$

Arguments

COMPQ (input) CHARACTER*1
= 'N': do not compute Q;
= 'I': Q is initialized to the unit matrix, and the orthogonal/unitary matrix Q is returned;
= 'V': Q must contain an orthogonal/unitary matrix Q_1 on entry, and the product Q_1*Q is returned.

COMPZ (input) CHARACTER*1
= 'N': do not compute Z;
= 'I': Z is initialized to the unit matrix, and the orthogonal/unitary matrix Z is returned;
= 'V': Z must contain an orthogonal/unitary matrix Z_1 on entry, and the product Z_1*Z is returned.

N (input) INTEGER
The order of the matrices A and B. $N \geq 0$.

ILO, IHI (input) INTEGER
It is assumed that A is already upper triangular in rows and columns 1:ilo−1 and ihi+1:n. ILO and IHI are normally set by a previous call to SGGBAL/CGGBAL; otherwise they should be set to 1 and N respectively.
$1 \leq ILO \leq IHI \leq N$, if $N > 0$; $ILO = 1$ and $IHI = 0$, if $N = 0$.

A (input/output) REAL/COMPLEX array, dimension (LDA,N)
On entry, the n-by-n general matrix to be reduced.
On exit, the upper triangle and the first subdiagonal of A are overwritten with the upper Hessenberg matrix H, and the rest is set to zero.

LDA (input) INTEGER
The leading dimension of the array A. $LDA \geq \max(1,N)$.

B (input/output) REAL/COMPLEX array, dimension (LDB,N)
On entry, the n-by-n upper triangular matrix B.
On exit, the upper triangular matrix $T = Q^H*B*Z$. The elements below the diagonal are set to zero.

LDB (input) INTEGER
The leading dimension of the array B. $LDB \geq \max(1,N)$.

Q (input/output) REAL/COMPLEX array, dimension (LDQ,N)
If COMPQ = 'N': Q is not referenced.
If COMPQ = 'I': on entry, Q need not be set, and on exit it contains the orthogonal/unitary matrix Q, where Q^H is the product of the Givens transformations which are applied to A and B on the left.
If COMPQ = 'V': on entry, Q must contain an orthogonal/unitary matrix Q_1, and on exit this is overwritten by Q_1*Q.

LDQ (input) INTEGER
The leading dimension of the array Q.
$LDQ \geq \max(1,N)$ if COMPQ = 'V' or 'I'; $LDQ \geq 1$ otherwise.

Z (input/output) REAL/COMPLEX array, dimension (LDZ,N)
If COMPZ = 'N': Z is not referenced.
If COMPZ = 'I': on entry, Z need not be set, and on exit it contains the orthogonal/unitary matrix Z, which is the product of the Givens transformations which are applied to A and B on the right.
If COMPZ = 'V': on entry, Z must contain an orthogonal/unitary matrix Z_1, and on exit this is overwritten by Z_1*Z.

LDZ (input) INTEGER
The leading dimension of the array Z.
$LDZ \geq \max(1,N)$ if COMPZ = 'V' or 'I'; $LDZ \geq 1$ otherwise.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SGGLSE/CGGLSE

```
      SUBROUTINE SGGLSE( M, N, P, A, LDA, B, LDB, C, D, X, WORK, LWORK,
     $                   INFO )
      INTEGER            INFO, LDA, LDB, LWORK, M, N, P
      REAL               A( LDA, * ), B( LDB, * ), C( * ), D( * ),
     $                   WORK( * ), X( * )

      SUBROUTINE CGGLSE( M, N, P, A, LDA, B, LDB, C, D, X, WORK, LWORK,
     $                   INFO )
      INTEGER            INFO, LDA, LDB, LWORK, M, N, P
      COMPLEX            A( LDA, * ), B( LDB, * ), C( * ), D( * ),
     $                   WORK( * ), X( * )
```

Purpose

SGGLSE/CGGLSE solves the linear equality-constrained least squares (LSE) problem:

$$\min \| c - A*x \|_2 \text{ subject to } B*x = d$$

where A is an m-by-n matrix, B is a p-by-n matrix, c is a given m-vector, and d is a given p-vector. It is assumed that $p \le n \le m+p$, and

$$\text{rank}(B) = p, \quad \text{and} \quad \text{rank}\begin{pmatrix} A \\ B \end{pmatrix} = n.$$

These conditions ensure that the LSE problem has a unique solution, which is obtained using a GRQ factorization of the matrices B and A.

Arguments

M (input) INTEGER
The number of rows of the matrix A. $M \ge 0$.

N (input) INTEGER
The number of columns of the matrices A and B. $N \ge 0$.

P (input) INTEGER
The number of rows of the matrix B. $0 \le P \le N \le M+P$.

A (input/output) REAL/COMPLEX array, dimension (LDA,N)
On entry, the m-by-n matrix A.
On exit, A is destroyed.

LDA (input) INTEGER
The leading dimension of the array A. $LDA \ge \max(1,M)$.

B (input/output) REAL/COMPLEX array, dimension (LDB,N)
On entry, the p-by-n matrix B.
On exit, B is destroyed.

LDB (input) INTEGER
The leading dimension of the array B. $LDB \ge \max(1,P)$.

C (input/output) REAL/COMPLEX array, dimension (M)
On entry, C contains the right hand side vector for the least squares part of the LSE problem.
On exit, the residual sum of squares for the solution is given by the sum of squares of elements n−p+1 to m of vector C.

D (input/output) REAL/COMPLEX array, dimension (P)
On entry, D contains the right hand side vector for the constrained equation.
On exit, D is destroyed.

X (output) REAL/COMPLEX array, dimension (N)
On exit, X is the solution of the LSE problem.

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK. $LWORK \ge \max(1,M+N+P)$. For optimum performance $LWORK \ge P+\min(M,N)+\max(M,N)*NB$, where NB is an upper bound for the optimal blocksizes for SGEQRF/CGEQRF, SGERQF/CGERQF, SORMQR/CUNMQR, and SORMRQ/CUNMRQ.

If LWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SGGQRF/CGGQRF

```
      SUBROUTINE SGGQRF( N, M, P, A, LDA, TAUA, B, LDB, TAUB, WORK,
     $                   LWORK, INFO )
      INTEGER            INFO, LDA, LDB, LWORK, M, N, P
      REAL               A( LDA, * ), B( LDB, * ), TAUA( * ), TAUB( * ),
     $                   WORK( * )
      SUBROUTINE CGGQRF( N, M, P, A, LDA, TAUA, B, LDB, TAUB, WORK,
     $                   LWORK, INFO )
      INTEGER            INFO, LDA, LDB, LWORK, M, N, P
      COMPLEX            A( LDA, * ), B( LDB, * ), TAUA( * ), TAUB( * ),
     $                   WORK( * )
```

Purpose

SGGQRF/CGGQRF computes a generalized QR factorization of an n-by-m matrix A and an n-by-p matrix B:

$$A = Q*R, \quad B = Q*T*Z,$$

where Q is an n-by-n orthogonal/unitary matrix, Z is a p-by-p orthogonal/unitary matrix, and R and T assume one of the forms:

$$\text{if} \quad n \ge m, \qquad R = \begin{array}{cc} & m \\ \begin{array}{c} m \\ n-m \end{array} & \begin{pmatrix} R_{11} \\ 0 \end{pmatrix} \end{array}$$

or

$$\text{if} \quad n < m, \qquad R = \begin{array}{cc} & \begin{array}{cc} n & m-n \end{array} \\ n & \begin{pmatrix} R_{11} & R_{12} \end{pmatrix} \end{array}$$

where R_{11} is upper triangular, and

$$\text{if} \quad n \le p, \qquad T = \begin{array}{c} \\ n \end{array} \begin{array}{c} \begin{array}{cc} p-n & n \end{array} \\ \begin{pmatrix} 0 & T_{12} \end{pmatrix} \end{array}$$

or

$$\text{if} \quad n > p, \qquad T = \begin{array}{c} \\ n-p \\ p \end{array} \begin{array}{c} p \\ \begin{pmatrix} T_{11} \\ T_{21} \end{pmatrix} \end{array}$$

where T_{12} or T_{21} is a p-by-p upper triangular matrix.

In particular, if B is square and nonsingular, the GQR factorization of A and B implicitly gives the QR factorization of $B^{-1}*A$:

$$B^{-1}*A = Z^H*(T^{-1}*R).$$

Arguments

N (input) INTEGER
The number of rows of the matrices A and B. $N \ge 0$.

M (input) INTEGER
The number of columns of the matrix A. $M \ge 0$.

P (input) INTEGER
The number of columns of the matrix B. $P \ge 0$.

A (input/output) REAL/COMPLEX array, dimension (LDA,M)
On entry, the n-by-m matrix A.
On exit, the elements on and above the diagonal of the array contain the min(n,m)-by-m upper trapezoidal matrix R (R is upper triangular if $N \ge M$); the elements below the diagonal, with the array TAUA, represent the orthogonal/unitary matrix Q as a product of min(N,M) elementary reflectors.

LDA (input) INTEGER
The leading dimension of the array A. $LDA \ge \max(1,N)$.

TAUA (output) REAL/COMPLEX array, dimension (min(N,M))
The scalar factors of the elementary reflectors which represent the orthogonal/unitary matrix Q.

B (input/output) REAL/COMPLEX array, dimension (LDB,P)
On entry, the n-by-p matrix B.
On exit, if $N \le P$, the upper triangle of the subarray B(1:n,p−n+1:p) contains the n-by-n upper triangular matrix T;
if $N > P$, the elements on and above the $(n-p)^{th}$ subdiagonal contain the n-by-p upper trapezoidal matrix T; the remaining elements, with the array TAUB, represent the orthogonal/unitary matrix Z as a product of elementary reflectors.

LDB (input) INTEGER
The leading dimension of the array B. $LDB \ge \max(1,N)$.

TAUB (output) REAL/COMPLEX array, dimension (min(N,P))
The scalar factors of the elementary reflectors which represent the orthogonal/unitary matrix Z.

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension for the array WORK. $LWORK \ge \max(1,N,M,P)$. For optimum performance $LWORK \ge \max(1,N,M,P)*\max(NB1,NB2,NB3)$, where NB1 is the optimal blocksize for the QR factorization of an n-by-m matrix, NB2 is the optimal blocksize for the RQ factorization of an n-by-p matrix, and NB3 is the optimal blocksize for a call of SORMQR/CUNMQR.

If LWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SGGRQF/CGGRQF

```
      SUBROUTINE SGGRQF( M, P, N, A, LDA, TAUA, B, LDB, TAUB, WORK,
     $                   LWORK, INFO )
      INTEGER            INFO, LDA, LDB, LWORK, M, N, P
      REAL               A( LDA, * ), B( LDB, * ), TAUA( * ), TAUB( * ),
     $                   WORK( * )
      SUBROUTINE CGGRQF( M, P, N, A, LDA, TAUA, B, LDB, TAUB, WORK,
     $                   LWORK, INFO )
      INTEGER            INFO, LDA, LDB, LWORK, M, N, P
      COMPLEX            A( LDA, * ), B( LDB, * ), TAUA( * ), TAUB( * ),
     $                   WORK( * )
```

Purpose

SGGRQF/CGGRQF computes a generalized RQ factorization of an m-by-n matrix A and a p-by-n matrix B:

$$A = R*Q, \; B = Z*T*Q,$$

where Q is an n-by-n orthogonal/unitary matrix, and Z is a p-by-p orthogonal/unitary matrix, and R and T assume one of the forms:

$$\text{if} \quad m \le n, \qquad R = \begin{array}{c} \\ m \end{array} \begin{array}{c} \begin{array}{cc} n-m & m \end{array} \\ \begin{pmatrix} 0 & R_{12} \end{pmatrix} \end{array}$$

or

$$\text{if} \quad m > n, \qquad R = \begin{array}{c} \\ m-n \\ n \end{array} \begin{array}{c} n \\ \begin{pmatrix} R_{11} \\ R_{21} \end{pmatrix} \end{array}$$

where R_{12} or R_{21} is upper triangular, and

$$\text{if} \quad p \geq n, \qquad T = \begin{array}{c} \\ n \\ p-n \end{array} \begin{array}{c} n \\ \begin{pmatrix} T_{11} \\ 0 \end{pmatrix} \end{array}$$

or

$$\text{if} \quad p < n, \qquad T = \begin{array}{c} \\ p \end{array} \begin{array}{c} \begin{array}{cc} p & n-p \end{array} \\ \begin{pmatrix} T_{11} & T_{12} \end{pmatrix} \end{array}$$

where T_{11} is upper triangular.

In particular, if B is square and nonsingular, the GRQ factorization of A and B implicitly gives the RQ factorization of $A*B^{-1}$:

$$A*B^{-1} = (R*T^{-1})*Z^H$$

Arguments

M (input) INTEGER
The number of rows of the matrix A. $M \geq 0$.

P (input) INTEGER
The number of rows of the matrix B. $P \geq 0$.

N (input) INTEGER
The number of columns of the matrices A and B. $N \geq 0$.

A (input/output) REAL/COMPLEX array, dimension (LDA,N)
On entry, the m-by-n matrix A.
On exit, if $M \leq N$, the upper triangle of the subarray A(1:m,n−m+1:n) contains the m-by-m upper triangular matrix R;
if $M > N$, the elements on and above the $(m-n)^{th}$ subdiagonal contain the m-by-n upper trapezoidal matrix R; the remaining elements, with the array TAUA, represent the orthogonal/unitary matrix Q as a product of elementary reflectors.

LDA (input) INTEGER
The leading dimension of the array A. $LDA \geq \max(1,M)$.

TAUA (output) REAL/COMPLEX array, dimension (min(M,N))
The scalar factors of the elementary reflectors which represent the orthogonal/unitary matrix Q.

B (input/output) REAL/COMPLEX array, dimension (LDB,N)
On entry, the p-by-n matrix B.
On exit, the elements on and above the diagonal of the array contain the min(p,n)-by-n upper trapezoidal matrix T (T is upper triangular if $P \geq N$); the elements below the diagonal, with the array TAUB, represent the orthogonal/unitary matrix Z as a product of elementary reflectors.

LDB (input) INTEGER
The leading dimension of the array B. $LDB \geq \max(1,P)$.

TAUB (output) REAL/COMPLEX array, dimension (min(P,N))
The scalar factors of the elementary reflectors which represent the orthogonal/unitary matrix Z.

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK. $LWORK \geq \max(1,N,M,P)$. For optimum performance $LWORK \geq \max(1,N,M,P)*\max(NB1,NB2,NB3)$, where NB1 is the optimal blocksize for the RQ factorization of an m-by-n matrix, NB2 is the optimal blocksize for the QR factorization of the p-by-n matrix, and NB3 is the optimal blocksize for a call of SORMRQ/CUNMRQ.

If LWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SGGSVD/CGGSVD

```
      SUBROUTINE SGGSVD( JOBU, JOBV, JOBQ, M, N, P, K, L, A, LDA, B,
     $                   LDB, ALPHA, BETA, U, LDU, V, LDV, Q, LDQ,
     $                   WORK, IWORK, INFO )
      CHARACTER          JOBQ, JOBU, JOBV
      INTEGER            INFO, K, L, LDA, LDB, LDQ, LDU, LDV, M, N, P
      INTEGER            IWORK( * )
      REAL               A( LDA, * ), ALPHA( * ), B( LDB, * ),
     $                   BETA( * ), Q( LDQ, * ), U( LDU, * ),
     $                   V( LDV, * ), WORK( * )

      SUBROUTINE CGGSVD( JOBU, JOBV, JOBQ, M, N, P, K, L, A, LDA, B,
     $                   LDB, ALPHA, BETA, U, LDU, V, LDV, Q, LDQ,
     $                   WORK, RWORK, IWORK, INFO )
      CHARACTER          JOBQ, JOBU, JOBV
      INTEGER            INFO, K, L, LDA, LDB, LDQ, LDU, LDV, M, N, P
      INTEGER            IWORK( * )
      REAL               ALPHA( * ), BETA( * ), RWORK( * )
      COMPLEX            A( LDA, * ), B( LDB, * ), Q( LDQ, * ),
     $                   U( LDU, * ), V( LDV, * ), WORK( * )
```

Purpose

SGGSVD/CGGSVD computes the generalized singular value decomposition (GSVD) of an m-by-n real/complex matrix A and p-by-n real/complex matrix B:

$$U^H*A*Q = D_1*\begin{pmatrix} 0 & R \end{pmatrix}, \quad V^H*B*Q = D_2*\begin{pmatrix} 0 & R \end{pmatrix}$$

where U, V and Q are orthogonal/unitary matrices. Let k+l = the effective numerical rank of the matrix $(A^H,B^H)^H$, then R is a (k+l)-by-(k+l) nonsingular upper triangular matrix, D_1 and D_2 are m-by-(k+l) and p-by-(k+l) "diagonal" matrices and of the following structures, respectively:

If m−k−l ≥ 0,

$$D_1 = \begin{array}{r} \\ k \\ l \\ m-k-l \end{array}\!\!\begin{array}{c} \begin{array}{cc} k & l \end{array} \\ \begin{pmatrix} I & 0 \\ 0 & C \\ 0 & 0 \end{pmatrix} \end{array}$$

$$D_2 = \begin{array}{r} \\ l \\ p-l \end{array}\!\!\begin{array}{c} \begin{array}{cc} k & l \end{array} \\ \begin{pmatrix} 0 & S \\ 0 & 0 \end{pmatrix} \end{array}$$

$$\begin{pmatrix} 0 & R \end{pmatrix} = \begin{array}{r} \\ k \\ l \end{array}\!\!\begin{array}{c} \begin{array}{ccc} n-k-l & k & l \end{array} \\ \begin{pmatrix} 0 & R_{11} & R_{12} \\ 0 & 0 & R_{22} \end{pmatrix} \end{array}$$

where

C = diag(ALPHA(k+1), ... , ALPHA(k+l)),
S = diag(BETA(k+1), ... , BETA(k+l)),
$C^2 + S^2 = I$;

R is stored in A(1:k+l, n−k−l+1:n) on exit.

If m−k−l < 0,

$$D_1 = \begin{array}{r} \\ k \\ m-k \end{array}\!\!\begin{array}{c} \begin{array}{ccc} k & m-k & k+l-m \end{array} \\ \begin{pmatrix} I & 0 & 0 \\ 0 & C & 0 \end{pmatrix} \end{array}$$

$$D_2 = \begin{array}{r} \\ m-k \\ k+l-m \\ p-l \end{array}\!\!\begin{array}{c} \begin{array}{ccc} k & m-k & k+l-m \end{array} \\ \begin{pmatrix} 0 & S & 0 \\ 0 & 0 & I \\ 0 & 0 & 0 \end{pmatrix} \end{array}$$

$$\begin{pmatrix} 0 & R \end{pmatrix} = \begin{array}{r} \\ k \\ m-k \\ k+l-m \end{array}\!\!\begin{array}{c} \begin{array}{cccc} n-k-l & k & m-k & k+l-m \end{array} \\ \begin{pmatrix} 0 & R_{11} & R_{12} & R_{13} \\ 0 & 0 & R_{22} & R_{23} \\ 0 & 0 & 0 & R_{33} \end{pmatrix} \end{array}$$

where

C = diag(ALPHA(k+1), ... , ALPHA(m)),
S = diag(BETA(k+1), ... , BETA(m)),
$C^2 + S^2 = I$;

$\begin{pmatrix} R_{11} & R_{12} & R_{13} \\ 0 & R_{22} & R_{23} \end{pmatrix}$ is stored in A(1:m, n−k−l+1:n), and R_{33} is stored in B(m−k+1:l, n+m−k−l+1:n) on exit.

The routine computes C, S, R, and optionally the orthogonal/unitary transformation matrices U, V and Q.

In particular, if B is an n-by-n nonsingular matrix, then the GSVD of A and B implicitly gives the SVD of $A*B^{-1}$:

$$A*B^{-1} = U*(D_1*D_2{}^{-1})*V^H.$$

If $(A^H,B^H)^H$ has orthonormal columns, then the GSVD of A and B is also equal to the CS decomposition of A and B. Furthermore, the GSVD can be used to derive the solution of the eigenvalue problem:

$$A^H * A * x = \lambda * B^H * B * x.$$

In some literature, the GSVD of A and B is presented in the form

$$U^H * A * X = \begin{pmatrix} 0 & D_1 \end{pmatrix},$$
$$V^H * B * X = \begin{pmatrix} 0 & D_2 \end{pmatrix}$$

where U and V are orthogonal/unitary and X is nonsingular, D_1 and D_2 are "diagonal". The former GSVD form can be converted to the latter form by taking the nonsingular matrix X as

$$X = Q * \begin{pmatrix} I & 0 \\ 0 & R^{-1} \end{pmatrix}.$$

Arguments

JOBU (input) CHARACTER*1
= 'U': Orthogonal/Unitary matrix U is computed;
= 'N': U is not computed.

JOBV (input) CHARACTER*1
= 'V': Orthogonal/Unitary matrix V is computed;
= 'N': V is not computed.

JOBQ (input) CHARACTER*1
= 'Q': Orthogonal/Unitary matrix Q is computed;
= 'N': Q is not computed.

M (input) INTEGER
The number of rows of the matrix A. M ≥ 0.

N (input) INTEGER
The number of columns of the matrices A and B. $N \geq 0$.

P (input) INTEGER
The number of rows of the matrix B. $P \geq 0$.

K (output) INTEGER

L (output) INTEGER
On exit, K and L specify the dimension of the subblocks.
$K + L$ = effective numerical rank of $(A^H, B^H)^H$.

A (input/output) REAL/COMPLEX array, dimension (LDA,N)
On entry, the m-by-n matrix A.
On exit, A contains the triangular matrix R, or part of R.

LDA (input) INTEGER
The leading dimension of the array A. $LDA \geq \max(1,M)$.

B (input/output) REAL/COMPLEX array, dimension (LDB,N)
On entry, the p-by-n matrix B.
On exit, B contains part of the triangular matrix R if $M-K-L < 0$.

LDB (input) INTEGER
The leading dimension of the array B. $LDA/LDB \geq \max(1,P)$.

ALPHA (output) REAL array, dimension (N)

BETA (output) REAL array, dimension (N)
On exit, ALPHA and BETA contain the generalized singular value pairs of A and B;
ALPHA(1:K) = 1,
BETA(1:K) = 0,
and if $M-K-L \geq 0$,
ALPHA(K+1:K+L) = C,
BETA(K+1:K+L) = S,
or if $M-K-L < 0$,
ALPHA(K+1:M) = C, ALPHA(M+1:K+L) = 0
BETA(K+1:M) = S, BETA(M+1:K+L) = 1
and
ALPHA(K+L+1:N) = 0
BETA(K+L+1:N) = 0

U (output) REAL/COMPLEX array, dimension (LDU,M)
If JOBU = 'U', U contains the m-by-m orthogonal/unitary matrix U.
If JOBU = 'N', U is not referenced.

LDU (input) INTEGER
The leading dimension of the array U.
$LDU \geq \max(1,M)$ if JOBU = 'U'; $LDU \geq 1$ otherwise.

V (output) REAL/COMPLEX array, dimension (LDV,P)
If JOBV = 'V', V contains the p-by-p orthogonal/unitary matrix V.
If JOBV = 'N', V is not referenced.

LDV (input) INTEGER
The leading dimension of the array V.
$LDV \geq \max(1,P)$ if JOBV = 'V'; $LDV \geq 1$ otherwise.

Q (output) REAL/COMPLEX array, dimension (LDQ,N)
If JOBQ = 'Q', Q contains the n-by-n orthogonal/unitary matrix Q.
If JOBQ = 'N', Q is not referenced.

LDQ (input) INTEGER
The leading dimension of the array Q. $LDQ \geq \max(1,N)$ if JOBQ = 'Q'; $LDQ \geq 1$ otherwise.

WORK (workspace) REAL/COMPLEX array,
dimension (max(3*N,M,P)+N)

IWORK (workspace/output) INTEGER array, dimension (N)
On exit, IWORK stores the sorting information. More precisely, the following loop will sort ALPHA

```
for I = K+1, min(M,K+L)
    swap ALPHA(I) and ALPHA(IWORK(I))
endfor
```

such that $ALPHA(1) \geq ALPHA(2) \geq \ldots \geq ALPHA(N)$.

RWORK *CGGSVD only* (workspace) REAL array, dimension (2*N)

INFO (output)INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = 1, the Jacobi-type procedure failed to converge. For further details, see subroutine STGSJA/CTGSJA.

SGGSVP/CGGSVP

```
      SUBROUTINE SGGSVP( JOBU, JOBV, JOBQ, M, P, N, A, LDA, B, LDB,
     $                   TOLA, TOLB, K, L, U, LDU, V, LDV, Q, LDQ,
     $                   IWORK, TAU, WORK, INFO )
      CHARACTER          JOBQ, JOBU, JOBV
      INTEGER            INFO, K, L, LDA, LDB, LDQ, LDU, LDV, M, N, P
      REAL               TOLA, TOLB
      INTEGER            IWORK( * )
      REAL               A( LDA, * ), B( LDB, * ), Q( LDQ, * ),
     $                   TAU( * ), U( LDU, * ), V( LDV, * ), WORK( * )
```

```
      SUBROUTINE CGGSVP( JOBU, JOBV, JOBQ, M, P, N, A, LDA, B, LDB,
     $                   TOLA, TOLB, K, L, U, LDU, V, LDV, Q, LDQ,
     $                   IWORK, RWORK, TAU, WORK, INFO )
      CHARACTER          JOBQ, JOBU, JOBV
      INTEGER            INFO, K, L, LDA, LDB, LDQ, LDU, LDV, M, N, P
      REAL               TOLA, TOLB
      INTEGER            IWORK( * )
      REAL               RWORK( * )
      COMPLEX            A( LDA, * ), B( LDB, * ), Q( LDQ, * ),
     $                   TAU( * ), U( LDU, * ), V( LDV, * ), WORK( * )
```

Purpose

SGGSVP/CGGSVP computes orthogonal/unitary matrices U, V and Q such that

$$U^H * A * Q = \begin{array}{c} \\ k \\ l \\ m-k-l \end{array} \begin{array}{c} \begin{array}{ccc} n-k-l & k & l \end{array} \\ \begin{pmatrix} 0 & A_{12} & A_{13} \\ 0 & 0 & A_{23} \\ 0 & 0 & 0 \end{pmatrix} \end{array} \text{ if } m-k-l \geq 0;$$

$$= \begin{array}{c} \\ k \\ m-k \end{array} \begin{array}{c} \begin{array}{ccc} n-k-l & k & l \end{array} \\ \begin{pmatrix} 0 & A_{12} & A_{13} \\ 0 & 0 & A_{23} \end{pmatrix} \end{array} \text{ if } m-k-l < 0;$$

$$V^H * B * Q = \begin{array}{c} \\ l \\ p-l \end{array} \begin{array}{c} \begin{array}{ccc} n-k-l & k & l \end{array} \\ \begin{pmatrix} 0 & 0 & B_{13} \\ 0 & 0 & 0 \end{pmatrix} \end{array}$$

where the k-by-k matrix A_{12} and l-by-l matrix B_{13} are nonsingular upper triangular; A_{23} is l-by-l upper triangular if $m-k-l \geq 0$, otherwise A_{23} is (m−k)-by-l upper trapezoidal. k+l = the effective numerical rank of the (m+p)-by-n matrix $(A^H, B^H)^H$.

This decomposition is the preprocessing step for computing the Generalized Singular Value Decomposition (GSVD), see subroutine SGGSVD/CGGSVD.

Arguments

JOBU (input) CHARACTER*1
= 'U': Orthogonal/Unitary matrix U is computed;
= 'N': U is not computed.

JOBV (input) CHARACTER*1
= 'V': Orthogonal/Unitary matrix V is computed;
= 'N': V is not computed.

JOBQ (input) CHARACTER*1
= 'Q': Orthogonal/Unitary matrix Q is computed;
= 'N': Q is not computed.

M (input) INTEGER
The number of rows of the matrix A. $M \geq 0$.

P (input) INTEGER
The number of rows of the matrix B. $P \geq 0$.

N (input) INTEGER
The number of columns of the matrices A and B. $N \geq 0$.

A (input/output) REAL/COMPLEX array, dimension (LDA,N)
On entry, the m-by-n matrix A.
On exit, A contains the triangular (or trapezoidal) matrix described in the Purpose section.

LDA (input) INTEGER
The leading dimension of the array A. $LDA \geq \max(1,M)$.

B (input/output) REAL/COMPLEX array, dimension (LDB,N)
On entry, the p-by-n matrix B.
On exit, B contains the triangular matrix described in the Purpose section.

LDB (input) INTEGER
The leading dimension of the array B. $LDB \geq \max(1,P)$.

TOLA (input) REAL

TOLB (input) REAL
TOLA and TOLB are the thresholds to determine the effective numerical rank of matrix B and a subblock of A. Generally, they are set to
TOLA = max(M,N)*$||A||$*MACHEPS,
TOLB = max(P,N)*$||B||$*MACHEPS.
The size of TOLA and TOLB may affect the size of backward errors of the decomposition.

K, L (output) INTEGER
On exit, K and L specify the dimension of the subblocks.
K+L = effective numerical rank of $(A^H, B^H)^H$.

U (output) REAL/COMPLEX array, dimension (LDU,M)
If JOBU = 'U', U contains the orthogonal/unitary matrix U.
If JOBU = 'N', U is not referenced.

LDU (input) INTEGER
The leading dimension of the array U.
$LDU \geq \max(1,M)$ if JOBU = 'U'; $LDU \geq 1$ otherwise.

V (output) REAL/COMPLEX array, dimension (LDV,M)
If JOBV = 'V', V contains the orthogonal/unitary matrix V.
If JOBV = 'N', V is not referenced.

LDV (input) INTEGER
The leading dimension of the array V.
$LDV \geq \max(1,P)$ if JOBV = 'V'; $LDV \geq 1$ otherwise.

Q (output) REAL/COMPLEX array, dimension (LDQ,N)
If JOBQ = 'Q', Q contains the orthogonal/unitary matrix Q.
If JOBQ = 'N', Q is not referenced.

LDQ (input) INTEGER
The leading dimension of the array Q. LDQ $\geq$ max(1,N). LDQ $\geq$ max(1,N) if JOBQ = 'Q'; LDQ $\geq$ 1 otherwise.

IWORK (workspace) INTEGER array, dimension (N)

RWORK *CGGSVP only* (workspace) REAL array, dimension (2*N)

TAU (workspace) REAL/COMPLEX array, dimension (N)

WORK (workspace) REAL/COMPLEX array, dimension (max(3*N,M,P))

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = $-$i, the ith argument had an illegal value.

SGTCON/CGTCON

```
      SUBROUTINE SGTCON( NORM, N, DL, D, DU, DU2, IPIV, ANORM, RCOND,
     $                   WORK, IWORK, INFO )
      CHARACTER          NORM
      INTEGER            INFO, N
      REAL               ANORM, RCOND
      INTEGER            IPIV( * ), IWORK( * )
      REAL               D( * ), DL( * ), DU( * ), DU2( * ), WORK( * )

      SUBROUTINE CGTCON( NORM, N, DL, D, DU, DU2, IPIV, ANORM, RCOND,
     $                   WORK, INFO )
      CHARACTER          NORM
      INTEGER            INFO, N
      REAL               ANORM, RCOND
      INTEGER            IPIV( * )
      COMPLEX            D( * ), DL( * ), DU( * ), DU2( * ), WORK( * )
```

Purpose

SGTCON/CGTCON estimates the reciprocal of the condition number of a real/complex tridiagonal matrix A using the LU factorization as computed by SGTTRF/CGTTRF.

An estimate is obtained for $\|A^{-1}\|$, and the reciprocal of the condition number is computed as RCOND = $1/(\|A\| * \|A^{-1}\|)$.

Arguments

NORM (input) CHARACTER*1
Specifies whether the 1-norm condition number or the infinity-norm condition number is required:
= '1' or 'O': 1-norm;
= 'I': Infinity-norm.

N (input) INTEGER
The order of the matrix A. N $\geq$ 0.

DL (input) REAL/COMPLEX array, dimension (N$-$1)
The (n$-$1) multipliers that define the matrix L from the LU factorization of A as computed by SGTTRF/CGTTRF.

D (input) REAL/COMPLEX array, dimension (N)
The n diagonal elements of the upper triangular matrix U from the LU factorization of A.

DU (input) REAL/COMPLEX array, dimension (N$-$1)
The (n$-$1) elements of the first superdiagonal of U.

DU2 (input) REAL/COMPLEX array, dimension (N$-$2)
The (n$-$2) elements of the second superdiagonal of U.

IPIV (input) INTEGER array, dimension (N)
The pivot indices; for 1 $\leq$ i $\leq$ n, row i of the matrix was interchanged with row IPIV(i). IPIV(i) will always be either i or i+1; IPIV(i) = i indicates a row interchange was not required.

ANORM (input) REAL
If NORM = '1' or 'O', the 1-norm of the original matrix A.
If NORM = 'I', the infinity-norm of the original matrix A.

RCOND (output) REAL
The reciprocal of the condition number of the matrix A, computed as RCOND = $1/(\|A\| * \|A^{-1}\|)$.

WORK (workspace) REAL/COMPLEX array, dimension (2*N)

IWORK *SGTCON only* (workspace) INTEGER array, dimension (N)

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = $-$i, the ith argument had an illegal value.

SGTRFS/CGTRFS

```
      SUBROUTINE SGTRFS( TRANS, N, NRHS, DL, D, DU, DLF, DF, DUF, DU2,
     $                   IPIV, B, LDB, X, LDX, FERR, BERR, WORK, IWORK,
     $                   INFO )
      CHARACTER          TRANS
      INTEGER            INFO, LDB, LDX, N, NRHS
      INTEGER            IPIV( * ), IWORK( * )
      REAL               B( LDB, * ), BERR( * ), D( * ), DF( * ),
     $                   DL( * ), DLF( * ), DU( * ), DU2( * ), DUF( * ),
     $                   FERR( * ), WORK( * ), X( LDX, * )
```

```
      SUBROUTINE CGTRFS( TRANS, N, NRHS, DL, D, DU, DLF, DF, DUF, DU2,
     $                   IPIV, B, LDB, X, LDX, FERR, BERR, WORK, RWORK,
     $                   INFO )
      CHARACTER          TRANS
      INTEGER            INFO, LDB, LDX, N, NRHS
      INTEGER            IPIV( * )
      REAL               BERR( * ), FERR( * ), RWORK( * )
      COMPLEX            B( LDB, * ), D( * ), DF( * ), DL( * ),
     $                   DLF( * ), DU( * ), DU2( * ), DUF( * ),
     $                   WORK( * ), X( LDX, * )
```

Purpose

SGTRFS/CGTRFS improves the computed solution to a system of linear equations when the coefficient matrix is tridiagonal, and provides error bounds and backward error estimates for the solution.

Arguments

TRANS (input) CHARACTER*1
Specifies the form of the system of equations:
= 'N': A*X = B (No transpose)
= 'T': A^T*X = B (Transpose)
= 'C': A^H*X = B (Conjugate transpose)

N (input) INTEGER
The order of the matrix A. N $\geq$ 0.

NRHS (input) INTEGER
The number of right hand sides, i.e., the number of columns of the matrix B. NRHS $\geq$ 0.

DL (input) REAL/COMPLEX array, dimension (N−1)
The (n−1) subdiagonal elements of A.

D (input) REAL/COMPLEX array, dimension (N)
The diagonal elements of A.

DU (input) REAL/COMPLEX array, dimension (N−1)
The (n−1) superdiagonal elements of A.

DLF (input) REAL/COMPLEX array, dimension (N−1)
The (n−1) multipliers that define the matrix L from the LU factorization of A as computed by SGTTRF/CGTTRF.

DF (input) REAL/COMPLEX array, dimension (N)
The n diagonal elements of the upper triangular matrix U from the LU factorization of A.

DUF (input) REAL/COMPLEX array, dimension (N−1)
The (n−1) elements of the first superdiagonal of U.

DU2 (input) REAL/COMPLEX array, dimension (N−2)
The (n−2) elements of the second superdiagonal of U.

IPIV (input) INTEGER array, dimension (N)
The pivot indices; for 1 $\leq$ i $\leq$ n, row i of the matrix was interchanged with row IPIV(i). IPIV(i) will always be either i or i+1; IPIV(i) = i indicates a row interchange was not required.

B (input) REAL/COMPLEX array, dimension (LDB,NRHS)
The right hand side matrix B.

LDB (input) INTEGER
The leading dimension of the array B. LDB $\geq$ max(1,N).

X (input/output) REAL/COMPLEX array, dimension (LDX,NRHS)
On entry, the solution matrix X, as computed by SGTTRS/CGTTRS.
On exit, the improved solution matrix X.

LDX (input) INTEGER
The leading dimension of the array X. LDX $\geq$ max(1,N).

FERR (output) REAL array, dimension (NRHS)
The estimated forward error bound for each solution vector X(j) (the jth column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) − XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR (output) REAL array, dimension (NRHS)
The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK *SGTRFS* (workspace) REAL array, dimension (3*N)
CGTRFS (workspace) COMPLEX array, dimension (2*N)

IWORK *SGTRFS only* (workspace) INTEGER array, dimension (N)

RWORK *CGTRFS only* (workspace) REAL array, dimension (N)

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SGTSV/CGTSV

```
      SUBROUTINE SGTSV( N, NRHS, DL, D, DU, B, LDB, INFO )
      INTEGER            INFO, LDB, N, NRHS
      REAL               B( LDB, * ), D( * ), DL( * ), DU( * )

      SUBROUTINE CGTSV( N, NRHS, DL, D, DU, B, LDB, INFO )
      INTEGER            INFO, LDB, N, NRHS
      COMPLEX            B( LDB, * ), D( * ), DL( * ), DU( * )
```

Purpose

SGTSV/CGTSV solves the equation A*X = B, where A is an n-by-n tridiagonal matrix, by Gaussian elimination with partial pivoting.

Note that the equation A^T*X = B may be solved by interchanging the order of the arguments DU and DL.

Arguments

N (input) INTEGER
The order of the matrix A. N $\geq$ 0.

NRHS (input) INTEGER
The number of right hand sides, i.e., the number of columns of the matrix B. NRHS $\geq$ 0.

DL (input/output) REAL/COMPLEX array, dimension (N−1)
On entry, DL must contain the (n−1) subdiagonal elements of A.
On exit, DL is overwritten by the (n−2) elements of the second superdiagonal of the upper triangular matrix U from the LU factorization of A, in DL(1), ..., DL(n−2).

D (input/output) REAL/COMPLEX array, dimension (N)
On entry, D must contain the diagonal elements of A.
On exit, D is overwritten by the n diagonal elements of U.

DU (input/output) REAL/COMPLEX array, dimension (N−1)
On entry, DU must contain the (n−1) superdiagonal elements of A.
On exit, DU is overwritten by the (n−1) elements of the first superdiagonal of U.

B (input/output) REAL/COMPLEX array, dimension (LDB,NRHS)
On entry, the n-by-nrhs right hand side matrix B.
On exit, if INFO = 0, the n-by-nrhs solution matrix X.

LDB (input) INTEGER
The leading dimension of the array B. LDB $\geq$ max(1,N).

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, U(i,i) is exactly zero, and the solution has not been computed. The factorization has not been completed unless i = N.

SGTSVX/CGTSVX

```
      SUBROUTINE SGTSVX( FACT, TRANS, N, NRHS, DL, D, DU, DLF, DF, DUF,
     $                   DU2, IPIV, B, LDB, X, LDX, RCOND, FERR, BERR,
     $                   WORK, IWORK, INFO )
      CHARACTER          FACT, TRANS
      INTEGER            INFO, LDB, LDX, N, NRHS
      REAL               RCOND
      INTEGER            IPIV( * ), IWORK( * )
      REAL               B( LDB, * ), BERR( * ), D( * ), DF( * ),
     $                   DL( * ), DLF( * ), DU( * ), DU2( * ), DUF( * ),
     $                   FERR( * ), WORK( * ), X( LDX, * )

      SUBROUTINE CGTSVX( FACT, TRANS, N, NRHS, DL, D, DU, DLF, DF, DUF,
     $                   DU2, IPIV, B, LDB, X, LDX, RCOND, FERR, BERR,
     $                   WORK, RWORK, INFO )
      CHARACTER          FACT, TRANS
      INTEGER            INFO, LDB, LDX, N, NRHS
      REAL               RCOND
      INTEGER            IPIV( * )
      REAL               BERR( * ), FERR( * ), RWORK( * )
      COMPLEX            B( LDB, * ), D( * ), DF( * ), DL( * ),
     $                   DLF( * ), DU( * ), DU2( * ), DUF( * ),
     $                   WORK( * ), X( LDX, * )
```

Purpose

SGTSVX/CGTSVX uses the LU factorization to compute the solution to a real/complex system of linear equations A*X = B, A^T*X = B, or A^H*X = B, where A is a tridiagonal matrix of order n and X and B are n-by-nrhs matrices.

Error bounds on the solution and a condition estimate are also provided.

Description

The following steps are performed:

1. If FACT = 'N', the LU decomposition is used to factor the matrix A as A = L*U, where L is a product of permutation and unit lower bidiagonal matrices and U is upper triangular with nonzeros in only the main diagonal and first two superdiagonals.
2. If some U(i,i)=0, so that U is exactly singular, then the routine returns with INFO = i. Otherwise, the factored form of A is used to estimate the condition number of the matrix A. If the reciprocal of the condition number is less than machine precision, INFO = N+1 is returned as a warning, but the routine still goes on to solve for X and compute error bounds as described below.
3. The system of equations is solved for X using the factored form of A.
4. Iterative refinement is applied to improve the computed solution matrix and calculate error bounds and backward error estimates for it.

Arguments

FACT (input) CHARACTER*1
Specifies whether or not the factored form of A has been supplied on entry.
= 'F': DLF, DF, DUF, DU2, and IPIV contain the factored form of A; DL, D, DU, DLF, DF, DUF, DU2 and IPIV will not be modified.
= 'N': The matrix will be copied to DLF, DF, and DUF and factored.

TRANS (input) CHARACTER*1
Specifies the form of the system of equations:
= 'N': A*X = B (No transpose)
= 'T': A^T*X = B (Transpose)
= 'C': A^H*X = B (Conjugate transpose)

N (input) INTEGER
The order of the matrix A. N $\geq$ 0.

NRHS (input) INTEGER
The number of right hand sides, i.e., the number of columns of the matrix B. NRHS $\geq$ 0.

DL (input) REAL/COMPLEX array, dimension (N−1)
The (n−1) subdiagonal elements of A.

D (input) REAL/COMPLEX array, dimension (N)
The n diagonal elements of A.

DU (input) REAL/COMPLEX array, dimension (N−1)
The (n−1) superdiagonal elements of A.

DLF (input or output) REAL/COMPLEX array, dimension (N−1)
If FACT = 'F', then DLF is an input argument and on entry contains the (n−1) multipliers that define the matrix L from the LU factorization of A as computed by SGTTRF/CGTTRF.
If FACT = 'N', then DLF is an output argument and on exit contains the (n−1) multipliers that define the matrix L from the LU factorization of A.

DF (input or output) REAL/COMPLEX array, dimension (N)
If FACT = 'F', then DF is an input argument and on entry contains the n diagonal elements of the upper triangular matrix U from the LU factorization of A.
If FACT = 'N', then DF is an output argument and on exit contains the n diagonal elements of the upper triangular matrix U from the LU factorization of A.

DUF (input or output) REAL/COMPLEX array, dimension (N−1)
If FACT = 'F', then DUF is an input argument and on entry contains the (n−1) elements of the first superdiagonal of U.
If FACT = 'N', then DUF is an output argument and on exit contains the (n−1) elements of the first superdiagonal of U.

DU2 (input or output) REAL/COMPLEX array, dimension (N−2)
If FACT = 'F', then DU2 is an input argument and on entry contains the (n−2) elements of the second superdiagonal of U.
If FACT = 'N', then DU2 is an output argument and on exit contains the (n−2) elements of the second superdiagonal of U.

IPIV (input or output) INTEGER array, dimension (N)
If FACT = 'F', then IPIV is an input argument and on entry contains the pivot indices from the LU factorization of A as computed by SGTTRF/CGTTRF.
If FACT = 'N', then IPIV is an output argument and on exit contains the pivot indices from the LU factorization of A; row i of the matrix was interchanged with row IPIV(i). IPIV(i) will always be either i or i+1; IPIV(i) = i indicates a row interchange was not required.

B (input) REAL/COMPLEX array, dimension (LDB,NRHS)
The n-by-nrhs right hand side matrix B.

LDB (input) INTEGER
The leading dimension of the array B. LDB $\geq$ max(1,N).

X (output) REAL/COMPLEX array, dimension (LDX,NRHS)
If INFO = 0 or INFO = N+1, the N-by-NRHS solution matrix X.

LDX (input) INTEGER
The leading dimension of the array X. LDX $\geq$ max(1,N).

RCOND (output) REAL
The estimate of the reciprocal condition number of the matrix A. If RCOND is less than the machine precision (in particular, if RCOND = 0), the matrix is singular to working precision. This condition is indicated by a return code of INFO > 0.

FERR (output) REAL array, dimension (NRHS)
The estimated forward error bound for each solution vector X(j) (the jth column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) − XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR (output) REAL array, dimension (NRHS)
The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK *SGTSVX* (workspace) REAL array, dimension (3*N)
CGTSVX (workspace) COMPLEX array, dimension (2*N)

IWORK *SGTSVX only* (workspace) INTEGER array, dimension (N)

RWORK *CGTSVX only* (workspace) REAL array, dimension (N)

INFO (output) INTEGER

= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, and i is
≤ N: U(i,i) is exactly zero. The factorization has not been completed unless i = N, but the factor U is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned.
= N+1: U is nonsingular, but RCOND is less than machine precision, meaning that the matrix is singular to working precision. Nevertheless, the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the value of RCOND would suggest.

SGTTRF/CGTTRF

```
SUBROUTINE SGTTRF( N, DL, D, DU, DU2, IPIV, INFO )
INTEGER            INFO, N
INTEGER            IPIV( * )
REAL               D( * ), DL( * ), DU( * ), DU2( * )

SUBROUTINE CGTTRF( N, DL, D, DU, DU2, IPIV, INFO )
INTEGER            INFO, N
INTEGER            IPIV( * )
COMPLEX            D( * ), DL( * ), DU( * ), DU2( * )
```

Purpose

SGTTRF/CGTTRF computes an LU factorization of a real/complex tridiagonal matrix A using elimination with partial pivoting and row interchanges.

The factorization has the form A = L*U where L is a product of permutation and unit lower bidiagonal matrices and U is upper triangular with nonzeros in only the main diagonal and first two superdiagonals.

Arguments

N (input) INTEGER
The order of the matrix A. N ≥ 0.

DL (input/output) REAL/COMPLEX array, dimension (N−1)
On entry, DL must contain the (n−1) subdiagonal elements of A.
On exit, DL is overwritten by the (n−1) multipliers that define the matrix L from the LU factorization of A.

D (input/output) REAL/COMPLEX array, dimension (N)
On entry, D must contain the diagonal elements of A.
On exit, D is overwritten by the n diagonal elements of the upper triangular matrix U from the LU factorization of A.

DU (input/output) REAL/COMPLEX array, dimension (N−1)
On entry, DU must contain the (n−1) superdiagonal elements of A.
On exit, DU is overwritten by the (n−1) elements of the first superdiagonal of U.

DU2 (output) REAL/COMPLEX array, dimension (N−2)
On exit, DU2 is overwritten by the (n−2) elements of the second superdiagonal of U.

IPIV (output) INTEGER array, dimension (N)
The pivot indices; for 1 ≤ i ≤ n, row i of the matrix was interchanged with row IPIV(i). IPIV(i) will always be either i or i+1; IPIV(i) = i indicates a row interchange was not required.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, and division by zero will occur if it is used to solve a system of equations.

SGTTRS/CGTTRS

```
SUBROUTINE SGTTRS( TRANS, N, NRHS, DL, D, DU, DU2, IPIV, B, LDB,
$                  INFO )
CHARACTER          TRANS
INTEGER            INFO, LDB, N, NRHS
INTEGER            IPIV( * )
REAL               B( LDB, * ), D( * ), DL( * ), DU( * ), DU2( * )

SUBROUTINE CGTTRS( TRANS, N, NRHS, DL, D, DU, DU2, IPIV, B, LDB,
$                  INFO )
CHARACTER          TRANS
INTEGER            INFO, LDB, N, NRHS
INTEGER            IPIV( * )
COMPLEX            B( LDB, * ), D( * ), DL( * ), DU( * ), DU2( * )
```

Purpose

SGTTRS/CGTTRS solves one of the systems of equations A*X = B, A^T*X = B, or A^H*X = B, with a tridiagonal matrix A using the LU factorization computed by SGTTRF/CGTTRF

Arguments

TRANS (input) CHARACTER
Specifies the form of the system of equations:
= 'N': A*X = B (No transpose)
= 'T': A^T*X = B (Transpose)

= 'C': $A^H*X = B$ (Conjugate transpose)

N (input) INTEGER
The order of the matrix A. $N \geq 0$.

NRHS (input) INTEGER
The number of right hand sides, i.e., the number of columns of the matrix B. $NRHS \geq 0$.

DL (input) REAL/COMPLEX array, dimension (N−1)
The (n−1) multipliers that define the matrix L from the LU factorization of A.

D (input) REAL/COMPLEX array, dimension (N)
The n diagonal elements of the upper triangular matrix U from the LU factorization of A.

DU (input) REAL/COMPLEX array, dimension (N−1)
The (n−1) elements of the first superdiagonal of U.

DU2 (input) REAL/COMPLEX array, dimension (N−2)
The (n−2) elements of the second superdiagonal of U.

IPIV (input) INTEGER array, dimension (N)
The pivot indices; for $1 \leq i \leq n$, row i of the matrix was interchanged with row IPIV(i). IPIV(i) will always be either i or i+1; IPIV(i) = i indicates a row interchange was not required.

B (input/output) REAL/COMPLEX array, dimension (LDB,NRHS)
On entry, the right hand side matrix B.
On exit, B is overwritten by the solution matrix X.

LDB (input) INTEGER
The leading dimension of the array B. $LDB \geq \max(1,N)$.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SHGEQZ/CHGEQZ

```
      SUBROUTINE SHGEQZ( JOB, COMPQ, COMPZ, N, ILO, IHI, A, LDA, B, LDB,
     $                   ALPHAR, ALPHAI, BETA, Q, LDQ, Z, LDZ, WORK,
     $                   LWORK, INFO )
      CHARACTER          COMPQ, COMPZ, JOB
      INTEGER            IHI, ILO, INFO, LDA, LDB, LDQ, LDZ, LWORK, N
      REAL               A( LDA, * ), ALPHAI( * ), ALPHAR( * ),
     $                   B( LDB, * ), BETA( * ), Q( LDQ, * ), WORK( * ),
     $                   Z( LDZ, * )
      SUBROUTINE CHGEQZ( JOB, COMPQ, COMPZ, N, ILO, IHI, A, LDA, B, LDB,
     $                   ALPHA, BETA, Q, LDQ, Z, LDZ, WORK, LWORK,
     $                   RWORK, INFO )
      CHARACTER          COMPQ, COMPZ, JOB
      INTEGER            IHI, ILO, INFO, LDA, LDB, LDQ, LDZ, LWORK, N
      REAL               RWORK( * )
      COMPLEX            A( LDA, * ), ALPHA( * ), B( LDB, * ),
     $                   BETA( * ), Q( LDQ, * ), WORK( * ), Z( LDZ, * )
```

Purpose

SHGEQZ/CHGEQZ implements a single-/double-shift version (*SHGEQZ*) or single-shift version (*CHGEQZ*) of the QZ method for finding the generalized eigenvalues w(j) = (ALPHAR(j) + i*ALPHAI(j))/BETAR(j) (*SHGEQZ*) or w(i) = ALPHA(i)/BETA(i) (*CHGEQZ*) of the equation

det(A − w(i)*B) = 0

SHGEQZ only
In addition, the pair A,B may be reduced to generalized Schur form: B is upper triangular, and A is block upper triangular, where the diagonal blocks are either 1-by-1 or 2-by-2, the 2-by-2 blocks having complex generalized eigenvalues (see the description of the argument JOB.)

If JOB='S', then the pair (A,B) is simultaneously reduced to Schur form by applying one orthogonal tranformation (usually called Q) on the left and another (usually called Z) on the right. The 2-by-2 upper-triangular diagonal blocks of B corresponding to 2-by-2 blocks of A will be reduced to positive diagonal matrices. (I.e., if A(j+1,j) is non-zero, then B(j+1,j) = B(j,j+1) = 0 and B(j,j) and B(j+1,j+1) will be positive.)

If JOB='E', then at each iteration, the same transformations are computed, but they are only applied to those parts of A and B which are needed to compute ALPHAR, ALPHAI, and BETAR.

CHGEQZ only
If JOB='S', then the pair (A,B) is simultaneously reduced to Schur form (i.e., A and B are both upper triangular) by applying one unitary tranformation (usually called Q) on the left and another (usually called Z) on the right. The diagonal elements of A are then ALPHA(1),...,ALPHA(n), and of B are BETA(1),...,BETA(n).

SHGEQZ and CHGEQZ
If JOB='S' and COMPQ and COMPZ are 'V' or 'I', then the orthogonal/unitary transformations used to reduce (A,B) are accumulated into the arrays Q and Z such that:

$$Q(in)*A(in)*Z(in)^H = Q(out)*A(out)*Z(out)^H$$

$$Q(in)*B(in)*Z(in)^H = Q(out)*B(out)*Z(out)^H$$

Arguments

JOB (input) CHARACTER*1
= 'E': compute only ALPHAR, ALPHAI (*SHGEQZ*) or ALPHA (*CHGEQZ*) and BETA. A and B will not necessarily be put into generalized Schur form.
= 'S': put A and B into generalized Schur form, as well as computing ALPHAR, ALPHAI (*SHGEQZ*) or ALPHA (*CHGEQZ*), and BETA.

COMPQ (input) CHARACTER*1
= 'N': do not modify Q.
= 'V': multiply the array Q on the right by the transpose/conjugate-transpose of the orthogonal/unitary tranformation that is aplied to the left side of A and B to reduce them to Schur form.
= 'I': like COMPQ='V', except that Q will be initialized to the identity first.

COMPZ (input) CHARACTER*1
= 'N': do not modify Z.
= 'V': multiply the array Z on the right by the orthogonal/unitary tranformation that is applied to the right side of A and B to reduce them to Schur form.
= 'I': like COMPZ='V', except that Z will be initialized to the identity first.

N (input) INTEGER
The order of the matrices A, B, Q, and Z. N $\geq$ 0

ILO, IHI (input) INTEGER
It is assumed that A is already upper triangular in rows and columns 1:ilo−1 and ihi+1:n.
1 $\leq$ ILO $\leq$ IHI $\leq$ N, if N > 0; ILO = 1 and IHI = 0, if N = 0.

A (input/output) REAL/COMPLEX array, dimension (LDA,N)
On entry, the n-by-n upper Hessenberg matrix A. Elements below the subdiagonal must be zero.
If JOB='S', then on exit A and B will have been simultaneously reduced to generalized Schur form.
If JOB='E', then on exit A will have been destroyed.
SHGEQZ
The diagonal blocks will be correct, but the off-diagonal portion will be meaningless.

LDA (input) INTEGER
The leading dimension of the array A. LDA $\geq$ max(1,N).

B (input/output) REAL/COMPLEX array, dimension (LDB,N)
On entry, the n-by-n upper triangular matrix B. Elements below the diagonal must be zero.
SHGEQZ
2-by-2 blocks in B corresponding to 2-by-2 blocks in A will be reduced to positive diagonal form. (I.e., if A(j+1,j) is non-zero, then B(j+1,j) = B(j,j+1) = 0 and B(j,j) and B(j+1,j+1) will be positive.)
SHGEQZ and CHGEQZ
If JOB='S', then on exit A and B will have been simultaneously reduced to generalized Schur form.
If JOB='E', then on exit B will have been destroyed.
SHGEQZ
Elements corresponding to diagonal blocks of A will be correct, but the off-diagonal portion will be meaningless.

LDB (input) INTEGER
The leading dimension of the array B. LDB $\geq$ max(1,N)

ALPHAR *SHGEQZ* (output) REAL array, dimension (N)
ALPHAR(1:n) will be set to real parts of the diagonal elements of A that would result from reducing A and B to Schur form and then further reducing them both to triangular form using unitary transformations s.t. the diagonal of B was non-negative real. Thus, if A(j,j) is in a 1-by-1 block (i.e., A(j+1,j)=A(j,j+1)=0), then ALPHAR(j)=A(j,j).
Note that the (real or complex) values
(ALPHAR(j) + i*ALPHAI(j))/BETA(j), j=1,...,n, are the generalized eigenvalues of the matrix pencil A − w*B.

ALPHAI *SHGEQZ* (output) REAL array, dimension (N)
ALPHAI(1:n) will be set to imaginary parts of the diagonal elements of A that would result from reducing A and B to Schur form and then further reducing them both to triangular form using unitary transformations s.t. the diagonal of B was non-negative real. Thus, if A(j,j) is in a 1-by-1 block (i.e., A(j+1,j)=A(j,j+1)=0), then ALPHAR(j)=0.
Note that the (real or complex) values
(ALPHAR(j) + i*ALPHAI(j))/BETA(j), j=1,...,n, are the generalized eigenvalues of the matrix pencil A − w*B.

ALPHA *CHGEQZ* (output) COMPLEX array, dimension (N)
The diagonal elements of A when the pair (A,B) has been reduced to Schur form. ALPHA(i)/BETA(i) i=1,...,n are the generalized eigenvalues.

BETA (output) REAL/COMPLEX array, dimension (N)
SHGEQZ
BETA(1:n) will be set to the (real) diagonal elements of B that would result from reducing A and B to Schur form and then further reducing them both to triangular form using unitary transformations s.t. the diagonal of B was non-negative real. Thus, if A(j,j) is in a 1-by-1 block (i.e., A(j+1,j)=A(j,j+1)=0), then BETA(j)=B(j,j). Note that the (real or complex) values (ALPHAR(j) + i*ALPHAI(j))/BETA(j), j=1,...,n, are the generalized eigenvalues of the matrix pencil A − w*B. (Note that BETA(1:n) will always be non-negative, and no BETAI is necessary.)
CHGEQZ
The diagonal elements of B when the pair (A,B) has been reduced to

Schur form. ALPHA(i)/BETA(i) i=1,...,n are the generalized eigenvalues. A and B are normalized so that BETA(1),...,BETA(n) are non-negative real numbers.

Q (input/output) REAL/COMPLEX array, dimension (LDQ,N)
If COMPQ='N', then Q will not be referenced.
If COMPQ='V' or 'I', then the transpose/conjugate-transpose of the orthogonal/unitary transformations which are applied to A and B on the left will be applied to the array Q on the right.

LDQ (input) INTEGER
The leading dimension of the array Q. LDQ $\geq$ 1.
If COMPQ='V' or 'I', then LDQ $\geq$ N.

Z (input/output) REAL/COMPLEX array, dimension (LDZ,N)
If COMPZ='N', then Z will not be referenced.
If COMPZ='V' or 'I', then the orthogonal/unitary transformations which are applied to A and B on the right will be applied to the array Z on the right.

LDZ (input) INTEGER
The leading dimension of the array Z. LDZ $\geq$ 1.
If COMPZ='V' or 'I', then LDZ $\geq$ N.

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if INFO $\geq$ 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK. LWORK $\geq$ max(1,N).

If LWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK *CHGEQZ only* (workspace) REAL array, dimension (N)

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
= 1,...,N: the QZ iteration did not converge. (A,B) is not in Schur form, but ALPHAR(i), ALPHAI(i) (*SHGEQZ*) or ALPHA(i) (*CHGEQZ*) , and BETA(i), i=info+1,...,n should be correct.
= N+1,...,2*N: the shift calculation failed. (A,B) is not in Schur form, but ALPHAR(i), ALPHAI(i) (*SHGEQZ*), or ALPHA(i) (*CHGEQZ*) and BETA(i), i=info−n+1,...,N should be correct.
> 2*N: various "impossible" errors.

SHSEIN/CHSEIN

```
 SUBROUTINE SHSEIN( SIDE, EIGSRC, INITV, SELECT, N, H, LDH, WR, WI,
$                   VL, LDVL, VR, LDVR, MM, M, WORK, IFAILL,
$                   IFAILR, INFO )
 CHARACTER          EIGSRC, INITV, SIDE
 INTEGER            INFO, LDH, LDVL, LDVR, M, MM, N
 LOGICAL            SELECT( * )
 INTEGER            IFAILL( * ), IFAILR( * )
 REAL               H( LDH, * ), VL( LDVL, * ), VR( LDVR, * ),
$                   WI( * ), WORK( * ), WR( * )

 SUBROUTINE CHSEIN( SIDE, EIGSRC, INITV, SELECT, N, H, LDH, W, VL,
$                   LDVL, VR, LDVR, MM, M, WORK, RWORK, IFAILL,
$                   IFAILR, INFO )
 CHARACTER          EIGSRC, INITV, SIDE
 INTEGER            INFO, LDH, LDVL, LDVR, M, MM, N
 LOGICAL            SELECT( * )
 INTEGER            IFAILL( * ), IFAILR( * )
 REAL               RWORK( * )
 COMPLEX            H( LDH, * ), VL( LDVL, * ), VR( LDVR, * ),
$                   W( * ), WORK( * )
```

Purpose

SHSEIN/CHSEIN uses inverse iteration to find specified right and/or left eigenvectors of a real/complex upper Hessenberg matrix H.

The right eigenvector x and the left eigenvector y of the matrix H corresponding to an eigenvalue w are defined by:

$$H*x = w*x, \; y^H*H = w*y^H.$$

Arguments

SIDE (input) CHARACTER*1
= 'R': compute right eigenvectors only;
= 'L': compute left eigenvectors only;
= 'B': compute both right and left eigenvectors.

EIGSRC (input) CHARACTER*1
Specifies the source of eigenvalues supplied in (WR,WI)/W:
= 'Q': the eigenvalues were found using SHSEQR/CHSEQR; thus, if H has zero subdiagonal elements, and so is block-triangular, then the jth eigenvalue can be assumed to be an eigenvalue of the block containing the jth row/column. This property allows SHSEIN/CHSEIN to perform inverse iteration on just one diagonal block.
= 'N': no assumptions are made on the correspondence between eigenvalues and diagonal blocks. In this case, SHSEIN/CHSEIN must always perform inverse iteration using the whole matrix H.

INITV (input) CHARACTER*1
= 'N': no initial vectors are supplied;
= 'U': user-supplied initial vectors are stored in the arrays VL and/or VR.

SELECT *SHSEIN* (input/output) LOGICAL array, dimension (N)
CHSEIN (input) LOGICAL array, dimension (N)
SHSEIN
Specifies the eigenvectors to be computed. To select the real eigenvector corresponding to a real eigenvalue WR(j), SELECT(j) must be set to .TRUE.. To select the complex eigenvector corresponding to a complex eigenvalue (WR(j),WI(j)), with complex conjugate (WR(j+1),WI(j+1)), either SELECT(j) or SELECT(j+1) or both must be set to .TRUE.; then on exit SELECT(j) is .TRUE. and SELECT(j+1) is .FALSE..
CHSEIN
Specifies the eigenvectors to be computed. To select the eigenvector corresponding to the eigenvalue W(j), SELECT(j) must be set to .TRUE..

N (input) INTEGER
The order of the matrix H. $N \geq 0$.

H (input) REAL/COMPLEX array, dimension (LDH,N)
The upper Hessenberg matrix H.

LDH (input) INTEGER
The leading dimension of the array H. $LDH \geq \max(1,N)$.

WR *SHSEIN only* (input/output) REAL array, dimension (N)
WI *SHSEIN only* (input) REAL array, dimension (N)
On entry, the real and imaginary parts of the eigenvalues of H; a complex conjugate pair of eigenvalues must be stored in consecutive elements of WR and WI.
On exit, WR may have been altered since close eigenvalues are perturbed slightly in searching for independent eigenvectors.

W *CHSEIN only* (input/output) COMPLEX array, dimension (N)
On entry, the eigenvalues of H.
On exit, the real parts of W may have been altered since close eigenvalues are perturbed slightly in searching for independent eigenvectors.

VL (input/output) REAL/COMPLEX array, dimension (LDVL,MM)
On entry, if INITV = 'U' and SIDE = 'L' or 'B', VL must contain starting vectors for the inverse iteration for the left eigenvectors; the starting vector for each eigenvector must be in the same column(s) in which the eigenvector will be stored.
On exit, if SIDE = 'L' or 'B', the left eigenvectors specified by SELECT will be stored one after another in the columns of VL, in the same order as their eigenvalues. If SIDE = 'R', VL is not referenced.
SHSEIN only
A complex eigenvector corresponding to a complex eigenvalue is stored in two consecutive columns, the first holding the real part and the second the imaginary part.

LDVL (input) INTEGER
The leading dimension of the array VL.
$LDVL \geq \max(1,N)$ if SIDE = 'L' or 'B'; $LDVL \geq 1$ otherwise.

VR (input/output) REAL/COMPLEX array, dimension (LDVR,MM)
On entry, if INITV = 'U' and SIDE = 'R' or 'B', VR must contain starting vectors for the inverse iteration for the right eigenvectors; the starting vector for each eigenvector must be in the same column(s) in which the eigenvector will be stored.
On exit, if SIDE = 'R' or 'B', the right eigenvectors specified by SELECT will be stored one after another in the columns of VR, in the same order as their eigenvalues. If SIDE = 'L', VR is not referenced.
SHSEIN
A complex eigenvector corresponding to a complex eigenvalue is stored in two consecutive columns, the first holding the real part and the second the imaginary part.

LDVR (input) INTEGER
The leading dimension of the array VR.
$LDVR \geq \max(1,N)$ if SIDE = 'R' or 'B'; $LDVR \geq 1$ otherwise.

MM (input) INTEGER
The number of columns in the arrays VL and/or VR. $MM \geq M$.

M (output) INTEGER
The number of columns in the arrays VL and/or VR actually used to store the eigenvectors.
SHSEIN
Each selected real eigenvector occupies one column and each selected complex eigenvector occupies two columns.
CHSEIN
Each selected eigenvector occupies one column.

WORK *SHSEIN* (workspace) REAL array, dimension ((N+2)*N)
CHSEIN (workspace) COMPLEX array, dimension (N*N)

RWORK *CHSEIN only* (workspace) REAL array, dimension (N)

IFAILL (output) INTEGER array, dimension (MM)
If SIDE = 'L' or 'B', IFAILL(i) = j > 0 if the left eigenvector in the ith column of VL (corresponding to the eigenvalue w(j)) failed to converge; IFAILL(i) = 0 if the eigenvector converged satisfactorily.
If SIDE = 'R', IFAILL is not referenced.
SHSEIN only
If the ith and $(i+1)^{st}$ columns of VL hold a complex eigenvector, then IFAILL(i) and IFAILL(i+1) are set to the same value.

IFAILR (output) INTEGER array, dimension (MM)
If SIDE = 'R' or 'B', IFAILR(i) = j > 0 if the right eigenvector in the ith column of VR (corresponding to the jth eigenvalue) failed to converge; IFAILR(i) = 0 if the eigenvector converged satisfactorily.
If SIDE = 'L', IFAILR is not referenced.
SHSEIN only

If the ith and $(i+1)^{st}$ columns of VR hold a complex eigenvector, then IFAILR(i) and IFAILR(i+1) are set to the same value.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, i is the number of eigenvectors which failed to converge; see IFAILL and IFAILR for further details.

SHSEQR/CHSEQR

```
      SUBROUTINE SHSEQR( JOB, COMPZ, N, ILO, IHI, H, LDH, WR, WI, Z,
     $                   LDZ, WORK, LWORK, INFO )
      CHARACTER          COMPZ, JOB
      INTEGER            IHI, ILO, INFO, LDH, LDZ, LWORK, N
      REAL               H( LDH, * ), WI( * ), WORK( * ), WR( * ),
     $                   Z( LDZ, * )

      SUBROUTINE CHSEQR( JOB, COMPZ, N, ILO, IHI, H, LDH, W, Z, LDZ,
     $                   WORK, LWORK, INFO )
      CHARACTER          COMPZ, JOB
      INTEGER            IHI, ILO, INFO, LDH, LDZ, LWORK, N
      COMPLEX            H( LDH, * ), W( * ), WORK( * ), Z( LDZ, * )
```

Purpose

SHSEQR/CHSEQR computes the eigenvalues of a real/complex upper Hessenberg matrix H and, optionally, the matrices T and Z from the Schur decomposition $H = Z*T*Z^H$, where T is an upper quasi-triangular/triangular matrix (the Schur form), and Z is the orthogonal/unitary matrix of Schur vectors.

Optionally Z may be postmultiplied into an input orthogonal/unitary matrix Q, so that this routine can give the Schur factorization of a matrix A which has been reduced to the Hessenberg form H by the orthogonal/unitary matrix Q:

$$A = Q*H*Q^H = (Q*Z)*T*(Q*Z)^H.$$

Arguments

JOB (input) CHARACTER*1
= 'E': compute eigenvalues only;
= 'S': compute eigenvalues and the Schur form T.

COMPZ (input) CHARACTER*1
= 'N': no Schur vectors are computed;
= 'I': Z is initialized to the unit matrix and the matrix Z of Schur vectors of H is returned;
= 'V': Z must contain an orthogonal/unitary matrix Q on entry, and the product Q*Z is returned.

N (input) INTEGER
The order of the matrix H. N ≥ 0.

ILO, IHI (input) INTEGER
It is assumed that H is already upper triangular in rows and columns 1:ilo−1 and ihi+1:n. ILO and IHI are normally set by a previous call to SGEBAL,/CGEBAL, and then passed to SGEHRD/CGEHRD when the matrix output by SGEBAL/CGEBAL is reduced to Hessenberg form. Otherwise ILO and IHI should be set to 1 and N respectively.
1 ≤ ILO ≤ IHI ≤ N, if N > 0; ILO = 1 and IHI = 0, if N = 0.

H (input/output) REAL/COMPLEX array, dimension (LDH,N)
On entry, the upper Hessenberg matrix H.
On exit, if JOB = 'S', H contains the upper quasi-triangular/triangular matrix T from the Schur decomposition (the Schur form).
If JOB = 'E', the contents of H are unspecified on exit.
SHSEQR only
2-by-2 diagonal blocks (corresponding to complex conjugate pairs of eigenvalues) are returned in standard form, with H(i,i) = H(i+1,i+1) and H(i+1,i)*H(i,i+1) < 0.

LDH (input) INTEGER
The leading dimension of the array H. LDH ≥ max(1,N).

WR, WI *SHSEQR only* (output) REAL array, dimension (N)
The real and imaginary parts, respectively, of the computed eigenvalues. If two eigenvalues are computed as a complex conjugate pair, they are stored in consecutive elements of WR and WI, say the ith and $(i+1)^{st}$, with WI(i) > 0 and WI(i+1) < 0. If JOB = 'S', the eigenvalues are stored in the same order as on the diagonal of the Schur form returned in H, with WR(i) = H(i,i) and, if H(i:i+1,i:i+1) is a 2-by-2 diagonal block, WI(i) = $\sqrt{(H(i+1,i)*H(i,i+1))}$ and WI(i+1) = −WI(i).

W *CHSEQR only* (output) COMPLEX array, dimension (N)
The computed eigenvalues. If JOB = 'S', the eigenvalues are stored in the same order as on the diagonal of the Schur form returned in H, with W(i) = H(i,i).

Z (input/output) REAL/COMPLEX array, dimension (LDZ,N)
If COMPZ = 'N': Z is not referenced.
If COMPZ = 'I': on entry, Z need not be set, and on exit, Z contains the orthogonal/unitary matrix Z of the Schur vectors of H.
If COMPZ = 'V': on entry, Z must contain an n-by-n matrix Q, which is assumed to be equal to the unit matrix except for the submatrix Z(ilo:ihi,ilo:ihi); on exit Z contains Q*Z. Normally Q is the orthogonal/unitary matrix generated by SORGHR/CUNGHR after the call to SGEHRD/CGEHRD which formed the Hessenberg matrix H.

LDZ (input) INTEGER
The leading dimension of the array Z. LDZ ≥ max(1,N) if COMPZ = 'I' or 'V'; LDZ ≥ 1 otherwise.

WORK (workspace) REAL/COMPLEX array, dimension (N)

LWORK (input) INTEGER
This argument is currently redundant.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, SHSEQR/CHSEQR failed to compute all the eigenvalues in a total of 30*(IHI−ILO+1) iterations; elements 1:ilo−1 and i+1:n of WR and WI (*SHSEQR*) or of W (*CHSEQR*) contain those eigenvalues which have been successfully computed.

SOPGTR/CUPGTR

```
SUBROUTINE SOPGTR( UPLO, N, AP, TAU, Q, LDQ, WORK, INFO )
CHARACTER          UPLO
INTEGER            INFO, LDQ, N
REAL               AP( * ), Q( LDQ, * ), TAU( * ), WORK( * )

SUBROUTINE CUPGTR( UPLO, N, AP, TAU, Q, LDQ, WORK, INFO )
CHARACTER          UPLO
INTEGER            INFO, LDQ, N
COMPLEX            AP( * ), Q( LDQ, * ), TAU( * ), WORK( * )
```

Purpose

SOPGTR/CUPGTR generates a real/complex orthogonal/unitary matrix Q which is defined as the product of n−1 elementary reflectors H_i of order n, as returned by SSPTRD/CHPTRD using packed storage:

if UPLO = 'U', $Q = H_{n-1} \cdots H_2 H_1$,

if UPLO = 'L', $Q = H_1 H_2 \cdots H_{n-1}$.

Arguments

UPLO (input) CHARACTER*1
= 'U': Upper triangular packed storage used in previous call to SSPTRD/CHPTRD;
= 'L': Lower triangular packed storage used in previous call to SSPTRD/CHPTRD.

N (input) INTEGER
The order of the matrix Q. N ≥ 0.

AP (input) REAL/COMPLEX array, dimension (N*(N+1)/2)
The vectors which define the elementary reflectors, as returned by SSPTRD/CHPTRD.

TAU (input) REAL/COMPLEX array, dimension (N−1)
TAU(i) must contain the scalar factor of the elementary reflector H_i, as returned by SSPTRD/CHPTRD.

Q (output) REAL/COMPLEX array, dimension (LDQ,N)
The n-by-n orthogonal/unitary matrix Q.

LDQ (input) INTEGER
The leading dimension of the array Q. LDQ ≥ max(1,N).

WORK (workspace) REAL/COMPLEX array, dimension (N−1)

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SOPMTR/CUPMTR

```
SUBROUTINE SOPMTR( SIDE, UPLO, TRANS, M, N, AP, TAU, C, LDC, WORK,
$                  INFO )
CHARACTER          SIDE, TRANS, UPLO
INTEGER            INFO, LDC, M, N
REAL               AP( * ), C( LDC, * ), TAU( * ), WORK( * )

SUBROUTINE CUPMTR( SIDE, UPLO, TRANS, M, N, AP, TAU, C, LDC, WORK,
$                  INFO )
CHARACTER          SIDE, TRANS, UPLO
INTEGER            INFO, LDC, M, N
COMPLEX            AP( * ), C( LDC, * ), TAU( * ), WORK( * )
```

Purpose

SOPMTR/CUPMTR overwrites the general real/complex m-by-n matrix C with

	SIDE = 'L'	SIDE = 'R'	
TRANS = 'N':	Q*C	C*Q	
TRANS = 'T':	Q^T*C	C*Q^T	(*SOPMTR only*)
TRANS = 'C':	Q^H*C	C*Q^H	(*CUPMTR only*)

where Q is a real/complex orthogonal/unitary matrix of order nq, with nq = m if SIDE = 'L' and nq = n if SIDE = 'R'. Q is defined as the product of nq−1 elementary reflectors, as returned by SSPTRD/CHPTRD using packed storage:

if UPLO = 'U', $Q = H_{nq-1} \cdots H_2 H_1$;

if UPLO = 'L', $Q = H_1 H_2 \cdots H_{nq-1}$.

Arguments

SIDE (input) CHARACTER*1
= 'L': apply Q or Q^H from the Left;
= 'R': apply Q or Q^H from the Right.

UPLO (input) CHARACTER*1

= 'U': Upper triangular packed storage used in previous call to SSPTRD/CHPTRD;
= 'L': Lower triangular packed storage used in previous call to SSPTRD/CHPTRD.

TRANS (input) CHARACTER*1
= 'N': No transpose, apply Q;
= 'T': Transpose, apply Q^T (*SOPMTR only*)
= 'C': Conjugate transpose, apply Q^H (*CUPMTR only*)

M (input) INTEGER
The number of rows of the matrix C. $M \geq 0$.

N (input) INTEGER
The number of columns of the matrix C. $N \geq 0$.

AP (input) REAL/COMPLEX array, dimension
(M*(M+1)/2) if SIDE = 'L', or (N*(N+1)/2) if SIDE = 'R'
The vectors which define the elementary reflectors, as returned by SSPTRD/CHPTRD. AP is modified by the routine but restored on exit.

TAU (input) REAL/COMPLEX array, dimension
(M−1) if SIDE = 'L' or (N−1) if SIDE = 'R'
TAU(i) must contain the scalar factor of the elementary reflector H_i, as returned by SSPTRD/CHPTRD.

C (input/output) REAL/COMPLEX array, dimension (LDC,N)
On entry, the m-by-n matrix C.
On exit, C is overwritten by Q*C or Q^H*C or C*Q^H or C*Q.

LDC (input) INTEGER
The leading dimension of the array C. LDC $\geq$ max(1,M).

WORK (workspace) REAL/COMPLEX array, dimension
(N) if SIDE = 'L', or (M) if SIDE = 'R'

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SORGBR/CUNGBR

```
SUBROUTINE SORGBR( VECT, M, N, K, A, LDA, TAU, WORK, LWORK, INFO )
CHARACTER          VECT
INTEGER            INFO, K, LDA, LWORK, M, N
REAL               A( LDA, * ), TAU( * ), WORK( LWORK )

SUBROUTINE CUNGBR( VECT, M, N, K, A, LDA, TAU, WORK, LWORK, INFO )
CHARACTER          VECT
INTEGER            INFO, K, LDA, LWORK, M, N
COMPLEX            A( LDA, * ), TAU( * ), WORK( LWORK )
```

Purpose

SORGBR/CUNGBR generates one of the real/complex orthogonal/unitary matrices Q or P^H determined by SGEBRD/CGEBRD when reducing a real/complex matrix A to bidiagonal form: A = Q*B*P^H. Q and P^H are defined as products of elementary reflectors H_i or G_i respectively.

If VECT = 'Q', A is assumed to have been an m-by-k matrix, and Q is of order M:
if $M \geq K$, $Q = H_1 H_2 \cdots H_k$ and SORGBR/CUNGBR returns the first N columns of Q, where $M \geq N \geq K$;
if $M < K$, $Q = H_1 H_2 \cdots H_{m-1}$ and SORGBR/CUNGBR returns Q as an m-by-m matrix.

If VECT = 'P', A is assumed to have been a k-by-n matrix, and P^H is of order n:
if $K < N$, $P^H = G_k \cdots G_2 G_1$ and SORGBR/CUNGBR returns the first M rows of P^H, where $N \geq M \geq K$;
if $K \geq N$, $P^H = G_{n-1} \cdots G_2 G_1$ and SORGBR/CUNGBR returns P^H as an n-by-n matrix.

Arguments

VECT (input) CHARACTER*1
Specifies whether the matrix Q or the matrix P^H is required, as defined in the transformation applied by SGEBRD/CGEBRD:
= 'Q': generate Q;
= 'P': generate P^H.

M (input) INTEGER
The number of rows of the matrix Q or P^H to be returned.
$M \geq 0$.

N (input) INTEGER
The number of columns of the matrix Q or P^H to be returned.
$N \geq 0$.
If VECT = 'Q', $M \geq N \geq$ min(M,K);
if VECT = 'P', $N \geq M \geq$ min(N,K).

K (input) INTEGER
If VECT = 'Q', the number of columns in the original m-by-k matrix reduced by SGEBRD/CGEBRD.
If VECT = 'P', the number of rows in the original k-by-n matrix reduced by SGEBRD/CGEBRD.
$K \geq 0$.

A (input/output) REAL/COMPLEX array, dimension (LDA,N)
On entry, the vectors which define the elementary reflectors, as returned by SGEBRD/CGEBRD.
On exit, the m-by-n matrix Q or P^H.

LDA (input) INTEGER
The leading dimension of the array A. LDA $\geq$ max(1,M).

TAU (input) REAL/COMPLEX array, dimension
(min(M,K)) if VECT = 'Q', or (min(N,K)) if VECT = 'P'

TAU(i) must contain the scalar factor of the elementary reflector H_i or G_i, which determines Q or P^H, as returned by SGEBRD/CGEBRD in its array argument TAUQ or TAUP.

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK. LWORK $\geq$ max(1,min(M,N)). For optimum performance LWORK $\geq$ min(M,N)*NB, where NB is the optimal blocksize.

If LWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SORGHR/CUNGHR

```
SUBROUTINE SORGHR( N, ILO, IHI, A, LDA, TAU, WORK, LWORK, INFO )
INTEGER            IHI, ILO, INFO, LDA, LWORK, N
REAL               A( LDA, * ), TAU( * ), WORK( LWORK )

SUBROUTINE CUNGHR( N, ILO, IHI, A, LDA, TAU, WORK, LWORK, INFO )
INTEGER            IHI, ILO, INFO, LDA, LWORK, N
COMPLEX            A( LDA, * ), TAU( * ), WORK( LWORK )
```

Purpose

SORGHR/CUNGHR generates a real/complex orthogonal/unitary matrix Q which is defined as the product of ihi−ilo elementary reflectors H_i of order n, as returned by SGEHRD/CGEHRD:

$Q = H_{ilo} H_{ilo+1} \cdots H_{ihi-1}$.

Arguments

N (input) INTEGER
The order of the matrix Q. N $\geq$ 0.

ILO, IHI (input) INTEGER
ILO and IHI must have the same values as in the previous call of SGEHRD/CGEHRD. Q is equal to the unit matrix except in the submatrix Q(ilo+1:ihi,ilo+1:ihi).
1 $\leq$ ILO $\leq$ IHI $\leq$ N, if N > 0; ILO = 1 and IHI = 0, if N = 0.

A (input/output) REAL/COMPLEX array, dimension (LDA,N)
On entry, the vectors which define the elementary reflectors, as returned by SGEHRD/CGEHRD.
On exit, the n-by-n orthogonal/unitary matrix Q.

LDA (input) INTEGER
The leading dimension of the array A. LDA $\geq$ max(1,N).

TAU (input) REAL/COMPLEX array, dimension (N−1)
TAU(i) must contain the scalar factor of the elementary reflector H_i, as returned by SGEHRD/CGEHRD.

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK. LWORK $\geq$ IHI−ILO. For optimum performance LWORK $\geq$ (IHI−ILO)*NB, where NB is the optimal blocksize.

If LWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SORGLQ/CUNGLQ

```
SUBROUTINE SORGLQ( M, N, K, A, LDA, TAU, WORK, LWORK, INFO )
INTEGER            INFO, K, LDA, LWORK, M, N
REAL               A( LDA, * ), TAU( * ), WORK( LWORK )

SUBROUTINE CUNGLQ( M, N, K, A, LDA, TAU, WORK, LWORK, INFO )
INTEGER            INFO, K, LDA, LWORK, M, N
COMPLEX            A( LDA, * ), TAU( * ), WORK( LWORK )
```

Purpose

SORGLQ/CUNGLQ generates an m-by-n real/complex matrix Q with orthonormal rows, which is defined as the first m rows of a product of k elementary reflectors of order n

$Q = H_k \cdots H_2 H_1$ $(SORGLQ)$

$Q = H_k{}^H \cdots H_2{}^H H_1{}^H$ $(CUNGLQ)$

as returned by SGELQF/CGELQF.

Arguments

M (input) INTEGER
The number of rows of the matrix Q. M $\geq$ 0.

N (input) INTEGER
The number of columns of the matrix Q. N $\geq$ M.

K (input) INTEGER
The number of elementary reflectors whose product defines the matrix Q. M $\geq$ K $\geq$ 0.

A (input/output) REAL/COMPLEX array, dimension (LDA,N)
On entry, the ith row must contain the vector which defines the elementary reflector H_i, for i = 1,2,...,k, as returned by SGELQF/CGELQF in the first k rows of its array argument A.
On exit, the m-by-n matrix Q.

LDA (input) INTEGER
The first dimension of the array A. LDA $\geq$ max(1,M).

TAU (input) REAL/COMPLEX array, dimension (K)
TAU(i) must contain the scalar factor of the elementary reflector H_i, as returned by SGELQF/CGELQF.

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK. LWORK $\geq$ max(1,M). For optimum performance LWORK $\geq$ M*NB, where NB is the optimal blocksize.

If LWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument has an illegal value

SORGQL/CUNGQL

```
SUBROUTINE SORGQL( M, N, K, A, LDA, TAU, WORK, LWORK, INFO )
INTEGER            INFO, K, LDA, LWORK, M, N
REAL               A( LDA, * ), TAU( * ), WORK( LWORK )

SUBROUTINE CUNGQL( M, N, K, A, LDA, TAU, WORK, LWORK, INFO )
INTEGER            INFO, K, LDA, LWORK, M, N
COMPLEX            A( LDA, * ), TAU( * ), WORK( LWORK )
```

Purpose

SORGQL/CUNGQL generates an m-by-n real/complex matrix Q with orthonormal columns, which is defined as the last n columns of a product of k elementary reflectors H_i of order m

$$Q = H_k \cdots H_2 H_1$$

as returned by SGEQLF/CGEQLF.

Arguments

M (input) INTEGER
The number of rows of the matrix Q. M $\geq$ 0.

N (input) INTEGER
The number of columns of the matrix Q. M $\geq$ N $\geq$ 0.

K (input) INTEGER
The number of elementary reflectors whose product defines the matrix Q. N $\geq$ K $\geq$ 0.

A (input/output) REAL/COMPLEX array, dimension (LDA,N)
On entry, the $(n-k+i)^{th}$ column must contain the vector which defines the elementary reflector H_i, for i = 1,2,...,k, as returned by SGEQLF/CGEQLF in the last k columns of its array argument A.
On exit, the m-by-n matrix Q.

LDA (input) INTEGER
The first dimension of the array A. LDA $\geq$ max(1,M).

TAU (input) REAL/COMPLEX array, dimension (K)
TAU(i) must contain the scalar factor of the elementary reflector H_i, as returned by SGEQLF/CGEQLF.

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK. LWORK $\geq$ max(1,N). For optimum performance LWORK $\geq$ N*NB, where NB is the optimal blocksize.

If LWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument has an illegal value

SORGQR/CUNGQR

```
SUBROUTINE SORGQR( M, N, K, A, LDA, TAU, WORK, LWORK, INFO )
INTEGER            INFO, K, LDA, LWORK, M, N
REAL               A( LDA, * ), TAU( * ), WORK( LWORK )

SUBROUTINE CUNGQR( M, N, K, A, LDA, TAU, WORK, LWORK, INFO )
INTEGER            INFO, K, LDA, LWORK, M, N
COMPLEX            A( LDA, * ), TAU( * ), WORK( LWORK )
```

Purpose

SORGQR/CUNGQR generates an m-by-n real/complex matrix Q with orthonormal columns, which is defined as the first n columns of a product of k elementary reflectors H_i of order m

$$Q = H_1 H_2 \cdots H_k$$

as returned by SGEQRF/CGEQRF.

Arguments

M (input) INTEGER
The number of rows of the matrix Q. M $\geq$ 0.

N (input) INTEGER
The number of columns of the matrix Q. M $\geq$ N $\geq$ 0.

K (input) INTEGER
The number of elementary reflectors whose product defines the matrix Q. N $\geq$ K $\geq$ 0.

A (input/output) REAL/COMPLEX array, dimension (LDA,N)
On entry, the ith column must contain the vector which defines the elementary reflector H_i, for i = 1,2,...,k, as returned by SGEQRF/CGEQRF in the first k columns of its array argument A.
On exit, the m-by-n matrix Q.

LDA (input) INTEGER
The first dimension of the array A. LDA $\geq$ max(1,M).

TAU (input) REAL/COMPLEX array, dimension (K)
TAU(i) must contain the scalar factor of the elementary reflector H_i, as returned by SGEQRF/CGEQRF.

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK. LWORK $\geq$ max(1,N). For optimum performance LWORK $\geq$ N*NB, where NB is the optimal blocksize.

If LWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument has an illegal value

SORGRQ/CUNGRQ

```
SUBROUTINE SORGRQ( M, N, K, A, LDA, TAU, WORK, LWORK, INFO )
INTEGER            INFO, K, LDA, LWORK, M, N
REAL               A( LDA, * ), TAU( * ), WORK( LWORK )

SUBROUTINE CUNGRQ( M, N, K, A, LDA, TAU, WORK, LWORK, INFO )
INTEGER            INFO, K, LDA, LWORK, M, N
COMPLEX            A( LDA, * ), TAU( * ), WORK( LWORK )
```

Purpose

SORGRQ/CUNGRQ generates an m-by-n real/complex matrix Q with orthonormal rows, which is defined as the last m rows of a product of k elementary reflectors H_i of order n

$$Q = H_1 H_2 \cdots H_k \ (SORGRQ)$$

$$Q = H_1{}^H H_2{}^H \cdots H_k{}^H \ (CUNGRQ)$$

as returned by SGERQF/CGERQF.

Arguments

M (input) INTEGER
The number of rows of the matrix Q. M $\geq$ 0.

N (input) INTEGER
The number of columns of the matrix Q. N $\geq$ M.

K (input) INTEGER
The number of elementary reflectors whose product defines the matrix Q. M $\geq$ K $\geq$ 0.

A (input/output) REAL/COMPLEX array, dimension (LDA,N)
On entry, the $(m-k+i)^{th}$ row must contain the vector which defines the elementary reflector H_i, for i = 1,2,...,k, as returned by SGERQF/CGERQF in the last k rows of its array argument A.
On exit, the m-by-n matrix Q.

LDA (input) INTEGER
The first dimension of the array A. LDA $\geq$ max(1,M).

TAU (input) REAL/COMPLEX array, dimension (K)
TAU(i) must contain the scalar factor of the elementary reflector H_i, as returned by SGERQF/CGERQF.

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK. LWORK $\geq$ max(1,M). For optimum performance LWORK $\geq$ M*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = $-$i, the ith argument has an illegal value

---

SORGTR/CUNGTR

```
SUBROUTINE SORGTR( UPLO, N, A, LDA, TAU, WORK, LWORK, INFO )
CHARACTER          UPLO
INTEGER            INFO, LDA, LWORK, N
REAL               A( LDA, * ), TAU( * ), WORK( LWORK )

SUBROUTINE CUNGTR( UPLO, N, A, LDA, TAU, WORK, LWORK, INFO )
CHARACTER          UPLO
INTEGER            INFO, LDA, LWORK, N
COMPLEX            A( LDA, * ), TAU( * ), WORK( LWORK )
```

Purpose

SORGTR/CUNGTR generates a real/complex orthogonal/unitary matrix Q which is defined as the product of n$-$1 elementary reflectors H_i of order n, as returned by SSYTRD/CHETRD:

if UPLO = 'U', $Q = H_{n-1} \cdots H_2 H_1$,

if UPLO = 'L', $Q = H_1 H_2 \cdots H_{n-1}$.

Arguments

UPLO (input) CHARACTER*1
= 'U': Upper triangle of A contains elementary reflectors from SSYTRD/CHETRD;
= 'L': Lower triangle of A contains elementary reflectors from SSYTRD/CHETRD.

N (input) INTEGER
The order of the matrix Q. N $\geq$ 0.

A (input/output) REAL/COMPLEX array, dimension (LDA,N)
On entry, the vectors which define the elementary reflectors, as returned by SSYTRD/CHETRD.
On exit, the n-by-n orthogonal/unitary matrix Q.

LDA (input) INTEGER
The leading dimension of the array A. LDA $\geq$ max(1,N).

TAU (input) REAL/COMPLEX array, dimension (N$-$1)
TAU(i) must contain the scalar factor of the elementary reflector H_i, as returned by SSYTRD/CHETRD.

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK. LWORK $\geq$ max(1,N$-$1). For optimum performance LWORK $\geq$ (N$-$1)*NB, where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = $-$i, the ith argument had an illegal value.

SORMBR/CUNMBR

```
 SUBROUTINE SORMBR( VECT, SIDE, TRANS, M, N, K, A, LDA, TAU, C,
$                   LDC, WORK, LWORK, INFO )
 CHARACTER          SIDE, TRANS, VECT
 INTEGER            INFO, K, LDA, LDC, LWORK, M, N
 REAL               A( LDA, * ), C( LDC, * ), TAU( * ),
$                   WORK( LWORK )

 SUBROUTINE CUNMBR( VECT, SIDE, TRANS, M, N, K, A, LDA, TAU, C,
$                   LDC, WORK, LWORK, INFO )
 CHARACTER          SIDE, TRANS, VECT
 INTEGER            INFO, K, LDA, LDC, LWORK, M, N
 COMPLEX            A( LDA, * ), C( LDC, * ), TAU( * ),
$                   WORK( LWORK )
```

Purpose

If VECT = 'Q', SORMBR/CUNMBR overwrites the general real/complex m-by-n matrix C with

	SIDE = 'L'	SIDE = 'R'	
TRANS = 'N':	Q*C	C*Q	
TRANS = 'T':	Q^T*C	C*Q^T	(*SORMBR only*)
TRANS = 'C':	Q^H*C	C*Q^H	(*CUNMBR only*)

If VECT = 'P', SORMBR/CUNMBR overwrites the general real/complex m-by-n matrix C with

	SIDE = 'L'	SIDE = 'R'	
TRANS = 'N':	P*C	C*P	
TRANS = 'T':	P^T*C	C*P^T	(*SORMBR only*)
TRANS = 'C':	P^H*C	C*P^H	(*CUNMBR only*)

Here Q and P^H are the orthogonal/unitary matrices determined by SGEBRD/CGEBRD when reducing a real/complex matrix A to bidiagonal form: A = Q*B*P^H. Q and P^H are defined as products of elementary reflectors H_i and G_i, respectively.

Let nq = m if SIDE = 'L' and nq = n if SIDE = 'R'. Thus nq is the order of the orthogonal/unitary matrix Q or P^H that is applied.

If VECT = 'Q', A is assumed to have been an nq-by-k matrix:

if nq $\geq$ k, $Q = H_1 H_2 \cdots H_k$;

if nq < k, $Q = H_1 H_2 \cdots H_{nq-1}$.

If VECT = 'P', A is assumed to have been a k-by-nq matrix:

if k < nq, $P = G_1 G_2 \cdots G_k$;

if k $\geq$ nq, $P = G_1 G_2 \cdots G_{nq-1}$.

Arguments

VECT (input) CHARACTER*1
= 'Q': apply Q or Q^H;
= 'P': apply P or P^H.

SIDE (input) CHARACTER*1
= 'L': apply Q, Q^H, P or P^H from the Left;
= 'R': apply Q, Q^H, P or P^H from the Right.

TRANS (input) CHARACTER*1
= 'N': No transpose, apply Q or P;
= 'T': Transpose, apply Q^T or P^T (*SORMBR only*);
= 'C': Conjugate transpose, apply Q^H or P^H (*CUNMBR only*).

M (input) INTEGER
The number of rows of the matrix C. M $\geq$ 0.

N (input) INTEGER
The number of columns of the matrix C. N $\geq$ 0.

K (input) INTEGER
If VECT = 'Q', the number of columns in the original matrix reduced by SGEBRD/CGEBRD.
If VECT = 'P', the number of rows in the original matrix reduced by SGEBRD/CGEBRD.
K $\geq$ 0.

A (input) REAL/COMPLEX array, dimension (LDA,min(nq,K)) if VECT = 'Q', or (LDA,nq) if VECT = 'P'
The vectors which define the elementary reflectors H_i and G_i, whose products determine the matrices Q and P, as returned by SGEBRD/CGEBRD.

LDA (input) INTEGER
The leading dimension of the array A.
If VECT = 'Q', LDA $\geq$ max(1,nq);
if VECT = 'P', LDA $\geq$ max(1,min(nq,K)).

TAU (input) REAL/COMPLEX array, dimension (min(nq,K))
TAU(i) must contain the scalar factor of the elementary reflector H_i or G_i which determines Q or P, as returned by SGEBRD/CGEBRD in the array argument TAUQ or TAUP.

C (input/output) REAL/COMPLEX array, dimension (LDC,N)
On entry, the m-by-n matrix C.
On exit, C is overwritten by Q*C or Q^H*C or C*Q^H or C*Q or P*C or P^H*C or C*P or C*P^H.

LDC (input) INTEGER
The leading dimension of the array C. LDC $\geq$ max(1,M).

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK. If SIDE = 'L', LWORK $\geq$ max(1,N); if SIDE = 'R', LWORK $\geq$ max(1,M). For optimum performance LWORK $\geq$ N*NB if SIDE = 'L', and LWORK $\geq$ M*NB if SIDE = 'R', where NB is the optimal blocksize.

If LWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SORMHR/CUNMHR

```
      SUBROUTINE SORMHR( SIDE, TRANS, M, N, ILO, IHI, A, LDA, TAU, C,
     $                   LDC, WORK, LWORK, INFO )
      CHARACTER          SIDE, TRANS
      INTEGER            IHI, ILO, INFO, LDA, LDC, LWORK, M, N
      REAL               A( LDA, * ), C( LDC, * ), TAU( * ),
     $                   WORK( LWORK )

      SUBROUTINE CUNMHR( SIDE, TRANS, M, N, ILO, IHI, A, LDA, TAU, C,
     $                   LDC, WORK, LWORK, INFO )
      CHARACTER          SIDE, TRANS
      INTEGER            IHI, ILO, INFO, LDA, LDC, LWORK, M, N
      COMPLEX            A( LDA, * ), C( LDC, * ), TAU( * ),
     $                   WORK( LWORK )
```

Purpose

SORMHR/CUNMHR overwrites the general real/complex m-by-n matrix C with

	SIDE = 'L'	SIDE = 'R'	
TRANS = 'N':	Q*C	C*Q	
TRANS = 'T':	Q^T*C	C*Q^T	(*SORMHR only*)
TRANS = 'C':	Q^H*C	C*Q^H	(*CUNMHR only*)

where Q is a real/complex orthogonal/unitary matrix of order nq, with nq = m if SIDE = 'L' and nq = n if SIDE = 'R'. Q is defined as the product of ihi−ilo elementary reflectors, as returned by SGEHRD/CGEHRD:

$Q = H_{ilo}H_{ilo+1}\cdots H_{ihi-1}$.

Arguments

SIDE (input) CHARACTER*1
= 'L': apply Q or Q^H from the Left;
= 'R': apply Q or Q^H from the Right.

TRANS (input) CHARACTER*1
= 'N': No transpose, apply Q
= 'T': Transpose, apply Q^T (*SORMHR only*)
= 'C': Conjugate transpose, apply Q^H (*CUNMHR only*)

M (input) INTEGER
The number of rows of the matrix C. M $\geq$ 0.

N (input) INTEGER
The number of columns of the matrix C. N $\geq$ 0.

ILO, IHI (input) INTEGER
ILO and IHI must have the same values as in the previous call of SGEHRD/CGEHRD. Q is equal to the unit matrix except in the submatrix Q(ilo+1:ihi,ilo+1:ihi).
If SIDE = 'L', then 1 $\leq$ ILO $\leq$ IHI $\leq$ M, if M > 0, and ILO = 1 and IHI = 0, if M = 0;
if SIDE = 'R', then 1 $\leq$ ILO $\leq$ IHI $\leq$ N, if N > 0, and ILO = 1 and IHI = 0, if N = 0.

A (input) REAL/COMPLEX array, dimension
(LDA,M) if SIDE = 'L', or (LDA,N) if SIDE = 'R'
The vectors which define the elementary reflectors, as returned by SGEHRD/CGEHRD.

LDA (input) INTEGER
The leading dimension of the array A.
LDA $\geq$ max(1,M) if SIDE = 'L';
LDA $\geq$ max(1,N) if SIDE = 'R'.

TAU (input) REAL/COMPLEX array, dimension
(M−1) if SIDE = 'L', or (N−1) if SIDE = 'R'
TAU(i) must contain the scalar factor of the elementary reflector H_i, as returned by SGEHRD/CGEHRD.

C (input/output) REAL/COMPLEX array, dimension (LDC,N)
On entry, the m-by-n matrix C.
On exit, C is overwritten by Q*C or Q^H*C or C*Q^H or C*Q.

LDC (input) INTEGER
The leading dimension of the array C. LDC $\geq$ max(1,M).

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK. If SIDE = 'L', LWORK $\geq$ max(1,N); if SIDE = 'R', LWORK $\geq$ max(1,M). For optimum performance LWORK $\geq$ N*NB if SIDE = 'L', and LWORK $\geq$ M*NB if SIDE = 'R', where NB is the optimal blocksize.

If LWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SORMLQ/CUNMLQ

```
      SUBROUTINE SORMLQ( SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC,
     $                   WORK, LWORK, INFO )
      CHARACTER          SIDE, TRANS
      INTEGER            INFO, K, LDA, LDC, LWORK, M, N
      REAL               A( LDA, * ), C( LDC, * ), TAU( * ),
     $                   WORK( LWORK )

      SUBROUTINE CUNMLQ( SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC,
     $                   WORK, LWORK, INFO )
      CHARACTER          SIDE, TRANS
      INTEGER            INFO, K, LDA, LDC, LWORK, M, N
      COMPLEX            A( LDA, * ), C( LDC, * ), TAU( * ),
     $                   WORK( LWORK )
```

Purpose

SORMLQ/CUNMLQ overwrites the general real/complex m-by-n matrix C with

	SIDE = 'L'	SIDE = 'R'	
TRANS = 'N':	Q*C	C*Q	
TRANS = 'T':	Q^T*C	C*Q^T	*(SORMLQ only)*
TRANS = 'C':	Q^H*C	C*Q^H	*(CUNMLQ only)*

where Q is a real/complex orthogonal/unitary matrix defined as the product of k elementary reflectors H_i

$Q = H_k \cdots H_2 H_1$ *(SORMLQ)*

$Q = H_k{}^H \cdots H_2{}^H H_1{}^H$ *(CUNMLQ)*

as returned by SGELQF/CGELQF. Q is of order m if SIDE = 'L' and of order n if SIDE = 'R'.

Arguments

SIDE (input) CHARACTER*1
= 'L': apply Q or Q^H from the Left;
= 'R': apply Q or Q^H from the Right.

TRANS (input) CHARACTER*1
= 'N': No transpose, apply Q
= 'T': Transpose, apply Q^T *(SORMLQ only)*
= 'C': Conjugate transpose, apply Q^H *(CUNMLQ only)*

M (input) INTEGER
The number of rows of the matrix C. M $\geq$ 0.

N (input) INTEGER
The number of columns of the matrix C. N $\geq$ 0.

K (input) INTEGER
The number of elementary reflectors whose product defines the matrix Q.
If SIDE = 'L', M $\geq$ K $\geq$ 0;
if SIDE = 'R', N $\geq$ K $\geq$ 0.

A (input) REAL/COMPLEX array, dimension (LDA,M) if SIDE = 'L', or (LDA,N) if SIDE = 'R'
The ith row must contain the vector which defines the elementary reflector H_i, for i = 1,2,...,k, as returned by SGELQF/CGELQF in the first k rows of its array argument A. A is modified by the routine but restored on exit.

LDA (input) INTEGER
The leading dimension of the array A. LDA $\geq$ max(1,K).

TAU (input) REAL/COMPLEX array, dimension (K)
TAU(i) must contain the scalar factor of the elementary reflector H_i, as returned by SGELQF/CGELQF.

C (input/output) REAL/COMPLEX array, dimension (LDC,N)
On entry, the m-by-n matrix C.
On exit, C is overwritten by Q*C or Q^H*C or C*Q^H or C*Q.

LDC (input) INTEGER
The leading dimension of the array C. LDC $\geq$ max(1,M).

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK. If SIDE = 'L', LWORK $\geq$ max(1,N); if SIDE = 'R', LWORK $\geq$ max(1,M). For optimum performance LWORK $\geq$ N*NB if SIDE = 'L', and LWORK $\geq$ M*NB if SIDE = 'R', where NB is the optimal blocksize.

If LWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SORMQL/CUNMQL

```
      SUBROUTINE SORMQL( SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC,
     $                   WORK, LWORK, INFO )
      CHARACTER          SIDE, TRANS
      INTEGER            INFO, K, LDA, LDC, LWORK, M, N
      REAL               A( LDA, * ), C( LDC, * ), TAU( * ),
     $                   WORK( LWORK )
```

```
      SUBROUTINE CUNMQL( SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC,
     $                   WORK, LWORK, INFO )
      CHARACTER          SIDE, TRANS
      INTEGER            INFO, K, LDA, LDC, LWORK, M, N
      COMPLEX            A( LDA, * ), C( LDC, * ), TAU( * ),
     $                   WORK( LWORK )
```

Purpose

SORMQL/CUNMQL overwrites the general real/complex m-by-n matrix C with

	SIDE = 'L'	SIDE = 'R'	
TRANS = 'N':	Q*C	C*Q	
TRANS = 'T':	Q^T*C	C*Q^T	*(SORMQL only)*
TRANS = 'C':	Q^H*C	C*Q^H	*(CUNMQL only)*

where Q is a real/complex orthogonal/unitary matrix defined as the product of k elementary reflectors H_i

$$Q = H_k \cdots H_2 H_1$$

as returned by SGEQLF/CGEQLF. Q is of order m if SIDE = 'L' and of order n if SIDE = 'R'.

Arguments

SIDE (input) CHARACTER*1
= 'L': apply Q or Q^H from the Left;
= 'R': apply Q or Q^H from the Right.

TRANS (input) CHARACTER*1
= 'N': No transpose, apply Q
= 'T': Transpose, apply Q^T (*SORMQL only*)
= 'C': Conjugate transpose, apply Q^H (*CUNMQL only*)

M (input) INTEGER
The number of rows of the matrix C. M $\geq$ 0.

N (input) INTEGER
The number of columns of the matrix C. N $\geq$ 0.

K (input) INTEGER
The number of elementary reflectors whose product defines the matrix Q.
If SIDE = 'L', M $\geq$ K $\geq$ 0;
if SIDE = 'R', N $\geq$ K $\geq$ 0.

A (input) REAL/COMPLEX array, dimension (LDA,K)
The ith column must contain the vector which defines the elementary reflector H_i, for i = 1,2,...,k, as returned by SGEQLF/CGEQLF in the last k columns of its array argument A. A is modified by the routine but restored on exit.

LDA (input) INTEGER
The leading dimension of the array A.
If SIDE = 'L', LDA $\geq$ max(1,M);
if SIDE = 'R', LDA $\geq$ max(1,N).

TAU (input) REAL/COMPLEX array, dimension (K)
TAU(i) must contain the scalar factor of the elementary reflector H_i, as returned by SGEQLF/CGEQLF.

C (input/output) REAL/COMPLEX array, dimension (LDC,N)
On entry, the m-by-n matrix C.
On exit, C is overwritten by Q*C or Q^H*C or C*Q^H or C*Q.

LDC (input) INTEGER
The leading dimension of the array C. LDC $\geq$ max(1,M).

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK. If SIDE = 'L', LWORK $\geq$ max(1,N); if SIDE = 'R', LWORK $\geq$ max(1,M). For optimum performance LWORK $\geq$ N*NB if SIDE = 'L', and LWORK $\geq$ M*NB if SIDE = 'R', where NB is the optimal blocksize.

If LWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SORMQR/CUNMQR

```
      SUBROUTINE SORMQR( SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC,
     $                   WORK, LWORK, INFO )
      CHARACTER          SIDE, TRANS
      INTEGER            INFO, K, LDA, LDC, LWORK, M, N
      REAL               A( LDA, * ), C( LDC, * ), TAU( * ),
     $                   WORK( LWORK )

      SUBROUTINE CUNMQR( SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC,
     $                   WORK, LWORK, INFO )
      CHARACTER          SIDE, TRANS
      INTEGER            INFO, K, LDA, LDC, LWORK, M, N
      COMPLEX            A( LDA, * ), C( LDC, * ), TAU( * ),
     $                   WORK( LWORK )
```

Purpose

SORMQR/CUNMQR overwrites the general real/complex m-by-n matrix C with

	SIDE = 'L'	SIDE = 'R'	
TRANS = 'N':	Q*C	C*Q	
TRANS = 'T':	Q^T*C	C*Q^T	(*SORMQR only*)
TRANS = 'C':	Q^H*C	C*Q^H	(*CUNMQR only*)

where Q is a real/complex orthogonal/unitary matrix defined as the product of k elementary reflectors H_i

$Q = H_1 H_2 \cdots H_k$

as returned by SGEQRF/CGEQRF. Q is of order m if SIDE = 'L' and of order n if SIDE = 'R'.

Arguments

SIDE (input) CHARACTER*1
= 'L': apply Q or Q^H from the Left;
= 'R': apply Q or Q^H from the Right.

TRANS (input) CHARACTER*1
= 'N': No transpose, apply Q
= 'T': Transpose, apply Q^T (*SORMQR only*)
= 'C': Conjugate transpose, apply Q^H (*CUNMQR only*)

M (input) INTEGER
The number of rows of the matrix C. M $\geq$ 0.

N (input) INTEGER
The number of columns of the matrix C. N $\geq$ 0.

K (input) INTEGER
The number of elementary reflectors whose product defines the matrix Q.
If SIDE = 'L', M $\geq$ K $\geq$ 0;
if SIDE = 'R', N $\geq$ K $\geq$ 0.

A (input) REAL/COMPLEX array, dimension (LDA,K)
The ith column must contain the vector which defines the elementary reflector H_i, for i = 1,2,...,k, as returned by SGEQRF/CGEQRF in the first k columns of its array argument A. A is modified by the routine but restored on exit.

LDA (input) INTEGER
The leading dimension of the array A.
If SIDE = 'L', LDA $\geq$ max(1,M);
if SIDE = 'R', LDA $\geq$ max(1,N).

TAU (input) REAL/COMPLEX array, dimension (K)
TAU(i) must contain the scalar factor of the elementary reflector H_i, as returned by SGEQRF/CGEQRF.

C (input/output) REAL/COMPLEX array, dimension (LDC,N)
On entry, the m-by-n matrix C.
On exit, C is overwritten by Q*C or Q^H*C or C*Q^H or C*Q.

LDC (input) INTEGER
The leading dimension of the array C. LDC $\geq$ max(1,M).

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK. If SIDE = 'L', LWORK $\geq$ max(1,N); if SIDE = 'R', LWORK $\geq$ max(1,M). For optimum performance LWORK $\geq$ N*NB if SIDE = 'L', and LWORK $\geq$ M*NB if SIDE = 'R', where NB is the optimal blocksize.

If LWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SORMRQ/CUNMRQ

```
      SUBROUTINE SORMRQ( SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC,
     $                   WORK, LWORK, INFO )
      CHARACTER          SIDE, TRANS
      INTEGER            INFO, K, LDA, LDC, LWORK, M, N
      REAL               A( LDA, * ), C( LDC, * ), TAU( * ),
     $                   WORK( LWORK )

      SUBROUTINE CUNMRQ( SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC,
     $                   WORK, LWORK, INFO )
      CHARACTER          SIDE, TRANS
      INTEGER            INFO, K, LDA, LDC, LWORK, M, N
      COMPLEX            A( LDA, * ), C( LDC, * ), TAU( * ),
     $                   WORK( LWORK )
```

Purpose

SORMRQ/CUNMRQ overwrites the general real/complex m-by-n matrix C with

	SIDE = 'L'	SIDE = 'R'	
TRANS = 'N':	Q*C	C*Q	
TRANS = 'T':	Q^T*C	C*Q^T	(*SORMRQ only*)
TRANS = 'C':	Q^H*C	C*Q^H	(*CUNMRQ only*)

where Q is a real/complex orthogonal/unitary matrix defined as the product of k elementary reflectors H_i

$Q = H_1 H_2 \cdots H_k$ (*SORMRQ*)

$Q = H_1{}^H H_2{}^H \cdots H_k{}^H$ (*CUNMRQ*)

as returned by SGERQF/CGERQF. Q is of order m if SIDE = 'L' and of order n if SIDE = 'R'.

Arguments

SIDE (input) CHARACTER*1
= 'L': apply Q or Q^H from the Left;
= 'R': apply Q or Q^H from the Right.

TRANS (input) CHARACTER*1
= 'N': No transpose, apply Q
= 'T': Transpose, apply Q^T (*SORMRQ only*)
= 'C': Conjugate transpose, apply Q^H (*CUNMRQ only*)

M (input) INTEGER
The number of rows of the matrix C. $M \geq 0$.

N (input) INTEGER
The number of columns of the matrix C. $N \geq 0$.

K (input) INTEGER
The number of elementary reflectors whose product defines the matrix Q.
If SIDE = 'L', $M \geq K \geq 0$;
if SIDE = 'R', $N \geq K \geq 0$.

A (input) REAL/COMPLEX array, dimension (LDA,M) if SIDE = 'L', or (LDA,N) if SIDE = 'R'
The ith row must contain the vector which defines the elementary reflector H_i, for i = 1,2,...,k, as returned by SGERQF/CGERQF in the last k rows of its array argument A. A is modified by the routine but restored on exit.

LDA (input) INTEGER
The leading dimension of the array A. LDA $\geq$ max(1,K).

TAU (input) REAL/COMPLEX array, dimension (K)
TAU(i) must contain the scalar factor of the elementary reflector H_i, as returned by SGERQF/CGERQF.

C (input/output) REAL/COMPLEX array, dimension (LDC,N)
On entry, the m-by-n matrix C.
On exit, C is overwritten by Q*C or Q^H*C or C*Q^H or C*Q.

LDC (input) INTEGER
The leading dimension of the array C. LDC $\geq$ max(1,M).

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK. If SIDE = 'L', LWORK $\geq$ max(1,N); if SIDE = 'R', LWORK $\geq$ max(1,M). For optimum performance LWORK $\geq$ N*NB if SIDE = 'L', and LWORK $\geq$ M*NB if SIDE = 'R', where NB is the optimal blocksize.

If LWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SORMRZ/CUNMRZ

```
      SUBROUTINE SORMRZ( SIDE, TRANS, M, N, K, L, A, LDA, TAU, C, LDC,
     $                   WORK, LWORK, INFO )
      CHARACTER          SIDE, TRANS
      INTEGER            INFO, K, L, LDA, LDC, LWORK, M, N
      REAL               A( LDA, * ), C( LDC, * ), TAU( * ),
     $                   WORK( LWORK )

      SUBROUTINE CUNMRZ( SIDE, TRANS, M, N, K, L, A, LDA, TAU, C, LDC,
     $                   WORK, LWORK, INFO )
      CHARACTER          SIDE, TRANS
      INTEGER            INFO, K, L, LDA, LDC, LWORK, M, N
      COMPLEX            A( LDA, * ), C( LDC, * ), TAU( * ),
     $                   WORK( LWORK )
```

Purpose

SORMRZ/CUNMRZ overwrites the general real/complex m-by-n matrix C with

	SIDE = 'L'	SIDE = 'R'	
TRANS = 'N':	Q*C	C*Q	
TRANS = 'T':	Q^T*C	C*Q^T	(*SORMRZ only*)
TRANS = 'C':	Q^H*C	C*Q^H	(*CUNMRZ only*)

where Q is a real/complex orthogonal/unitary matrix defined as the product of k elementary reflectors H_i

$Q = H_1 H_2 \cdots H_k$ (*SORMRZ*)

$Q = H_1{}^H H_2{}^H \cdots H_k{}^H$ (*CUNMRZ*)

as returned by STZRZF/CTZRZF. Q is of order m if SIDE = 'L' and of order n if SIDE = 'R'.

Arguments

SIDE (input) CHARACTER*1
= 'L': apply Q or Q^H from the Left;

= 'R': apply Q or Q^H from the Right.

TRANS (input) CHARACTER*1
= 'N': No transpose, apply Q
= 'T': Transpose, apply Q^T (*SORMRZ only*)
= 'C': Conjugate transpose, apply Q^H (*CUNMRZ only*)

M (input) INTEGER
The number of rows of the matrix C. $M \geq 0$.

N (input) INTEGER
The number of columns of the matrix C. $N \geq 0$.

K (input) INTEGER
The number of elementary reflectors whose product defines the matrix Q.
If SIDE = 'L', $M \geq K \geq 0$;
if SIDE = 'R', $N \geq K \geq 0$.

L (input) INTEGER
The number of columns of the matrix A containing the meaningful part of the Householder reflectors.
If SIDE = 'L', $M \geq L \geq 0$, if SIDE = 'R', $N \geq L \geq 0$.

A (input) REAL/COMPLEX array, dimension
(LDA,M) if SIDE = 'L', or (LDA,N) if SIDE = 'R'
The ith row must contain the vector which defines the elementary reflector H_i, for i = 1,2,...,k, as returned by STZRZF/CTZRZF in the last k rows of its array argument A. A is modified by the routine but restored on exit.

LDA (input) INTEGER
The leading dimension of the array A. LDA $\geq$ max(1,K).

TAU (input) REAL/COMPLEX array, dimension (K)
TAU(i) must contain the scalar factor of the elementary reflector H_i, as returned by STZRZF/CTZRZF.

C (input/output) REAL/COMPLEX array, dimension (LDC,N)
On entry, the m-by-n matrix C.
On exit, C is overwritten by Q*C or Q^H*C or C*Q^H or C*Q.

LDC (input) INTEGER
The leading dimension of the array C. LDC $\geq$ max(1,M).

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK. If SIDE = 'L', LWORK $\geq$ max(1,N); if SIDE = 'R', LWORK $\geq$ max(1,M). For optimum performance LWORK $\geq$ N*NB if SIDE = 'L', and LWORK $\geq$ M*NB if SIDE = 'R', where NB is the optimal blocksize.

If LWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SORMTR/CUNMTR

```
      SUBROUTINE SORMTR( SIDE, UPLO, TRANS, M, N, A, LDA, TAU, C, LDC.
     $                   WORK, LWORK, INFO )
      CHARACTER          SIDE, TRANS, UPLO
      INTEGER            INFO, LDA, LDC, LWORK, M, N
      REAL               A( LDA, * ), C( LDC, * ), TAU( * ),
     $                   WORK( LWORK )

      SUBROUTINE CUNMTR( SIDE, UPLO, TRANS, M, N, A, LDA, TAU, C, LDC,
     $                   WORK, LWORK, INFO )
      CHARACTER          SIDE, TRANS, UPLO
      INTEGER            INFO, LDA, LDC, LWORK, M, N
      COMPLEX            A( LDA, * ), C( LDC, * ), TAU( * ),
     $                   WORK( LWORK )
```

Purpose

SORMTR/CUNMTR overwrites the general real/complex m-by-n matrix C with

	SIDE = 'L'	SIDE = 'R'	
TRANS = 'N':	Q*C	C*Q	
TRANS = 'T':	Q^T*C	C*Q^T	(*SORMTR only*)
TRANS = 'C':	Q^H*C	C*Q^H	(*CUNMTR only*)

where Q is a real/complex orthogonal/unitary matrix of order nq, with nq = m if SIDE = 'L' and nq = n if SIDE = 'R'. Q is defined as the product of nq−1 elementary reflectors, as returned by SSYTRD/CHETRD:

if UPLO = 'U', $Q = H_{nq-1} \cdots H_2 H_1$;

if UPLO = 'L', $Q = H_1 H_2 \cdots H_{nq-1}$.

Arguments

SIDE (input) CHARACTER*1
= 'L': apply Q or Q^H from the Left;
= 'R': apply Q or Q^H from the Right.

UPLO (input) CHARACTER*1
= 'U': Upper triangle of A contains elementary reflectors from SSYTRD/CHETRD;

= 'L': Lower triangle of A contains elementary reflectors from SSYTRD/CHETRD.

TRANS (input) CHARACTER*1
= 'N': No transpose, apply Q
= 'T': Transpose, apply Q^T (*SORMTR only*)
= 'C': Conjugate transpose, apply Q^H (*CUNMTR only*)

M (input) INTEGER
The number of rows of the matrix C. M $\geq$ 0.

N (input) INTEGER
The number of columns of the matrix C. N $\geq$ 0.

A (input) REAL/COMPLEX array, dimension
(LDA,M) if SIDE = 'L', or (LDA,N) if SIDE = 'R'
The vectors which define the elementary reflectors, as returned by SSYTRD/CHETRD.

LDA (input) INTEGER
The leading dimension of the array A.
LDA $\geq$ max(1,M) if SIDE = 'L'; LDA $\geq$ max(1,N) if SIDE = 'R'.

TAU (input) REAL/COMPLEX array, dimension
(M−1) if SIDE = 'L', or (N−1) if SIDE = 'R'
TAU(i) must contain the scalar factor of the elementary reflector H_i, as returned by SSYTRD/CHETRD.

C (input/output) REAL/COMPLEX array, dimension (LDC,N)
On entry, the m-by-n matrix C.
On exit, C is overwritten by Q*C or Q^H*C or C*Q^H or C*Q.

LDC (input) INTEGER
The leading dimension of the array C. LDC $\geq$ max(1,M).

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK. If SIDE = 'L', LWORK $\geq$ max(1,N); if SIDE = 'R', LWORK $\geq$ max(1,M). For optimum performance LWORK $\geq$ N*NB if SIDE = 'L', and LWORK $\geq$ M*NB if SIDE = 'R', where NB is the optimal blocksize.

If LWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SPBCON/CPBCON

```
      SUBROUTINE SPBCON( UPLO, N, KD, AB, LDAB, ANORM, RCOND, WORK,
     $                   IWORK, INFO )
      CHARACTER          UPLO
      INTEGER            INFO, KD, LDAB, N
      REAL               ANORM, RCOND
      INTEGER            IWORK( * )
      REAL               AB( LDAB, * ), WORK( * )

      SUBROUTINE CPBCON( UPLO, N, KD, AB, LDAB, ANORM, RCOND, WORK,
     $                   RWORK, INFO )
      CHARACTER          UPLO
      INTEGER            INFO, KD, LDAB, N
      REAL               ANORM, RCOND
      REAL               RWORK( * )
      COMPLEX            AB( LDAB, * ), WORK( * )
```

Purpose

SPBCON/CPBCON estimates the reciprocal of the condition number (in the 1-norm) of a real/complex symmetric/Hermitian positive definite band matrix using the Cholesky factorization A = U^H*U or A = L*L^H computed by SPBTRF/CPBTRF.

An estimate is obtained for $\|A^{-1}\|$, and the reciprocal of the condition number is computed as RCOND = $1/(\|A\| * \|A^{-1}\|)$.

Arguments

UPLO (input) CHARACTER*1
= 'U': Upper triangular factor stored in AB;
= 'L': Lower triangular factor stored in AB.

N (input) INTEGER
The order of the matrix A. N $\geq$ 0.

KD (input) INTEGER
The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD $\geq$ 0.

AB (input) REAL/COMPLEX array, dimension (LDAB,N)
The triangular factor U or L from the Cholesky factorization A = U^H*U or A = L*L^H of the band matrix A, stored in the first kd+1 rows of the array. The jth column of U or L is stored in the jth column of the array AB as follows:
if UPLO ='U', AB(kd+1+i−j,j) = U(i,j) for max(1,j−kd)$\leq$i$\leq$j;
if UPLO ='L', AB(1+i−j,j) = L(i,j) for j$\leq$i$\leq$min(n,j+kd).

LDAB (input) INTEGER
The leading dimension of the array AB. LDAB $\geq$ KD+1.

ANORM (input) REAL
The 1-norm (or infinity-norm) of the symmetric/Hermitian band matrix

A.

RCOND (output) REAL
The reciprocal of the condition number of the matrix A, computed as RCOND = $1/(\|A\| * \|A^{-1}\|)$.

WORK *SPBCON* (workspace) REAL array, dimension (3*N)
CPBCON (workspace) COMPLEX array, dimension (2*N)

IWORK *SPBCON only* (workspace) INTEGER array, dimension (N)

RWORK *CPBCON only* (workspace) REAL array, dimension (N)

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SPBEQU/CPBEQU

```
      SUBROUTINE SPBEQU( UPLO, N, KD, AB, LDAB, S, SCOND, AMAX, INFO )
      CHARACTER          UPLO
      INTEGER            INFO, KD, LDAB, N
      REAL               AMAX, SCOND
      REAL               AB( LDAB, * ), S( * )

      SUBROUTINE CPBEQU( UPLO, N, KD, AB, LDAB, S, SCOND, AMAX, INFO )
      CHARACTER          UPLO
      INTEGER            INFO, KD, LDAB, N
      REAL               AMAX, SCOND
      REAL               S( * )
      COMPLEX            AB( LDAB, * )
```

Purpose

SPBEQU/CPBEQU computes row and column scalings intended to equilibrate a symmetric/Hermitian positive definite band matrix A and reduce its condition number (with respect to the two-norm). S contains the scale factors, $S(i) = 1/\sqrt{(A(i,i))}$, chosen so that the scaled matrix B with elements B(i,j) = S(i)*A(i,j)*S(j) has ones on the diagonal. This choice of S puts the condition number of B within a factor n of the smallest possible condition number over all possible diagonal scalings.

Arguments

UPLO (input) CHARACTER*1
= 'U': Upper triangular of A is stored;
= 'L': Lower triangular of A is stored.

N (input) INTEGER
The order of the matrix A. N ≥ 0.

KD (input) INTEGER
The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD ≥ 0.

AB (input) REAL/COMPLEX array, dimension (LDAB,N)
The upper or lower triangle of the symmetric/Hermitian band matrix A, stored in the first kd+1 rows of the array. The jth column of A is stored in the jth column of the array AB as follows:
if UPLO = 'U', AB(kd+1+i−j,j) = A(i,j) for max(1,j−kd)≤i≤j;
if UPLO = 'L', AB(1+i−j,j) = A(i,j) for j≤i≤min(n,j+kd).

LDAB (input) INTEGER
The leading dimension of the array A. LDAB ≥ KD+1.

S (output) REAL array, dimension (N)
If INFO = 0, S contains the scale factors for A.

SCOND (output) REAL
If INFO = 0, S contains the ratio of the smallest S(i) to the largest S(i). If SCOND ≥ 0.1 and AMAX is neither too large nor too small, it is not worth scaling by S.

AMAX (output) REAL
Absolute value of largest matrix element. If AMAX is very close to overflow or very close to underflow, the matrix should be scaled.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, the ith diagonal element is nonpositive.

SPBRFS/CPBRFS

```
      SUBROUTINE SPBRFS( UPLO, N, KD, NRHS, AB, LDAB, AFB, LDAFB, B,
     $                   LDB, X, LDX, FERR, BERR, WORK, IWORK, INFO )
      CHARACTER          UPLO
      INTEGER            INFO, KD, LDAB, LDAFB, LDB, LDX, N, NRHS
      INTEGER            IWORK( * )
      REAL               AB( LDAB, * ), AFB( LDAFB, * ), B( LDB, * ),
     $                   BERR( * ), FERR( * ), WORK( * ), X( LDX, * )

      SUBROUTINE CPBRFS( UPLO, N, KD, NRHS, AB, LDAB, AFB, LDAFB, B,
     $                   LDB, X, LDX, FERR, BERR, WORK, RWORK, INFO )
      CHARACTER          UPLO
      INTEGER            INFO, KD, LDAB, LDAFB, LDB, LDX, N, NRHS
      REAL               BERR( * ), FERR( * ), RWORK( * )
      COMPLEX            AB( LDAB, * ), AFB( LDAFB, * ), B( LDB, * ),
     $                   WORK( * ), X( LDX, * )
```

Purpose

SPBRFS/CPBRFS improves the computed solution to a system of linear equations when the coefficient matrix is symmetric/Hermitian positive definite and banded, and provides error bounds and backward error estimates for the solution.

Arguments

UPLO (input) CHARACTER*1
= 'U': Upper triangle of A is stored;
= 'L': Lower triangle of A is stored.

N (input) INTEGER
The order of the matrix A. N $\geq$ 0.

KD (input) INTEGER
The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD $\geq$ 0.

NRHS (input) INTEGER
The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS $\geq$ 0.

AB (input) REAL/COMPLEX array, dimension (LDAB,N)
The upper or lower triangle of the symmetric/Hermitian band matrix A, stored in the first kd+1 rows of the array. The jth column of A is stored in the jth column of the array AB as follows:
if UPLO = 'U', AB(kd+1+i−j,j) = A(i,j) for max(1,j−kd)$\leq$i$\leq$j;
if UPLO = 'L', AB(1+i−j,j) = A(i,j) for j$\leq$i$\leq$min(n,j+kd).

LDAB (input) INTEGER
The leading dimension of the array AB. LDAB $\geq$ KD+1.

AFB (input) REAL/COMPLEX array, dimension (LDAFB,N)
The triangular factor U or L from the Cholesky factorization A = U^H*U or A = L*L^H of the band matrix A as computed by SPBTRF/CPBTRF, in the same storage format as A (see AB).

LDAFB (input) INTEGER
The leading dimension of the array AFB. LDAFB $\geq$ KD+1.

B (input) REAL/COMPLEX array, dimension (LDB,NRHS)
The right hand side matrix B.

LDB (input) INTEGER
The leading dimension of the array B. LDB $\geq$ max(1,N).

X (input/output) REAL/COMPLEX array, dimension (LDX,NRHS)
On entry, the solution matrix X, as computed by SPBTRS/CPBTRS.
On exit, the improved solution matrix X.

LDX (input) INTEGER
The leading dimension of the array X. LDX $\geq$ max(1,N).

FERR (output) REAL array, dimension (NRHS)
The estimated forward error bound for each solution vector X(j) (the jth column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) − XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR (output) REAL array, dimension (NRHS)
The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK *SPBRFS* (workspace) REAL array, dimension (3*N)
CPBRFS (workspace) COMPLEX array, dimension (2*N)

IWORK *SPBRFS only* (workspace) INTEGER array, dimension (N)

RWORK *CPBRFS only* (workspace) REAL array, dimension (N)

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SPBSTF/CPBSTF

```
SUBROUTINE SPBSTF( UPLO, N, KD, AB, LDAB, INFO )
CHARACTER          UPLO
INTEGER            INFO, KD, LDAB, N
REAL               AB( LDAB, * )

SUBROUTINE CPBSTF( UPLO, N, KD, AB, LDAB, INFO )
CHARACTER          UPLO
INTEGER            INFO, KD, LDAB, N
COMPLEX            AB( LDAB, * )
```

Purpose

SPBSTF/CPBSTF computes a split Cholesky factorization of a real/complex symmetric/Hermitian positive definite band matrix A.

This routine is designed to be used in conjunction with SSBGST/CHBGST.

The factorization has the form A = S^H*S where S is a band matrix of the same bandwidth as A and the following structure:

$$S = \begin{pmatrix} U & \\ M & L \end{pmatrix}$$

where U is upper triangular of order m = (n+kd)/2, and L is lower triangular of order n−m.

Arguments

UPLO (input) CHARACTER*1
= 'U': Upper triangle of A is stored;
= 'L': Lower triangle of A is stored.

N (input) INTEGER
The order of the matrix A. $N \geq 0$.

KD (input) INTEGER
The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. $KD \geq 0$.

AB (input/output) REAL/COMPLEX array, dimension (LDAB,N)
On entry, the upper or lower triangle of the symmetric/Hermitian band matrix A, stored in the first kd+1 rows of the array. The jth column of A is stored in the jth column of the array AB as follows:
if UPLO = 'U', $AB(kd+1+i-j,j) = A(i,j)$ for $\max(1,j-kd) \leq i \leq j$;
if UPLO = 'L', $AB(1+i-j,j) = A(i,j)$ for $j \leq i \leq \min(n,j+kd)$.
On exit, if INFO = 0, the factor S from the split Cholesky factorization $A = S^H * S$.

LDAB (input) INTEGER
The leading dimension of the array AB. $LDAB \geq KD+1$.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = -i, the ith argument had an illegal value.
> 0: if INFO = i, the factorization could not be completed, because the updated element a(i,i) was negative; the matrix A is not positive definite.

SPBSV/CPBSV

```
SUBROUTINE SPBSV( UPLO, N, KD, NRHS, AB, LDAB, B, LDB, INFO )
CHARACTER          UPLO
INTEGER            INFO, KD, LDAB, LDB, N, NRHS
REAL               AB( LDAB, * ), B( LDB, * )

SUBROUTINE CPBSV( UPLO, N, KD, NRHS, AB, LDAB, B, LDB, INFO )
CHARACTER          UPLO
INTEGER            INFO, KD, LDAB, LDB, N, NRHS
COMPLEX            AB( LDAB, * ), B( LDB, * )
```

Purpose

SPBSV/CPBSV computes the solution to a real/complex system of linear equations $A*X = B$, where A is an n-by-n symmetric/Hermitian positive definite band matrix and X and B are n-by-nrhs matrices.

The Cholesky decomposition is used to factor A as $A = U^H*U$, if UPLO = 'U', or $A = L*L^H$, if UPLO = 'L', where U is an upper triangular band matrix, and L is a lower triangular band matrix, with the same number of superdiagonals or subdiagonals as A. The factored form of A is then used to solve the system of equations $A*X = B$.

Arguments

UPLO (input) CHARACTER*1
= 'U': Upper triangle of A is stored;
= 'L': Lower triangle of A is stored.

N (input) INTEGER
The number of linear equations, i.e., the order of the matrix A. $N \geq 0$.

KD (input) INTEGER
The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. $KD \geq 0$.

NRHS (input) INTEGER
The number of right hand sides, i.e., the number of columns of the matrix B. $NRHS \geq 0$.

AB (input/output) REAL/COMPLEX array, dimension (LDAB,N)
On entry, the upper or lower triangle of the symmetric/Hermitian band matrix A, stored in the first kd+1 rows of the array. The jth column of A is stored in the jth column of the array AB as follows:
if UPLO = 'U', $AB(kd+1+i-j,j) = A(i,j)$ for $\max(1,j-kd) \leq i \leq j$;
if UPLO = 'L', $AB(1+i-j,j) = A(i,j)$ for $j \leq i \leq \min(n,j+kd)$.
On exit, if INFO = 0, the triangular factor U or L from the Cholesky factorization $A = U^H*U$ or $A = L*L^H$ of the band matrix A, in the same storage format as A.

LDAB (input) INTEGER
The leading dimension of the array AB. $LDAB \geq KD+1$.

B (input/output) REAL/COMPLEX array, dimension (LDB,NRHS)
On entry, the n-by-nrhs right hand side matrix B.
On exit, if INFO = 0, the n-by-nrhs solution matrix X.

LDB (input) INTEGER
The leading dimension of the array B. $LDB \geq \max(1,N)$.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

SPBSVX/CPBSVX

```
      SUBROUTINE SPBSVX( FACT, UPLO, N, KD, NRHS, AB, LDAB, AFB, LDAFB,
     $                   EQUED, S, B, LDB, X, LDX, RCOND, FERR, BERR,
     $                   WORK, IWORK, INFO )
      CHARACTER          EQUED, FACT, UPLO
      INTEGER            INFO, KD, LDAB, LDAFB, LDB, LDX, N, NRHS
      REAL               RCOND
      INTEGER            IWORK( * )
      REAL               AB( LDAB, * ), AFB( LDAFB, * ), B( LDB, * ),
     $                   BERR( * ), FERR( * ), S( * ), WORK( * ),
     $                   X( LDX, * )

      SUBROUTINE CPBSVX( FACT, UPLO, N, KD, NRHS, AB, LDAB, AFB, LDAFB,
     $                   EQUED, S, B, LDB, X, LDX, RCOND, FERR, BERR,
     $                   WORK, RWORK, INFO )
      CHARACTER          EQUED, FACT, UPLO
      INTEGER            INFO, KD, LDAB, LDAFB, LDB, LDX, N, NRHS
      REAL               RCOND
      REAL               BERR( * ), FERR( * ), RWORK( * ), S( * )
      COMPLEX            AB( LDAB, * ), AFB( LDAFB, * ), B( LDB, * ),
     $                   WORK( * ), X( LDX, * )
```

Purpose

SPBSVX/CPBSVX uses the Cholesky factorization $A = U^H*U$ or $A = L*L^H$ to compute the solution to a real/complex system of linear equations $A*X = B$, where A is an n-by-n symmetric/Hermitian positive definite band matrix and X and B are n-by-nrhs matrices.

Error bounds on the solution and a condition estimate are also provided.

Description

The following steps are performed:

1. If FACT = 'E', real scaling factors are computed to equilibrate the system:

   $$\mathrm{diag}(S)*A*\mathrm{diag}(S)*(\mathrm{diag}(S))^{-1}*X = \mathrm{diag}(S)*B$$

   Whether or not the system will be equilibrated depends on the scaling of the matrix A, but if equilibration is used, A is overwritten by diag(S)*A*diag(S) and B by diag(S)*B.

2. If FACT = 'N' or 'E', the Cholesky decomposition is used to factor the matrix A (after equilibration if FACT = 'E') as
   $A = U^H*U$, if UPLO = 'U', or
   $A = L*L^H$, if UPLO = 'L',
   where U is an upper triangular band matrix, and L is a lower triangular band matrix.

3. If the leading i-by-i principal minor is not positive definite, then the routine returns with INFO = i. Otherwise, the factored form of A is used to estimate the condition number of the matrix A. If the reciprocal of the condition number is less than machine precision, INFO = N+1 is returned as a warning, but the routine still goes on to solve for X and compute error bounds as described below.

4. The system of equations is solved for X using the factored form of A.

5. Iterative refinement is applied to improve the computed solution matrix and calculate error bounds and backward error estimates for it.

6. If equilibration was used, the matrix X is premultiplied by diag(S) so that it solves the original system before equilibration.

Arguments

FACT (input) CHARACTER*1
Specifies whether or not the factored form of the matrix A is supplied on entry, and if not, whether the matrix A should be equilibrated before it is factored.
= 'F': On entry, AFB contains the factored form of A. If EQUED = 'Y', the matrix A has been equilibrated with scaling factors given by S. AB and AFB will not be modified.
= 'N': The matrix A will be copied to AFB and factored.
= 'E': The matrix A will be equilibrated if necessary, then copied to AFB and factored.

UPLO (input) CHARACTER*1
= 'U': Upper triangle of A is stored;
= 'L': Lower triangle of A is stored.

N (input) INTEGER
The number of linear equations, i.e., the order of the matrix A. $N \geq 0$.

KD (input) INTEGER
The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. $KD \geq 0$.

NRHS (input) INTEGER
The number of right-hand sides, i.e., the number of columns of the matrices B and X. $NRHS \geq 0$.

AB (input/output) REAL/COMPLEX array, dimension (LDAB,N)
On entry, the upper or lower triangle of the symmetric/Hermitian band matrix A, stored in the first kd+1 rows of the array, except if FACT = 'F' and EQUED = 'Y', then A must contain the equilibrated matrix diag(S)*A*diag(S). The jth column of A is stored in the jth column of the array AB as follows:
if UPLO = 'U', $AB(kd+1+i-j,j) = A(i,j)$ for $\max(1,j-kd) \leq i \leq j$;
if UPLO = 'L', $AB(1+i-j,j) = A(i,j)$ for $j \leq i \leq \min(n,j+kd)$.
On exit, if FACT = 'E' and EQUED = 'Y', A is overwritten by diag(S)*A*diag(S).

LDAB (input) INTEGER
The leading dimension of the array A. $LDAB \geq KD+1$.

AFB (input or output) REAL/COMPLEX array, dimension (LDAFB,N)
If FACT = 'F', then AFB is an input argument and on entry contains the triangular factor U or L from the Cholesky factorization $A = U^H*U$ or $A = L*L^H$ of the band matrix A, in the same storage format as A (see AB). If EQUED = 'Y', then AFB is the factored form of the equilibrated matrix A.
If FACT = 'N', then AFB is an output argument and on exit returns the triangular factor U or L from the Cholesky factorization $A = U^H*U$ or $A = L*L^H$.
If FACT = 'E', then AFB is an output argument and on exit returns the triangular factor U or L from the Cholesky factorization $A = U^H*U$ or $A = L*L^H$ of the equilibrated matrix A (see the description of A for the form of the equilibrated matrix).

LDAFB (input) INTEGER
The leading dimension of the array AFB. LDAFB $\geq$ KD+1.

EQUED (input or output) CHARACTER*1
Specifies the form of equilibration that was done.
= 'N': No equilibration (always true if FACT = 'N').
= 'Y': Equilibration was done, i.e., A has been replaced by diag(S)*A*diag(S).
EQUED is an input argument if FACT = 'F'; otherwise, it is an output argument.

S (input or output) REAL array, dimension (N)
The scale factors for A; not accessed if EQUED = 'N'. S is an input argument if FACT = 'F'; otherwise, S is an output argument. If FACT = 'F' and EQUED = 'Y', each element of S must be positive.

B (input/output) REAL/COMPLEX array, dimension (LDB,NRHS)
On entry, the n-by-nrhs right hand side matrix B.
On exit, if EQUED = 'N', B is not modified; if EQUED = 'Y', B is overwritten by diag(S)*B.

LDB (input) INTEGER
The leading dimension of the array B. LDB $\geq$ max(1,N).

X (output) REAL/COMPLEX array, dimension (LDX,NRHS)
If INFO = 0 or INFO = N+1, the N-by-NRHS solution matrix X to the original system of equations. Note that if EQUED = 'Y', A and B are modified on exit, and the solution to the equilibrated system is $(\mathrm{diag}(S))^{-1}*X$.

LDX (input) INTEGER
The leading dimension of the array X. LDX $\geq$ max(1,N).

RCOND (output) REAL
The estimate of the reciprocal condition number of the matrix A after equilibration (if done). If RCOND is less than the machine precision (in particular, if RCOND = 0), the matrix is singular to working precision. This condition is indicated by a return code of INFO > 0.

FERR (output) REAL array, dimension (NRHS)
The estimated forward error bound for each solution vector X(j) (the jth column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) − XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR (output) REAL array, dimension (NRHS)
The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK *SPBSVX* (workspace) REAL array, dimension (3*N)
CPBSVX (workspace) COMPLEX array, dimension (2*N)

IWORK *SPBSVX only* (workspace) INTEGER array, dimension (N)

RWORK *CPBSVX only* (workspace) REAL array, dimension (N)

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, and i is
$\leq$ N: the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed. RCOND = 0 is returned.
= N+1: U is nonsingular, but RCOND is less than machine precision, meaning that the matrix is singular to working precision. Nevertheless, the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the value of RCOND would suggest.

SPBTRF/CPBTRF

```
SUBROUTINE SPBTRF( UPLO, N, KD, AB, LDAB, INFO )
CHARACTER          UPLO
INTEGER            INFO, KD, LDAB, N
REAL               AB( LDAB, * )

SUBROUTINE CPBTRF( UPLO, N, KD, AB, LDAB, INFO )
CHARACTER          UPLO
INTEGER            INFO, KD, LDAB, N
COMPLEX            AB( LDAB, * )
```

Purpose

SPBTRF/CPBTRF computes the Cholesky factorization of a real/complex symmetric/Hermitian positive definite band matrix A.

The factorization has the form $A = U^H*U$, if UPLO = 'U', or $A = L*L^H$, if UPLO = 'L', where U is an upper triangular matrix and L is lower triangular.

Arguments

UPLO (input) CHARACTER*1
= 'U': Upper triangle of A is stored;
= 'L': Lower triangle of A is stored.

N (input) INTEGER
The order of the matrix A. $N \geq 0$.

KD (input) INTEGER
The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. $KD \geq 0$.

AB (input/output) REAL/COMPLEX array, dimension (LDAB,N)
On entry, the upper or lower triangle of the symmetric/Hermitian band matrix A, stored in the first kd+1 rows of the array. The jth column of A is stored in the jth column of the array AB as follows:
if UPLO = 'U', $AB(kd+1+i-j,j) = A(i,j)$ for $\max(1,j-kd) \leq i \leq j$;
if UPLO = 'L', $AB(1+i-j,j) = A(i,j)$ for $j \leq i \leq \min(n,j+kd)$.
On exit, if INFO = 0, the triangular factor U or L from the Cholesky factorization $A = U^H*U$ or $A = L*L^H$ of the band matrix A, in the same storage format as A.

LDAB (input) INTEGER
The leading dimension of the array AB. $LDAB \geq KD+1$.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, the leading minor of order i is not positive definite, and the factorization could not be completed.

SPBTRS/CPBTRS

```
SUBROUTINE SPBTRS( UPLO, N, KD, NRHS, AB, LDAB, B, LDB, INFO )
CHARACTER          UPLO
INTEGER            INFO, KD, LDAB, LDB, N, NRHS
REAL               AB( LDAB, * ), B( LDB, * )

SUBROUTINE CPBTRS( UPLO, N, KD, NRHS, AB, LDAB, B, LDB, INFO )
CHARACTER          UPLO
INTEGER            INFO, KD, LDAB, LDB, N, NRHS
COMPLEX            AB( LDAB, * ), B( LDB, * )
```

Purpose

SPBTRS/CPBTRS solves a system of linear equations $A*X = B$ with a symmetric/Hermitian positive definite band matrix A using the Cholesky factorization $A = U^H*U$ or $A = L*L^H$ computed by SPBTRF/CPBTRF.

Arguments

UPLO (input) CHARACTER*1
= 'U': Upper triangular factor stored in AB;
= 'L': Lower triangular factor stored in AB.

N (input) INTEGER
The order of the matrix A. $N \geq 0$.

KD (input) INTEGER
The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. $KD \geq 0$.

NRHS (input) INTEGER
The number of right hand sides, i.e., the number of columns of the matrix B. $NRHS \geq 0$.

AB (input) REAL/COMPLEX array, dimension (LDAB,N)
The triangular factor U or L from the Cholesky factorization $A = U^H*U$ or $A = L*L^H$ of the band matrix A, stored in the first kd+1 rows of the array. The jth column of U or L is stored in the jth column of the array AB as follows:
if UPLO ='U', $AB(kd+1+i-j,j) = U(i,j)$ for $\max(1,j-kd) \leq i \leq j$;
if UPLO ='L', $AB(1+i-j,j) = L(i,j)$ for $j \leq i \leq \min(n,j+kd)$.

LDAB (input) INTEGER
The leading dimension of the array AB. $LDAB \geq KD+1$.

B (input/output) REAL/COMPLEX array, dimension (LDB,NRHS)
On entry, the right hand side matrix B.
On exit, the solution matrix X.

LDB (input) INTEGER
The leading dimension of the array B. $LDB \geq \max(1,N)$.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SPOCON/CPOCON

```
SUBROUTINE SPOCON( UPLO, N, A, LDA, ANORM, RCOND, WORK, IWORK,
$                  INFO )
CHARACTER          UPLO
INTEGER            INFO, LDA, N
REAL               ANORM, RCOND
INTEGER            IWORK( * )
REAL               A( LDA, * ), WORK( * )
```

```
      SUBROUTINE CPOCON( UPLO, N, A, LDA, ANORM, RCOND, WORK, RWORK,
     $                   INFO )
      CHARACTER          UPLO
      INTEGER            INFO, LDA, N
      REAL               ANORM, RCOND
      REAL               RWORK( * )
      COMPLEX            A( LDA, * ), WORK( * )
```

Purpose

SPOCON/CPOCON estimates the reciprocal of the condition number (in the 1-norm) of a real/complex symmetric/Hermitian positive definite matrix using the Cholesky factorization $A = U^H*U$ or $A = L*L^H$ computed by SPOTRF/CPOTRF.

An estimate is obtained for $\|A^{-1}\|$, and the reciprocal of the condition number is computed as $RCOND = 1/(\|A\| * \|A^{-1}\|)$.

Arguments

UPLO (input) CHARACTER*1
= 'U': Upper triangle of A is stored;
= 'L': Lower triangle of A is stored.

N (input) INTEGER
The order of the matrix A. $N \geq 0$.

A (input) REAL/COMPLEX array, dimension (LDA,N)
The triangular factor U or L from the Cholesky factorization $A = U^H*U$ or $A = L*L^H$, as computed by SPOTRF/CPOTRF.

LDA (input) INTEGER
The leading dimension of the array A. $LDA \geq \max(1,N)$.

ANORM (input) REAL
The 1-norm (or infinity-norm) of the symmetric/Hermitian matrix A.

RCOND (output) REAL
The reciprocal of the condition number of the matrix A, computed as $RCOND = 1/(\|A\| * \|A^{-1}\|)$.

WORK *SPOCON* (workspace) REAL array, dimension (3*N)
CPOCON (workspace) COMPLEX array, dimension (2*N)

IWORK *SPOCON only* (workspace) INTEGER array, dimension (N)

RWORK *CPOCON only* (workspace) REAL array, dimension (N)

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SPOEQU/CPOEQU

```
      SUBROUTINE SPOEQU( N, A, LDA, S, SCOND, AMAX, INFO )
      INTEGER            INFO, LDA, N
      REAL               AMAX, SCOND
      REAL               A( LDA, * ), S( * )

      SUBROUTINE CPOEQU( N, A, LDA, S, SCOND, AMAX, INFO )
      INTEGER            INFO, LDA, N
      REAL               AMAX, SCOND
      REAL               S( * )
      COMPLEX            A( LDA, * )
```

Purpose

SPOEQU/CPOEQU computes row and column scalings intended to equilibrate a symmetric/Hermitian positive definite matrix A and reduce its condition number (with respect to the two-norm). S contains the scale factors, $S(i) = 1/\sqrt{(A(i,i))}$, chosen so that the scaled matrix B with elements $B(i,j) = S(i)*A(i,j)*S(j)$ has ones on the diagonal. This choice of S puts the condition number of B within a factor n of the smallest possible condition number over all possible diagonal scalings.

Arguments

N (input) INTEGER
The order of the matrix A. $N \geq 0$.

A (input) REAL/COMPLEX array, dimension (LDA,N)
The n-by-n symmetric/Hermitian positive definite matrix whose scaling factors are to be computed. Only the diagonal elements of A are referenced.

LDA (input) INTEGER
The leading dimension of the array A. $LDA \geq \max(1,N)$.

S (output) REAL array, dimension (N)
If INFO = 0, S contains the scale factors for A.

SCOND (output) REAL
If INFO = 0, S contains the ratio of the smallest S(i) to the largest S(i). If $SCOND \geq 0.1$ and AMAX is neither too large nor too small, it is not worth scaling by S.

AMAX (output) REAL
Absolute value of largest matrix element. If AMAX is very close to overflow or very close to underflow, the matrix should be scaled.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, the ith diagonal element is nonpositive.

SPORFS/CPORFS

```
      SUBROUTINE SPORFS( UPLO, N, NRHS, A, LDA, AF, LDAF, B, LDB, X,
     $                   LDX, FERR, BERR, WORK, IWORK, INFO )
      CHARACTER          UPLO
      INTEGER            INFO, LDA, LDAF, LDB, LDX, N, NRHS
      INTEGER            IWORK( * )
      REAL               A( LDA, * ), AF( LDAF, * ), B( LDB, * ),
     $                   BERR( * ), FERR( * ), WORK( * ), X( LDX, * )

      SUBROUTINE CPORFS( UPLO, N, NRHS, A, LDA, AF, LDAF, B, LDB, X,
     $                   LDX, FERR, BERR, WORK, RWORK, INFO )
      CHARACTER          UPLO
      INTEGER            INFO, LDA, LDAF, LDB, LDX, N, NRHS
      REAL               BERR( * ), FERR( * ), RWORK( * )
      COMPLEX            A( LDA, * ), AF( LDAF, * ), B( LDB, * ),
     $                   WORK( * ), X( LDX, * )
```

Purpose

SPORFS/CPORFS improves the computed solution to a system of linear equations when the coefficient matrix is symmetric/Hermitian positive definite, and provides error bounds and backward error estimates for the solution.

Arguments

UPLO (input) CHARACTER*1
= 'U': Upper triangle of A is stored;
= 'L': Lower triangle of A is stored.

N (input) INTEGER
The order of the matrix A. $N \geq 0$.

NRHS (input) INTEGER
The number of right hand sides, i.e., the number of columns of the matrices B and X. $NRHS \geq 0$.

A (input) REAL/COMPLEX array, dimension (LDA,N)
The symmetric/Hermitian matrix A. If UPLO = 'U', the leading n-by-n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n-by-n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

LDA (input) INTEGER
The leading dimension of the array A. $LDA \geq \max(1,N)$.

AF (input) REAL/COMPLEX array, dimension (LDAF,N)
The triangular factor U or L from the Cholesky factorization $A = U^H*U$ or $A = L*L^H$, as computed by SPOTRF/CPOTRF.

LDAF (input) INTEGER
The leading dimension of the array AF. $LDAF \geq \max(1,N)$.

B (input) REAL/COMPLEX array, dimension (LDB,NRHS)
The right hand side matrix B.

LDB (input) INTEGER
The leading dimension of the array B. $LDB \geq \max(1,N)$.

X (input/output) REAL/COMPLEX array, dimension (LDX,NRHS)
On entry, the solution matrix X, as computed by SPOTRS/CPOTRS.
On exit, the improved solution matrix X.

LDX (input) INTEGER
The leading dimension of the array X. $LDX \geq \max(1,N)$.

FERR (output) REAL array, dimension (NRHS)
The estimated forward error bound for each solution vector X(j) (the jth column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) − XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR (output) REAL array, dimension (NRHS)
The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK *SPORFS* (workspace) REAL array, dimension (3*N)
CPORFS (workspace) COMPLEX array, dimension (2*N)

IWORK *SPORFS only* (workspace) INTEGER array, dimension (N)

RWORK *CPORFS only* (workspace) REAL array, dimension (N)

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SPOSV/CPOSV

```
      SUBROUTINE SPOSV( UPLO, N, NRHS, A, LDA, B, LDB, INFO )
      CHARACTER          UPLO
      INTEGER            INFO, LDA, LDB, N, NRHS
      REAL               A( LDA, * ), B( LDB, * )

      SUBROUTINE CPOSV( UPLO, N, NRHS, A, LDA, B, LDB, INFO )
      CHARACTER          UPLO
      INTEGER            INFO, LDA, LDB, N, NRHS
      COMPLEX            A( LDA, * ), B( LDB, * )
```

Purpose

SPOSV/CPOSV computes the solution to a real/complex system of linear equations A*X = B, where A is an n-by-n symmetric/Hermitian positive definite matrix and X and B are n-by-nrhs matrices.

The Cholesky decomposition is used to factor A as $A = U^H*U$, if UPLO = 'U', or $A = L*L^H$, if UPLO = 'L', where U is an upper triangular matrix and L is a lower triangular matrix. The factored form of A is then used to solve the system of equations A*X = B.

Arguments

UPLO (input) CHARACTER*1
= 'U': Upper triangle of A is stored;
= 'L': Lower triangle of A is stored.

N (input) INTEGER
The number of linear equations, i.e., the order of the matrix A. $N \geq 0$.

NRHS (input) INTEGER
The number of right hand sides, i.e., the number of columns of the matrix B. $NRHS \geq 0$.

A (input/output) REAL/COMPLEX array, dimension (LDA,N)
On entry, the symmetric/Hermitian matrix A. If UPLO = 'U', the leading n-by-n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n-by-n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.
On exit, if INFO = 0, the factor U or L from the Cholesky factorization $A = U^H*U$ or $A = L*L^H$.

LDA (input) INTEGER
The leading dimension of the array A. $LDA \geq \max(1,N)$.

B (input/output) REAL/COMPLEX array, dimension (LDB,NRHS)
On entry, the n-by-nrhs right hand side matrix B.
On exit, if INFO = 0, the n-by-nrhs solution matrix X.

LDB (input) INTEGER
The leading dimension of the array B. $LDB \geq \max(1,N)$.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

SPOSVX/CPOSVX

```
      SUBROUTINE SPOSVX( FACT, UPLO, N, NRHS, A, LDA, AF, LDAF, EQUED,
     $                   S, B, LDB, X, LDX, RCOND, FERR, BERR, WORK,
     $                   IWORK, INFO )
      CHARACTER          EQUED, FACT, UPLO
      INTEGER            INFO, LDA, LDAF, LDB, LDX, N, NRHS
      REAL               RCOND
      INTEGER            IWORK( * )
      REAL               A( LDA, * ), AF( LDAF, * ), B( LDB, * ),
     $                   BERR( * ), FERR( * ), S( * ), WORK( * ),
     $                   X( LDX, * )

      SUBROUTINE CPOSVX( FACT, UPLO, N, NRHS, A, LDA, AF, LDAF, EQUED,
     $                   S, B, LDB, X, LDX, RCOND, FERR, BERR, WORK,
     $                   RWORK, INFO )
      CHARACTER          EQUED, FACT, UPLO
      INTEGER            INFO, LDA, LDAF, LDB, LDX, N, NRHS
      REAL               RCOND
      REAL               BERR( * ), FERR( * ), RWORK( * ), S( * )
      COMPLEX            A( LDA, * ), AF( LDAF, * ), B( LDB, * ),
     $                   WORK( * ), X( LDX, * )
```

Purpose

SPOSVX/CPOSVX uses the Cholesky factorization $A = U^H*U$ or $A = L*L^H$ to compute the solution to a real/complex system of linear equations A*X = B, where A is an n-by-n symmetric/Hermitian positive definite matrix and X and B are n-by-nrhs matrices.

Error bounds on the solution and a condition estimate are also provided.

Description

The following steps are performed:

1. If FACT = 'E', real scaling factors are computed to equilibrate the system:

   $$\mathrm{diag}(S)*A*\mathrm{diag}(S)*(\mathrm{diag}(S))^{-1}*X = \mathrm{diag}(S)*B$$

   Whether or not the system will be equilibrated depends on the scaling of the matrix A, but if equilibration is used, A is overwritten by diag(S)*A*diag(S) and B by diag(S)*B.

2. If FACT = 'N' or 'E', the Cholesky decomposition is used to factor the matrix A (after equilibration if FACT = 'E') as
   $A = U^H*U$, if UPLO = 'U', or
   $A = L*L^H$, if UPLO = 'L',
   where U is an upper triangular matrix and L is a lower triangular matrix.

3. If the leading i-by-i principal minor is not positive definite, then the routine returns with INFO = i. Otherwise, the factored form of A is used to estimate

the condition number of the matrix A. If the reciprocal of the condition number is less than machine precision, INFO = N+1 is returned as a warning, but the routine still goes on to solve for X and compute error bounds as described below.

4. The system of equations is solved for X using the factored form of A.

5. Iterative refinement is applied to improve the computed solution matrix and calculate error bounds and backward error estimates for it.

6. If equilibration was used, the matrix X is premultiplied by diag(S) so that it solves the original system before equilibration.

Arguments

FACT (input) CHARACTER*1
Specifies whether or not the factored form of the matrix A is supplied on entry, and if not, whether the matrix A should be equilibrated before it is factored.
= 'F': On entry, AF contains the factored form of A. If EQUED = 'Y', the matrix A has been equilibrated with scaling factors given by S. A and AF will not be modified.
= 'N': The matrix A will be copied to AF and factored.
= 'E': The matrix A will be equilibrated if necessary, then copied to AF and factored.

UPLO (input) CHARACTER*1
= 'U': Upper triangle of A is stored;
= 'L': Lower triangle of A is stored.

N (input) INTEGER
The number of linear equations, i.e., the order of the matrix A. N $\geq$ 0.

NRHS (input) INTEGER
The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS $\geq$ 0.

A (input/output) REAL/COMPLEX array, dimension (LDA,N)
On entry, the symmetric/Hermitian matrix A, except if FACT = 'F' and EQUED = 'Y', then A must contain the equilibrated matrix diag(S)*A*diag(S). If UPLO = 'U', the leading n-by-n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n-by-n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced. A is not modified if FACT = 'F' or 'N', or if FACT = 'E' and EQUED = 'N' on exit.
On exit, if FACT = 'E' and EQUED = 'Y', A is overwritten by diag(S)*A*diag(S).

LDA (input) INTEGER
The leading dimension of the array A. LDA $\geq$ max(1,N).

AF (input or output) REAL/COMPLEX array, dimension (LDAF,N)
If FACT = 'F', then AF is an input argument and on entry contains the triangular factor U or L from the Cholesky factorization A = U^H*U or A = L*L^H, in the same storage format as A. If EQUED $\neq$ 'N', then AF is the factored form of the equilibrated matrix diag(S)*A*diag(S).
If FACT = 'N', then AF is an output argument and on exit returns the triangular factor U or L from the Cholesky factorization A = U^H*U or A = L*L^H of the original matrix A.
If FACT = 'E', then AF is an output argument and on exit returns the triangular factor U or L from the Cholesky factorization A = U^H*U or A = L*L^H of the equilibrated matrix A (see the description of A for the form of the equilibrated matrix).

LDAF (input) INTEGER
The leading dimension of the array AF. LDAF $\geq$ max(1,N).

EQUED (input or output) CHARACTER*1
Specifies the form of equilibration that was done.
= 'N': No equilibration (always true if FACT = 'N').
= 'Y': Equilibration was done, i.e., A has been replaced by diag(S)*A*diag(S).
EQUED is an input argument if FACT = 'F'; otherwise, it is an output argument.

S (input or output) REAL array, dimension (N)
The scale factors for A; not accessed if EQUED = 'N'. S is an input argument if FACT = 'F'; otherwise, S is an output argument. If FACT = 'F' and EQUED = 'Y', each element of S must be positive.

B (input/output) REAL/COMPLEX array, dimension (LDB,NRHS)
On entry, the n-by-nrhs right hand side matrix B.
On exit, if EQUED = 'N', B is not modified; if EQUED = 'Y', B is overwritten by diag(S)*B.

LDB (input) INTEGER
The leading dimension of the array B. LDB $\geq$ max(1,N).

X (output) REAL/COMPLEX array, dimension (LDX,NRHS)
If INFO = 0 or INFO = N+1, the N-by-NRHS solution matrix X to the original system of equations. Note that if EQUED = 'Y', A and B are modified on exit, and the solution to the equilibrated system is $(\mathrm{diag}(S))^{-1}$*X.

LDX (input) INTEGER
The leading dimension of the array X. LDX $\geq$ max(1,N).

RCOND (output) REAL
The estimate of the reciprocal condition number of the matrix A after equilibration (if done). If RCOND is less than the machine precision (in particular, if RCOND = 0), the matrix is singular to working precision. This condition is indicated by a return code of INFO > 0.

FERR (output) REAL array, dimension (NRHS)
The estimated forward error bound for each solution vector X(j) (the jth column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the

magnitude of the largest element in (X(j) − XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR (output) REAL array, dimension (NRHS)
The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK *SPOSVX* (workspace) REAL array, dimension (3*N)
CPOSVX (workspace) COMPLEX array, dimension (2*N)

IWORK *SPOSVX only* (workspace) INTEGER array, dimension (N)

RWORK *CPOSVX only* (workspace) REAL array, dimension (N)

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, and i is
≤ N: the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution and error bounds could not be computed. RCOND = 0 is returned.
= N+1: U is nonsingular, but RCOND is less than machine precision, meaning that the matrix is singular to working precision. Nevertheless, the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the value of RCOND would suggest.

SPOTRF/CPOTRF

```
SUBROUTINE SPOTRF( UPLO, N, A, LDA, INFO )
CHARACTER          UPLO
INTEGER            INFO, LDA, N
REAL               A( LDA, * )

SUBROUTINE CPOTRF( UPLO, N, A, LDA, INFO )
CHARACTER          UPLO
INTEGER            INFO, LDA, N
COMPLEX            A( LDA, * )
```

Purpose

SPOTRF/CPOTRF computes the Cholesky factorization of a real/complex symmetric/Hermitian positive definite matrix A.

The factorization has the form $A = U^H*U$, if UPLO = 'U', or $A = L*L^H$, if UPLO = 'L', where U is an upper triangular matrix and L is lower triangular.

Arguments

UPLO (input) CHARACTER*1
= 'U': Upper triangle of A is stored;
= 'L': Lower triangle of A is stored.

N (input) INTEGER
The order of the matrix A. N ≥ 0.

A (input/output) REAL/COMPLEX array, dimension (LDA,N)
On entry, the symmetric/Hermitian matrix A. If UPLO = 'U', the leading n-by-n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n-by-n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.
On exit, if INFO = 0, the factor U or L from the Cholesky factorization $A = U^H*U$ or $A = L*L^H$.

LDA (input) INTEGER
The leading dimension of the array A. LDA ≥ max(1,N).

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, the leading minor of order i is not positive definite, and the factorization could not be completed.

SPOTRI/CPOTRI

```
SUBROUTINE SPOTRI( UPLO, N, A, LDA, INFO )
CHARACTER          UPLO
INTEGER            INFO, LDA, N
REAL               A( LDA, * )

SUBROUTINE CPOTRI( UPLO, N, A, LDA, INFO )
CHARACTER          UPLO
INTEGER            INFO, LDA, N
COMPLEX            A( LDA, * )
```

Purpose

SPOTRI/CPOTRI computes the inverse of a real/complex symmetric/Hermitian positive definite matrix A using the Cholesky factorization $A = U^H*U$ or $A = L*L^H$ computed by SPOTRF/CPOTRF.

Arguments

UPLO (input) CHARACTER*1
= 'U': Upper triangle of A is stored;

= 'L': Lower triangle of A is stored.

N (input) INTEGER
The order of the matrix A. N $\geq$ 0.

A (input/output) REAL/COMPLEX array, dimension (LDA,N)
On entry, the triangular factor U or L from the Cholesky factorization $A = U^H*U$ or $A = L*L^H$, as computed by SPOTRF/CPOTRF.
On exit, the upper or lower triangle of the (symmetric)/(Hermitian) inverse of A, overwriting the input factor U or L.

LDA (input) INTEGER
The leading dimension of the array A. LDA $\geq$ max(1,N).

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, the (i,i) element of the factor U or L is zero, and the inverse could not be computed.

SPOTRS/CPOTRS

```
SUBROUTINE SPOTRS( UPLO, N, NRHS, A, LDA, B, LDB, INFO )
CHARACTER          UPLO
INTEGER            INFO, LDA, LDB, N, NRHS
REAL               A( LDA, * ), B( LDB, * )

SUBROUTINE CPOTRS( UPLO, N, NRHS, A, LDA, B, LDB, INFO )
CHARACTER          UPLO
INTEGER            INFO, LDA, LDB, N, NRHS
COMPLEX            A( LDA, * ), B( LDB, * )
```

Purpose

SPOTRS/CPOTRS solves a system of linear equations A*X = B with a symmetric/Hermitian positive definite matrix A using the Cholesky factorization $A = U^H*U$ or $A = L*L^H$ computed by SPOTRF/CPOTRF.

Arguments

UPLO (input) CHARACTER*1
= 'U': Upper triangle of A is stored;
= 'L': Lower triangle of A is stored.

N (input) INTEGER
The order of the matrix A. N $\geq$ 0.

NRHS (input) INTEGER
The number of right hand sides, i.e., the number of columns of the matrix B. NRHS $\geq$ 0.

A (input) REAL/COMPLEX array, dimension (LDA,N)
The triangular factor U or L from the Cholesky factorization $A = U^H*U$ or $A = L*L^H$, as computed by SPOTRF/CPOTRF.

LDA (input) INTEGER
The leading dimension of the array A. LDA $\geq$ max(1,N).

B (input/output) REAL/COMPLEX array, dimension (LDB,NRHS)
On entry, the right hand side matrix B.
On exit, the solution matrix X.

LDB (input) INTEGER
The leading dimension of the array B. LDB $\geq$ max(1,N).

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SPPCON/CPPCON

```
SUBROUTINE SPPCON( UPLO, N, AP, ANORM, RCOND, WORK, IWORK, INFO )
CHARACTER          UPLO
INTEGER            INFO, N
REAL               ANORM, RCOND
INTEGER            IWORK( * )
REAL               AP( * ), WORK( * )

SUBROUTINE CPPCON( UPLO, N, AP, ANORM, RCOND, WORK, RWORK, INFO )
CHARACTER          UPLO
INTEGER            INFO, N
REAL               ANORM, RCOND
REAL               RWORK( * )
COMPLEX            AP( * ), WORK( * )
```

Purpose

SPPCON/CPPCON estimates the reciprocal of the condition number (in the 1-norm) of a real/complex symmetric/Hermitian positive definite packed matrix using the Cholesky factorization $A = U^H*U$ or $A = L*L^H$ computed by SPPTRF/CPPTRF.

An estimate is obtained for $||A^{-1}||$, and the reciprocal of the condition number is computed as RCOND = $1/(||A|| * ||A^{-1}||)$.

Arguments

UPLO (input) CHARACTER*1
= 'U': Upper triangle of A is stored;
= 'L': Lower triangle of A is stored.

N (input) INTEGER
The order of the matrix A. $N \geq 0$.

AP (input) REAL/COMPLEX array, dimension (N*(N+1)/2)
The triangular factor U or L from the Cholesky factorization $A = U^H*U$ or $A = L*L^H$, packed columnwise in a linear array. The jth column of U or L is stored in the array AP as follows:
if UPLO = 'U', AP(i + (j−1)*j/2) = U(i,j) for $1 \leq i \leq j$;
if UPLO = 'L', AP(i + (j−1)*(2*n−j)/2) = L(i,j) for $j \leq i \leq n$.

ANORM (input) REAL
The 1-norm (or infinity-norm) of the symmetric/Hermitian matrix A.

RCOND (output) REAL
The reciprocal of the condition number of the matrix A, computed as $RCOND = 1/(\|A\| * \|A^{-1}\|)$.

WORK *SPPCON* (workspace) REAL array, dimension (3*N)
CPPCON (workspace) COMPLEX array, dimension (2*N)

IWORK *SPPCON only* (workspace) INTEGER array, dimension (N)

RWORK *CPPCON only* (workspace) REAL array, dimension (N)

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SPPEQU/CPPEQU

```
      SUBROUTINE SPPEQU( UPLO, N, AP, S, SCOND, AMAX, INFO )
      CHARACTER          UPLO
      INTEGER            INFO, N
      REAL               AMAX, SCOND
      REAL               AP( * ), S( * )

      SUBROUTINE CPPEQU( UPLO, N, AP, S, SCOND, AMAX, INFO )
      CHARACTER          UPLO
      INTEGER            INFO, N
      REAL               AMAX, SCOND
      REAL               S( * )
      COMPLEX            AP( * )
```

Purpose

SPPEQU/CPPEQU computes row and column scalings intended to equilibrate a symmetric/Hermitian positive definite matrix A in packed storage and reduce its condition number (with respect to the two-norm). S contains the scale factors, $S(i) = 1/\sqrt{(A(i,i))}$, chosen so that the scaled matrix B with elements $B(i,j) = S(i)*A(i,j)*S(j)$ has ones on the diagonal. This choice of S puts the condition number of B within a factor n of the smallest possible condition number over all possible diagonal scalings.

Arguments

UPLO (input) CHARACTER*1
= 'U': Upper triangle of A is stored;
= 'L': Lower triangle of A is stored.

N (input) INTEGER
The order of the matrix A. $N \geq 0$.

AP (input) REAL/COMPLEX array, dimension (N*(N+1)/2)
The upper or lower triangle of the symmetric/Hermitian matrix A, packed columnwise in a linear array. The jth column of A is stored in the array AP as follows:
if UPLO = 'U', AP(i + (j−1)*j/2) = A(i,j) for $1 \leq i \leq j$;
if UPLO = 'L', AP(i + (j−1)*(2*n−j)/2) = A(i,j) for $j \leq i \leq n$.

S (output) REAL array, dimension (N)
If INFO = 0, S contains the scale factors for A.

SCOND (output) REAL
If INFO = 0, S contains the ratio of the smallest S(i) to the largest S(i). If SCOND $\geq$ 0.1 and AMAX is neither too large nor too small, it is not worth scaling by S.

AMAX (output) REAL
Absolute value of largest matrix element. If AMAX is very close to overflow or very close to underflow, the matrix should be scaled.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, the ith diagonal element is nonpositive.

SPPRFS/CPPRFS

```
      SUBROUTINE SPPRFS( UPLO, N, NRHS, AP, AFP, B, LDB, X, LDX, FERR,
     $                   BERR, WORK, IWORK, INFO )
      CHARACTER          UPLO
      INTEGER            INFO, LDB, LDX, N, NRHS
      INTEGER            IWORK( * )
      REAL               AFP( * ), AP( * ), B( LDB, * ), BERR( * ),
     $                   FERR( * ), WORK( * ), X( LDX, * )

      SUBROUTINE CPPRFS( UPLO, N, NRHS, AP, AFP, B, LDB, X, LDX, FERR,
     $                   BERR, WORK, RWORK, INFO )
      CHARACTER          UPLO
      INTEGER            INFO, LDB, LDX, N, NRHS
      REAL               BERR( * ), FERR( * ), RWORK( * )
      COMPLEX            AFP( * ), AP( * ), B( LDB, * ), WORK( * ),
     $                   X( LDX, * )
```

Purpose

SPPRFS/CPPRFS improves the computed solution to a system of linear equations when the coefficient matrix is symmetric/Hermitian positive definite and packed, and provides error bounds and backward error estimates for the solution.

Arguments

UPLO (input) CHARACTER*1
= 'U': Upper triangle of A is stored;
= 'L': Lower triangle of A is stored.

N (input) INTEGER
The order of the matrix A. $N \geq 0$.

NRHS (input) INTEGER
The number of right hand sides, i.e., the number of columns of the matrices B and X. $NRHS \geq 0$.

AP (input) REAL/COMPLEX array, dimension (N*(N+1)/2)
The upper or lower triangle of the symmetric/Hermitian matrix A, packed columnwise in a linear array. The jth column of A is stored in the array AP as follows:
if UPLO = 'U', $AP(i + (j-1)*j/2) = A(i,j)$ for $1 \leq i \leq j$;
if UPLO = 'L', $AP(i + (j-1)*(2*n-j)/2) = A(i,j)$ for $j \leq i \leq n$.

AFP (input) REAL/COMPLEX array, dimension (N*(N+1)/2)
The triangular factor U or L from the Cholesky factorization $A = U^H*U$ or $A = L*L^H$, as computed by SPPTRF/CPPTRF, packed columnwise in a linear array in the same format as A (see AP).

B (input) REAL/COMPLEX array, dimension (LDB,NRHS)
The right hand side matrix B.

LDB (input) INTEGER
The leading dimension of the array B. $LDB \geq \max(1,N)$.

X (input/output) REAL/COMPLEX array, dimension (LDX,NRHS)
On entry, the solution matrix X, as computed by SPPTRS/CPPTRS.
On exit, the improved solution matrix X.

LDX (input) INTEGER
The leading dimension of the array X. $LDX \geq \max(1,N)$.

FERR (output) REAL array, dimension (NRHS)
The estimated forward error bound for each solution vector X(j) (the jth column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) − XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR (output) REAL array, dimension (NRHS)
The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK *SPPRFS* (workspace) REAL array, dimension (3*N)
CPPRFS (workspace) COMPLEX array, dimension (2*N)

IWORK *SPPRFS only* (workspace) INTEGER array, dimension (N)

RWORK *CPPRFS only* (workspace) REAL array, dimension (N)

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SPPSV/CPPSV

```
SUBROUTINE SPPSV( UPLO, N, NRHS, AP, B, LDB, INFO )
CHARACTER          UPLO
INTEGER            INFO, LDB, N, NRHS
REAL               AP( * ), B( LDB, * )

SUBROUTINE CPPSV( UPLO, N, NRHS, AP, B, LDB, INFO )
CHARACTER          UPLO
INTEGER            INFO, LDB, N, NRHS
COMPLEX            AP( * ), B( LDB, * )
```

Purpose

SPPSV/CPPSV computes the solution to a real/complex system of linear equations A*X = B, where A is an n-by-n symmetric/Hermitian positive definite matrix stored in packed format and X and B are n-by-nrhs matrices.

The Cholesky decomposition is used to factor A as $A = U^H*U$, if UPLO = 'U', or $A = L*L^H$, if UPLO = 'L', where U is an upper triangular matrix and L is a lower triangular matrix. The factored form of A is then used to solve the system of equations A*X = B.

Arguments

UPLO (input) CHARACTER*1
= 'U': Upper triangle of A is stored;
= 'L': Lower triangle of A is stored.

N (input) INTEGER
The number of linear equations, i.e., the order of the matrix A. $N \geq 0$.

NRHS (input) INTEGER
The number of right hand sides, i.e., the number of columns of the matrix B. $NRHS \geq 0$.

AP (input/output) REAL/COMPLEX array, dimension (N*(N+1)/2)
On entry, the upper or lower triangle of the symmetric/Hermitian matrix A, packed columnwise in a linear array. The jth column of A is stored in the array AP as follows:
if UPLO = 'U', AP(i + (j−1)*j/2) = A(i,j) for $1 \le i \le j$;
if UPLO = 'L', AP(i + (j−1)*(2*n−j)/2) = A(i,j) for $j \le i \le n$.
On exit, if INFO = 0, the factor U or L from the Cholesky factorization $A = U^H*U$ or $A = L*L^H$, in the same storage format as A.

B (input/output) REAL/COMPLEX array, dimension (LDB,NRHS)
On entry, the n-by-nrhs right hand side matrix B.
On exit, if INFO = 0, the n-by-nrhs solution matrix X.

LDB (input) INTEGER
The leading dimension of the array B. LDB $\ge$ max(1,N).

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

SPPSVX/CPPSVX

```
      SUBROUTINE SPPSVX( FACT, UPLO, N, NRHS, AP, AFP, EQUED, S, B, LDB,
     $                   X, LDX, RCOND, FERR, BERR, WORK, IWORK, INFO )
      CHARACTER          EQUED, FACT, UPLO
      INTEGER            INFO, LDB, LDX, N, NRHS
      REAL               RCOND
      INTEGER            IWORK( * )
      REAL               AFP( * ), AP( * ), B( LDB, * ), BERR( * ),
     $                   FERR( * ), S( * ), WORK( * ), X( LDX, * )

      SUBROUTINE CPPSVX( FACT, UPLO, N, NRHS, AP, AFP, EQUED, S, B, LDB,
     $                   X, LDX, RCOND, FERR, BERR, WORK, RWORK, INFO )
      CHARACTER          EQUED, FACT, UPLO
      INTEGER            INFO, LDB, LDX, N, NRHS
      REAL               RCOND
      REAL               BERR( * ), FERR( * ), RWORK( * ), S( * )
      COMPLEX            AFP( * ), AP( * ), B( LDB, * ), WORK( * ),
     $                   X( LDX, * )
```

Purpose

SPPSVX/CPPSVX uses the Cholesky factorization $A = U^H*U$ or $A = L*L^H$ to compute the solution to a real/complex system of linear equations A*X = B, where A is an n-by-n symmetric/Hermitian positive definite matrix stored in packed format and X and B are n-by-nrhs matrices.

Error bounds on the solution and a condition estimate are also provided.

Description

The following steps are performed:

1. If FACT = 'E', real scaling factors are computed to equilibrate the system:

   $$\mathrm{diag}(S)*A*\mathrm{diag}(S)*(\mathrm{diag}(S))^{-1}*X = \mathrm{diag}(S)*B$$

   Whether or not the system will be equilibrated depends on the scaling of the matrix A, but if equilibration is used, A is overwritten by diag(S)*A*diag(S) and B by diag(S)*B.

2. If FACT = 'N' or 'E', the Cholesky decomposition is used to factor the matrix A (after equilibration if FACT = 'E') as
   $A = U^H*U$, if UPLO = 'U', or
   $A = L*L^H$, if UPLO = 'L',
   where U is an upper triangular matrix, and L is a lower triangular matrix.

3. If the leading i-by-i principal minor is not positive definite, then the routine returns with INFO = i. Otherwise, the factored form of A is used to estimate the condition number of the matrix A. If the reciprocal of the condition number is less than machine precision, INFO = N+1 is returned as a warning, but the routine still goes on to solve for X and compute error bounds as described below.

4. The system of equations is solved for X using the factored form of A.

5. Iterative refinement is applied to improve the computed solution matrix and calculate error bounds and backward error estimates for it.

6. If equilibration was used, the matrix X is premultiplied by diag(S) so that it solves the original system before equilibration.

Arguments

FACT (input) CHARACTER*1
Specifies whether or not the factored form of the matrix A is supplied on entry, and if not, whether the matrix A should be equilibrated before it is factored.
= 'F': On entry, AFP contains the factored form of A. If EQUED = 'Y', the matrix A has been equilibrated with scaling factors given by S. AP and AFP will not be modified.
= 'N': The matrix A will be copied to AFP and factored.
= 'E': The matrix A will be equilibrated if necessary, then copied to AFP and factored.

UPLO (input) CHARACTER*1
= 'U': Upper triangle of A is stored;
= 'L': Lower triangle of A is stored.

N (input) INTEGER
The number of linear equations, i.e., the order of the matrix A. N $\ge$ 0.

NRHS (input) INTEGER
The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS $\geq$ 0.

AP (input/output) REAL/COMPLEX array, dimension (N*(N+1)/2)
On entry, the upper or lower triangle of the symmetric/Hermitian matrix A, packed columnwise in a linear array, except if FACT = 'F' and EQUED = 'Y', then A must contain the equilibrated matrix diag(S)*A*diag(S). The jth column of A is stored in the array AP as follows:
if UPLO = 'U', AP(i + (j−1)*j/2) = A(i,j) for $1 \leq i \leq j$;
if UPLO = 'L', AP(i + (j−1)*(2*n−j)/2) = A(i,j) for $j \leq i \leq n$.
A is not modified if FACT = 'F' or 'N', or if FACT = 'E' and EQUED = 'N' on exit.
On exit, if FACT = 'E' and EQUED = 'Y', A is overwritten by diag(S)*A*diag(S).

AFP (input or output) REAL/COMPLEX array, dimension (N*(N+1)/2)
If FACT = 'F', then AFP is an input argument and on entry contains the triangular factor U or L from the Cholesky factorization $A = U^H*U$ or $A = L*L^H$, in the same storage format as A. If EQUED $\neq$ 'N', then AFP is the factored form of the equilibrated matrix A.
If FACT = 'N', then AFP is an output argument and on exit returns the triangular factor U or L from the Cholesky factorization $A = U^H*U$ or $A = L*L^H$ of the original matrix A.
If FACT = 'E', then AFP is an output argument and on exit returns the triangular factor U or L from the Cholesky factorization $A = U^H*U$ or $A = L*L^H$ of the equilibrated matrix A (see the description of AP for the form of the equilibrated matrix).

EQUED (input or output) CHARACTER*1
Specifies the form of equilibration that was done.
= 'N': No equilibration (always true if FACT = 'N').
= 'Y': Equilibration was done, i.e., A has been replaced by diag(S)*A*diag(S).
EQUED is an input argument if FACT = 'F'; otherwise, it is an output argument.

S (input or output) REAL array, dimension (N)
The scale factors for A; not accessed if EQUED = 'N'. S is an input argument if FACT = 'F'; otherwise, S is an output argument. If FACT = 'F' and EQUED = 'Y', each element of S must be positive.

B (input/output) REAL/COMPLEX array, dimension (LDB,NRHS)
On entry, the n-by-nrhs right hand side matrix B.
On exit, if EQUED = 'N', B is not modified; if EQUED = 'Y', B is overwritten by diag(S)*B.

LDB (input) INTEGER
The leading dimension of the array B. LDB $\geq$ max(1,N).

X (output) REAL/COMPLEX array, dimension (LDX,NRHS)
If INFO = 0 or INFO = N+1, the N-by-NRHS solution matrix X to the original system of equations. Note that if EQUED = 'Y', A and B are modified on exit, and the solution to the equilibrated system is $(\mathrm{diag}(S))^{-1}*X$.

LDX (input) INTEGER
The leading dimension of the array X. LDX $\geq$ max(1,N).

RCOND (output) REAL
The estimate of the reciprocal condition number of the matrix A after equilibration (if done). If RCOND is less than the machine precision (in particular, if RCOND = 0), the matrix is singular to working precision. This condition is indicated by a return code of INFO > 0.

FERR (output) REAL array, dimension (NRHS)
The estimated forward error bound for each solution vector X(j) (the jth column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) − XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR (output) REAL array, dimension (NRHS)
The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK *SPPSVX* (workspace) REAL array, dimension (3*N)
CPPSVX (workspace) COMPLEX array, dimension (2*N)

IWORK *SPPSVX only* (workspace) INTEGER array, dimension (N)

RWORK *CPPSVX only* (workspace) REAL array, dimension (N)

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, and i is
$\leq$ N: the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution and error bounds could not be computed. RCOND = 0 is returned.
= N+1: U is nonsingular, but RCOND is less than machine precision, meaning that the matrix is singular to working precision. Nevertheless, the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the value of RCOND would suggest.

SPPTRF/CPPTRF

```
SUBROUTINE SPPTRF( UPLO, N, AP, INFO )
CHARACTER          UPLO
INTEGER            INFO, N
REAL               AP( * )

SUBROUTINE CPPTRF( UPLO, N, AP, INFO )
CHARACTER          UPLO
INTEGER            INFO, N
COMPLEX            AP( * )
```

Purpose

SPPTRF/CPPTRF computes the Cholesky factorization of a real/complex symmetric/Hermitian positive definite matrix A stored in packed format.

The factorization has the form $A = U^H*U$, if UPLO = 'U', or $A = L*L^H$, if UPLO = 'L', where U is an upper triangular matrix and L is lower triangular.

Arguments

UPLO (input) CHARACTER*1
= 'U': Upper triangle of A is stored;
= 'L': Lower triangle of A is stored.

N (input) INTEGER
The order of the matrix A. $N \geq 0$.

AP (input/output) REAL/COMPLEX array, dimension (N*(N+1)/2)
On entry, the upper or lower triangle of the symmetric/Hermitian matrix A, packed columnwise in a linear array. The jth column of A is stored in the array AP as follows:
if UPLO = 'U', $AP(i + (j-1)*j/2) = A(i,j)$ for $1 \leq i \leq j$;
if UPLO = 'L', $AP(i + (j-1)*(2*n-j)/2) = A(i,j)$ for $j \leq i \leq n$.
On exit, if INFO = 0, the triangular factor U or L from the Cholesky factorization $A = U^H*U$ or $A = L*L^H$, in the same storage format as A.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, the leading minor of order i is not positive definite, and the factorization could not be completed.

SPPTRI/CPPTRI

```
SUBROUTINE SPPTRI( UPLO, N, AP, INFO )
CHARACTER          UPLO
INTEGER            INFO, N
REAL               AP( * )

SUBROUTINE CPPTRI( UPLO, N, AP, INFO )
CHARACTER          UPLO
INTEGER            INFO, N
COMPLEX            AP( * )
```

Purpose

SPPTRI/CPPTRI computes the inverse of a real/complex symmetric/Hermitian positive definite matrix A using the Cholesky factorization $A = U^H*U$ or $A = L*L^H$ computed by SPPTRF/CPPTRF.

Arguments

UPLO (input) CHARACTER*1
= 'U': Upper triangular factor is stored in AP;
= 'L': Lower triangular factor is stored in AP.

N (input) INTEGER
The order of the matrix A. $N \geq 0$.

AP (input/output) REAL/COMPLEX array, dimension (N*(N+1)/2)
On entry, the triangular factor U or L from the Cholesky factorization $A = U^H*U$ or $A = L*L^H$, packed columnwise as a linear array. The jth column of U or L is stored in the array AP as follows:
if UPLO = 'U', $AP(i + (j-1)*j/2) = U(i,j)$ for $1 \leq i \leq j$;
if UPLO = 'L', $AP(i + (j-1)*(2*n-j)/2) = L(i,j)$ for $j \leq i \leq n$.
On exit, the upper or lower triangle of the (symmetric)/(Hermitian) inverse of A, overwriting the input factor U or L.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, the (i,i) element of the factor U or L is zero, and the inverse could not be computed.

SPPTRS/CPPTRS

```
SUBROUTINE SPPTRS( UPLO, N, NRHS, AP, B, LDB, INFO )
CHARACTER          UPLO
INTEGER            INFO, LDB, N, NRHS
REAL               AP( * ), B( LDB, * )
```

```
SUBROUTINE CPPTRS( UPLO, N, NRHS, AP, B, LDB, INFO )
CHARACTER          UPLO
INTEGER            INFO, LDB, N, NRHS
COMPLEX            AP( * ), B( LDB, * )
```

Purpose

SPPTRS/CPPTRS solves a system of linear equations A*X = B with a symmetric/Hermitian positive definite matrix A in packed storage using the Cholesky factorization $A = U^H*U$ or $A = L*L^H$ computed by SPPTRF/CPPTRF.

Arguments

UPLO (input) CHARACTER*1
= 'U': Upper triangle of A is stored;
= 'L': Lower triangle of A is stored.

N (input) INTEGER
The order of the matrix A. $N \geq 0$.

NRHS (input) INTEGER
The number of right hand sides, i.e., the number of columns of the matrix B. $NRHS \geq 0$.

AP (input) REAL/COMPLEX array, dimension (N*(N+1)/2)
The triangular factor U or L from the Cholesky factorization $A = U^H*U$ or $A = L*L^H$, packed columnwise in a linear array. The jth column of U or L is stored in the array AP as follows:
if UPLO = 'U', AP(i + (j−1)*j/2) = U(i,j) for $1 \leq i \leq j$;
if UPLO = 'L', AP(i + (j−1)*(2*n−j)/2) = L(i,j) for $j \leq i \leq n$.

B (input/output) REAL/COMPLEX array, dimension (LDB,NRHS)
On entry, the right hand side matrix B.
On exit, the solution matrix X.

LDB (input) INTEGER
The leading dimension of the array B. $LDB \geq \max(1,N)$.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SPTCON/CPTCON

```
SUBROUTINE SPTCON( N, D, E, ANORM, RCOND, WORK, INFO )
INTEGER            INFO, N
REAL               ANORM, RCOND
REAL               D( * ), E( * ), WORK( * )
```

```
SUBROUTINE CPTCON( N, D, E, ANORM, RCOND, WORK, INFO )
INTEGER            INFO, N
REAL               ANORM, RCOND
REAL               D( * ), WORK( * )
COMPLEX            E( * )
```

Purpose

SPTCON/CPTCON computes the reciprocal of the condition number (in the 1-norm) of a real/complex symmetric/Hermitian positive definite tridiagonal matrix using the factorization $A = L*D*L^H$ or $A = U^H*D*U$ computed by SPTTRF/CPTTRF.

$\|A^{-1}\|$ is computed by a direct method, and the reciprocal of the condition number is computed as $RCOND = 1/(\|A\| * \|A^{-1}\|)$.

Arguments

N (input) INTEGER
The order of the matrix A. $N \geq 0$.

D (input) REAL array, dimension (N)
The n diagonal elements of the diagonal matrix D from the factorization of A, as computed by SPTTRF/CPTTRF.

E (input) REAL/COMPLEX array, dimension (N−1)
The (n−1) off-diagonal elements of the unit bidiagonal factor U or L from the factorization of A, as computed by SPTTRF/CPTTRF.

ANORM (input) REAL
The 1-norm of the original matrix A.

RCOND (output) REAL
The reciprocal of the condition number of the matrix A, computed as $RCOND = 1/(\|A\| * \|A^{-1}\|)$.

WORK (workspace) REAL array, dimension (N)

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SPTEQR/CPTEQR

```
SUBROUTINE SPTEQR( COMPZ, N, D, E, Z, LDZ, WORK, INFO )
CHARACTER          COMPZ
INTEGER            INFO, LDZ, N
REAL               D( * ), E( * ), WORK( * ), Z( LDZ, * )
```

```
SUBROUTINE CPTEQR( COMPZ, N, D, E, Z, LDZ, WORK, INFO )
CHARACTER          COMPZ
INTEGER            INFO, LDZ, N
REAL               D( * ), E( * ), WORK( * )
COMPLEX            Z( LDZ, * )
```

Purpose

SPTEQR/CPTEQR computes all eigenvalues and, optionally, eigenvectors of a symmetric positive definite tridiagonal matrix by first factoring the matrix using SPTTRF, and then calling SBDSQR/CBDSQR to compute the singular values of the bidiagonal factor.

This routine computes the eigenvalues of the positive definite tridiagonal matrix to high relative accuracy. This means that if the eigenvalues range over many orders of magnitude in size, then the small eigenvalues and corresponding eigenvectors will be computed more accurately than, for example, with the standard QR method.

The eigenvectors of a full or band symmetric/Hermitian positive definite matrix can be found if SSYTRD/CHETRD, SSPTRD/CHPTRD, or SSBTRD/CHBTRD has been used to reduce this matrix to tridiagonal form. (The reduction to tridiagonal form, however, may preclude the possibility of obtaining high relative accuracy in the small eigenvalues of the original matrix, these eigenvalues range over many orders of magnitude.)

Arguments

COMPZ (input) CHARACTER*1
- = 'N': Compute eigenvalues only.
- = 'V': Compute eigenvectors of original symmetric/Hermitian matrix also. Array Z contains the orthogonal/unitary matrix used to reduce the original matrix to tridiagonal form.
- = 'I': Compute eigenvectors of tridiagonal matrix also.

N (input) INTEGER
The order of the matrix. $N \geq 0$.

D (input/output) REAL array, dimension (N)
On entry, the n diagonal elements of the tridiagonal matrix.
On normal exit, D contains the eigenvalues, in descending order.

E (input/output) REAL array, dimension (N−1)
On entry, the (n−1) subdiagonal elements of the tridiagonal matrix.
On exit, E has been destroyed.

Z (input/output) REAL/COMPLEX array, dimension (LDZ, N)
On entry, if COMPZ = 'V', the orthogonal/unitary matrix used in the reduction to tridiagonal form.
On exit, if COMPZ = 'V', the orthonormal eigenvectors of the original symmetric/Hermitian matrix;
if COMPZ = 'I', the orthonormal eigenvectors of the tridiagonal matrix.
If INFO > 0 on exit, Z contains the eigenvectors associated with only the stored eigenvalues.
If COMPZ = 'N', then Z is not referenced.

LDZ (input) INTEGER
The leading dimension of the array Z. $LDZ \geq 1$, and if COMPZ = 'V' or 'I', $LDZ \geq \max(1,N)$.

WORK (workspace) REAL array, dimension (4*N)

INFO (output) INTEGER
- = 0: successful exit
- < 0: if INFO = −i, the ith argument had an illegal value.
- > 0: if INFO = i, and i is:
  - ≤ N: the Cholesky factorization of the matrix could not be performed because the ith principal minor was not positive definite.
  - > N: the SVD algorithm failed to converge; if INFO = N+i, i off-diagonal elements of the bidiagonal factor did not converge to zero.

SPTRFS/CPTRFS

```
SUBROUTINE SPTRFS( N, NRHS, D, E, DF, EF, B, LDB, X, LDX, FERR,
$                  BERR, WORK, INFO )
INTEGER            INFO, LDB, LDX, N, NRHS
REAL               B( LDB, * ), BERR( * ), D( * ), DF( * ),
$                  E( * ), EF( * ), FERR( * ), WORK( * ),
$                  X( LDX, * )

SUBROUTINE CPTRFS( UPLO, N, NRHS, D, E, DF, EF, B, LDB, X, LDX,
$                  FERR, BERR, WORK, RWORK, INFO )
CHARACTER          UPLO
INTEGER            INFO, LDB, LDX, N, NRHS
REAL               BERR( * ), D( * ), DF( * ), FERR( * ),
$                  RWORK( * )
COMPLEX            B( LDB, * ), E( * ), EF( * ), WORK( * ),
$                  X( LDX, * )
```

Purpose

SPTRFS/CPTRFS improves the computed solution to a system of linear equations when the coefficient matrix is symmetric/Hermitian positive definite and tridiagonal, and provides error bounds and backward error estimates for the solution.

Arguments

UPLO *CPTRFS only* (input) CHARACTER*1
Specifies whether the superdiagonal or the subdiagonal of the tridiagonal matrix A is stored and the form of the factorization:
- = 'U': E is the superdiagonal of A, and $A = U^H*D*U$;
- = 'L': E is the subdiagonal of A, and $A = L*D*L^H$.

(The two forms are equivalent if A is real.)

N (input) INTEGER
The order of the matrix A. $N \geq 0$.

NRHS (input) INTEGER
The number of right hand sides, i.e., the number of columns of the matrix B. $NRHS \geq 0$.

D (input) REAL array, dimension (N)
The n diagonal elements of the tridiagonal matrix A.

E (input) REAL/COMPLEX array, dimension (N−1)
The (n−1) off-diagonal elements of the tridiagonal matrix A (see UPLO).

DF (input) REAL array, dimension (N)
The n diagonal elements of the diagonal matrix D from the factorization computed by SPTTRF/CPTTRF.

EF (input) REAL/COMPLEX array, dimension (N−1)
The (n−1) off-diagonal elements of the unit bidiagonal factor U or L from the factorization computed by SPTTRF/CPTTRF (see UPLO).

B (input) REAL/COMPLEX array, dimension (LDB,NRHS)
The right hand side matrix B.

LDB (input) INTEGER
The leading dimension of the array B. $LDB \geq \max(1,N)$.

X (input/output) REAL/COMPLEX array, dimension (LDX,NRHS)
On entry, the solution matrix X, as computed by SPTTRS/CPTTRS.
On exit, the improved solution matrix X.

LDX (input) INTEGER
The leading dimension of the array X. $LDX \geq \max(1,N)$.

FERR (output) REAL array, dimension (NRHS)
The forward error bound for each solution vector X(j) (the jth column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) − XTRUE) divided by the magnitude of the largest element in X(j).

BERR (output) REAL array, dimension (NRHS)
The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK *SPTRFS* (workspace) REAL array, dimension (2*N)
CPTRFS (workspace) COMPLEX array, dimension (N)

RWORK *CPTRFS only* (workspace) REAL array, dimension (N)

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SPTSV/CPTSV

```
SUBROUTINE SPTSV( N, NRHS, D, E, B, LDB, INFO )
INTEGER            INFO, LDB, N, NRHS
REAL               B( LDB, * ), D( * ), E( * )

SUBROUTINE CPTSV( N, NRHS, D, E, B, LDB, INFO )
INTEGER            INFO, LDB, N, NRHS
REAL               D( * )
COMPLEX            B( LDB, * ), E( * )
```

Purpose

SPTSV/CPTSV computes the solution to a real/complex system of linear equations A*X = B, where A is an n-by-n symmetric/Hermitian positive definite tridiagonal matrix, and X and B are n-by-nrhs matrices.

A is factored as $A = L*D*L^H$, and the factored form of A is then used to solve the system of equations.

Arguments

N (input) INTEGER
The order of the matrix A. $N \geq 0$.

NRHS (input) INTEGER
The number of right hand sides, i.e., the number of columns of the matrix B. $NRHS \geq 0$.

D (input/output) REAL array, dimension (N)
On entry, the n diagonal elements of the tridiagonal matrix A. On exit, the n diagonal elements of the diagonal matrix D from the factorization $A = L*D*L^H$.

E (input/output) REAL/COMPLEX array, dimension (N−1)
On entry, the (n−1) subdiagonal elements of the tridiagonal matrix A.
On exit, the (n−1) subdiagonal elements of the unit bidiagonal factor L from the $L*D*L^H$ factorization of A.

B (input/output) REAL/COMPLEX array, dimension (LDB,NRHS)
On entry, the n-by-nrhs right hand side matrix B.
On exit, if INFO = 0, the n-by-nrhs solution matrix X.

LDB (input) INTEGER
The leading dimension of the array B. $LDB \geq \max(1,N)$.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, the leading minor of order i is not positive definite, and the solution has not been computed. The factorization has not been completed unless i = N.

SPTSVX/CPTSVX

```
      SUBROUTINE SPTSVX( FACT, N, NRHS, D, E, DF, EF, B, LDB, X, LDX,
     $                   RCOND, FERR, BERR, WORK, INFO )
      CHARACTER          FACT
      INTEGER            INFO, LDB, LDX, N, NRHS
      REAL               RCOND
      REAL               B( LDB, * ), BERR( * ), D( * ), DF( * ),
     $                   E( * ), EF( * ), FERR( * ), WORK( * ),
     $                   X( LDX, * )

      SUBROUTINE CPTSVX( FACT, N, NRHS, D, E, DF, EF, B, LDB, X, LDX,
     $                   RCOND, FERR, BERR, WORK, RWORK, INFO )
      CHARACTER          FACT
      INTEGER            INFO, LDB, LDX, N, NRHS
      REAL               RCOND
      REAL               BERR( * ), D( * ), DF( * ), FERR( * ),
     $                   RWORK( * )
      COMPLEX            B( LDB, * ), E( * ), EF( * ), WORK( * ),
     $                   X( LDX, * )
```

Purpose

SPTSVX/CPTSVX uses the factorization $A = L*D*L^H$ to compute the solution to a real/complex system of linear equations $A*X = B$, where A is an n-by-n symmetric/Hermitian positive definite tridiagonal matrix and X and B are n-by-nrhs matrices.

Error bounds on the solution and a condition estimate are also provided.

Description

The following steps are performed:

1. If FACT = 'N', the matrix A is factored as $A = L*D*L^H$, where L is a unit lower bidiagonal matrix and D is diagonal. The factorization can also be regarded as having the form $A = U^H*D*U$.
2. If the leading i-by-i principal minor is not positive definite, then the routine returns with INFO = i. Otherwise, the factored form of A is used to estimate the condition number of the matrix A. If the reciprocal of the condition number is less than machine precision, INFO = N+1 is returned as a warning, but the routine still goes on to solve for X and compute error bounds as described below.
3. The system of equations is solved for X using the factored form of A.
4. Iterative refinement is applied to improve the computed solution matrix and calculate error bounds and backward error estimates for it.

Arguments

FACT (input) CHARACTER*1
Specifies whether or not the factored form of the matrix A is supplied on entry.
= 'F': On entry, DF and EF contain the factored form of A. D, E, DF, and EF will not be modified.
= 'N': The matrix A will be copied to DF and EF and factored.

N (input) INTEGER
The order of the matrix A. $N \geq 0$.

NRHS (input) INTEGER
The number of right hand sides, i.e., the number of columns of the matrices B and X. $NRHS \geq 0$.

D (input) REAL array, dimension (N)
The n diagonal elements of the tridiagonal matrix A.

E (input) REAL/COMPLEX array, dimension (N−1)
The (n−1) subdiagonal elements of the tridiagonal matrix A.

DF (input or output) REAL array, dimension (N)
If FACT = 'F', then DF is an input argument and on entry contains the n diagonal elements of the diagonal matrix D from the $L*D*L^H$ factorization of A.
If FACT = 'N', then DF is an output argument and on exit contains the n diagonal elements of the diagonal matrix D from the $L*D*L^H$ factorization of A.

EF (input or output) REAL/COMPLEX array, dimension (N−1)
If FACT = 'F', then EF is an input argument and on entry contains the (n−1) subdiagonal elements of the unit bidiagonal factor L from the $L*D*L^H$ factorization of A.
If FACT = 'N', then EF is an output argument and on exit contains the (n−1) subdiagonal elements of the unit bidiagonal factor L from the $L*D*L^H$ factorization of A.

B (input) REAL/COMPLEX array, dimension (LDB,NRHS)
The n-by-nrhs right hand side matrix B.

LDB (input) INTEGER
The leading dimension of the array B. $LDB \geq \max(1,N)$.

X (output) REAL/COMPLEX array, dimension (LDX,NRHS)
If INFO = 0 of INFO = N+1, the N-by-NRHS solution matrix X.

LDX (input) INTEGER
The leading dimension of the array X. $LDX \geq \max(1,N)$.

RCOND (output) REAL
The reciprocal condition number of the matrix A. If RCOND is less than the machine precision (in particular, if RCOND = 0), the matrix is singular to working precision.
This condition is indicated by a return code of INFO > 0.

FERR (output) REAL array, dimension (NRHS)
The forward error bound for each solution vector X(j) (the jth column of the solution matrix X). If XTRUE is the true solution corresponding

to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) − XTRUE) divided by the magnitude of the largest element in X(j).

BERR (output) REAL array, dimension (NRHS)
The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK *SPTSVX* (workspace) REAL array, dimension (2*N)
CPTSVX (workspace) COMPLEX array, dimension (N)

RWORK *CPTSVX only* (workspace) REAL array, dimension (N)

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, and i is
≤ N: the leading minor of order i of A is not positive definite, so the factorization could not be completed unless i = N, and the solution and error bounds could not be computed. RCOND = 0 is returned.
= N+1: U is nonsingular, but RCOND is less than machine precision, meaning that the matrix is singular to working precision. Nevertheless, the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the value of RCOND would suggest.

SPTTRF/CPTTRF

```
SUBROUTINE SPTTRF( N, D, E, INFO )
INTEGER            INFO, N
REAL               D( * ), E( * )

SUBROUTINE CPTTRF( N, D, E, INFO )
INTEGER            INFO, N
REAL               D( * )
COMPLEX            E( * )
```

Purpose

SPTTRF/CPTTRF computes the factorization of a real/complex symmetric/Hermitian positive definite tridiagonal matrix A.

If the subdiagonal elements of A are supplied in the array E, the factorization has the form $A = L*D*L^H$, where D is diagonal and L is unit lower bidiagonal; if the superdiagonal elements of A are supplied, it has the form $A = U^H*D*U$, where U is unit upper bidiagonal. (The two forms are equivalent if A is real.)

Arguments

N (input) INTEGER
The order of the matrix A. N ≥ 0.

D (input/output) REAL array, dimension (N)
On entry, the n diagonal elements of the tridiagonal matrix A. On exit, the n diagonal elements of the diagonal matrix D from the $L*D*L^H$ factorization of A.

E (input/output) REAL/COMPLEX array, dimension (N−1)
On entry, the (n−1) off-diagonal elements of the tridiagonal matrix A. On exit, the (n−1) off-diagonal elements of the unit bidiagonal factor L or U from the factorization of A.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, the leading minor of order i is not positive definite; if i < N, the factorization could not be completed, while if i = N, the factorization was completed, but D(N) = 0.

SPTTRS/CPTTRS

```
SUBROUTINE SPTTRS( N, NRHS, D, E, B, LDB, INFO )
INTEGER            INFO, LDB, N, NRHS
REAL               B( LDB, * ), D( * ), E( * )

SUBROUTINE CPTTRS( UPLO, N, NRHS, D, E, B, LDB, INFO )
CHARACTER          UPLO
INTEGER            INFO, LDB, N, NRHS
REAL               D( * )
COMPLEX            B( LDB, * ), E( * )
```

Purpose

SPTTRS/CPTTRS solves a system of linear equations A*X = B with a symmetric/Hermitian positive definite tridiagonal matrix A using the factorization $A = L*D*L^H$ or $A = U^H*D*U$ computed by SPTTRF/CPTTRF.

Arguments

UPLO *CPTTRS only* (input) CHARACTER*1
Specifies whether the superdiagonal or the subdiagonal of the tridiagonal matrix A is stored and the form of the factorization:
= 'U': E is the superdiagonal of A, and $A = U^H*D*U$;
= 'L': E is the subdiagonal of A, and $A = L*D*L^H$.
(The two forms are equivalent if A is real.)

N (input) INTEGER
The order of the tridiagonal matrix A. N ≥ 0.

NRHS (input) INTEGER
The number of right hand sides, i.e., the number of columns of the matrix B. NRHS $\geq$ 0.

D (input) REAL array, dimension (N)
The n diagonal elements of the diagonal matrix D from the factorization computed by SPTTRF/CPTTRF.

E (input) REAL/COMPLEX array, dimension (N−1)
The (n−1) off-diagonal elements of the unit bidiagonal factor U or L from the factorization computed by SPTTRF/CPTTRF (see UPLO).

B (input/output) REAL/COMPLEX array, dimension (LDB,NRHS)
On entry, the right hand side matrix B.
On exit, the solution matrix X.

LDB (input) INTEGER
The leading dimension of the array B. LDB $\geq$ max(1,N).

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SSBEV/CHBEV

```
      SUBROUTINE SSBEV( JOBZ, UPLO, N, KD, AB, LDAB, W, Z, LDZ, WORK,
     $                  INFO )
      CHARACTER          JOBZ, UPLO
      INTEGER            INFO, KD, LDAB, LDZ, N
      REAL               AB( LDAB, * ), W( * ), WORK( * ), Z( LDZ, * )

      SUBROUTINE CHBEV( JOBZ, UPLO, N, KD, AB, LDAB, W, Z, LDZ, WORK,
     $                  RWORK, INFO )
      CHARACTER          JOBZ, UPLO
      INTEGER            INFO, KD, LDAB, LDZ, N
      REAL               RWORK( * ), W( * )
      COMPLEX            AB( LDAB, * ), WORK( * ), Z( LDZ, * )
```

Purpose

SSBEV/CHBEV computes all the eigenvalues and, optionally, eigenvectors of a real/complex symmetric/Hermitian band matrix A.

Arguments

JOBZ (input) CHARACTER*1
= 'N': Compute eigenvalues only;
= 'V': Compute eigenvalues and eigenvectors.

UPLO (input) CHARACTER*1
= 'U': Upper triangle of A is stored;
= 'L': Lower triangle of A is stored.

N (input) INTEGER
The order of the matrix A. N $\geq$ 0.

KD (input) INTEGER
The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD $\geq$ 0.

AB (input/output) REAL/COMPLEX array, dimension (LDAB, N)
On entry, the upper or lower triangle of the symmetric/Hermitian band matrix A, stored in the first kd+1 rows of the array. The jth column of A is stored in the jth column of the array AB as follows:
if UPLO = 'U', AB(kd+1+i−j,j) = A(i,j) for max(1,j−kd)$\leq$i$\leq$j;
if UPLO = 'L', AB(1+i−j,j) = A(i,j) for j$\leq$i$\leq$min(n,j+kd).
On exit, AB is overwritten by values generated during the reduction to tridiagonal form. If UPLO = 'U', the first superdiagonal and the diagonal of the tridiagonal matrix T are returned in rows kd and kd+1 of AB, and if UPLO = 'L', the diagonal and first subdiagonal of T are returned in the first two rows of AB.

LDAB (input) INTEGER
The leading dimension of the array AB. LDAB $\geq$ KD + 1.

W (output) REAL array, dimension (N)
If INFO = 0, the eigenvalues in ascending order.

Z (output) REAL/COMPLEX array, dimension (LDZ, N)
If JOBZ = 'V', then if INFO = 0, Z contains the orthonormal eigenvectors of the matrix A, with the ith column of Z holding the eigenvector associated with W(i).
If JOBZ = 'N', then Z is not referenced.

LDZ (input) INTEGER
The leading dimension of the array Z. LDZ $\geq$ 1, and if JOBZ = 'V', LDZ $\geq$ max(1,N).

WORK *SSBEV* (workspace) REAL array, dimension (max(1,3*N−2))
CHBEV (workspace) COMPLEX array, dimension (N)

RWORK *CHBEV only* (workspace) REAL array, dimension (max(1,3*N−2))

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: the algorithm failed to converge; if INFO = i, i off-diagonal elements of an intermediate tridiagonal form did not converge to zero.

SSBEVD/CHBEVD

```
      SUBROUTINE SSBEVD( JOBZ, UPLO, N, KD, AB, LDAB, W, Z, LDZ, WORK,
     $                   LWORK, IWORK, LIWORK, INFO )
      CHARACTER          JOBZ, UPLO
      INTEGER            INFO, KD, LDAB, LDZ, LIWORK, LWORK, N
      INTEGER            IWORK( * )
      REAL               AB( LDAB, * ), W( * ), WORK( * ), Z( LDZ, * )

      SUBROUTINE CHBEVD( JOBZ, UPLO, N, KD, AB, LDAB, W, Z, LDZ, WORK,
     $                   LWORK, RWORK, LRWORK, IWORK, LIWORK, INFO )
      CHARACTER          JOBZ, UPLO
      INTEGER            INFO, KD, LDAB, LDZ, LIWORK, LRWORK, LWORK, N
      INTEGER            IWORK( * )
      REAL               RWORK( * ), W( * )
      COMPLEX            AB( LDAB, * ), WORK( * ), Z( LDZ, * )
```

Purpose

SSBEVD/CHBEVD computes all the eigenvalues and, optionally, eigenvectors of a real/complex symmetric/Hermitian band matrix A. If eigenvectors are desired, it uses a divide and conquer algorithm.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Arguments

JOBZ (input) CHARACTER*1
= 'N': Compute eigenvalues only;
= 'V': Compute eigenvalues and eigenvectors.

UPLO (input) CHARACTER*1
= 'U': Upper triangle of A is stored;
= 'L': Lower triangle of A is stored.

N (input) INTEGER
The order of the matrix A. $N \geq 0$.

KD (input) INTEGER
The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. $KD \geq 0$.

AB (input/output) REAL/COMPLEX array, dimension (LDAB, N)
On entry, the upper or lower triangle of the symmetric/Hermitian band matrix A, stored in the first kd+1 rows of the array. The jth column of A is stored in the jth column of the array AB as follows:
if UPLO = 'U', $AB(kd+1+i-j,j) = A(i,j)$ for $\max(1,j-kd) \leq i \leq j$;
if UPLO = 'L', $AB(1+i-j,j) = A(i,j)$ for $j \leq i \leq \min(n,j+kd)$.
On exit, AB is overwritten by values generated during the reduction to tridiagonal form. If UPLO = 'U', the first superdiagonal and the diagonal of the tridiagonal matrix T are returned in rows kd and kd+1 of AB, and if UPLO = 'L', the diagonal and first subdiagonal of T are returned in the first two rows of AB.

LDAB (input) INTEGER
The leading dimension of the array AB. $LDAB \geq KD + 1$.

W (output) REAL array, dimension (N)
If INFO = 0, the eigenvalues in ascending order.

Z (output) REAL/COMPLEX array, dimension (LDZ, N)
If JOBZ = 'V', then if INFO = 0, Z contains the orthonormal eigenvectors of the matrix A, with the ith column of Z holding the eigenvector associated with W(i).
If JOBZ = 'N', then Z is not referenced.

LDZ (input) INTEGER
The leading dimension of the array Z. $LDZ \geq 1$, and if JOBZ = 'V', $LDZ \geq \max(1,N)$.

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK. If $N \leq 1$, $LWORK \geq 1$.
SSBEVD
If JOBZ = 'N' and $N > 2$, $LWORK \geq 2*N$.
If JOBZ = 'V' and $N > 2$, $LWORK \geq (1+5*N+2*N^2)$.
CHBEVD
If JOBZ = 'N' and $N > 1$, $LWORK \geq N$.
If JOBZ = 'V' and $N > 1$, $LWORK \geq 2*N^2$.

If LWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK *CHBEVD only* (workspace/output) REAL array, dimension (LRWORK)
On exit, if INFO = 0, RWORK(1) returns the optimal LRWORK.

LRWORK *CHBEVD only* (input) INTEGER
The dimension of array RWORK.
If $N \leq 1$, $LRWORK \geq 1$.
If JOBZ = 'N' and $N > 1$, $LRWORK \geq N$.
If JOBZ = 'V' and $N > 1$, $LRWORK \geq (1+5*N+2*N^2)$.

If LRWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the RWORK array, returns this value as the first entry of the RWORK array, and no error message related to LRWORK is issued by XERBLA.

IWORK (workspace/output) INTEGER array, dimension (LIWORK)
On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK (input) INTEGER
The dimension of the array LIWORK.
SSBEVD
If JOBZ = 'N' or N $\leq$ 1, LIWORK $\geq$ 1.
If JOBZ = 'V' and N > 2, LIWORK $\geq$ 3+5*N.
CHBEVD
If JOBZ = 'N' or N $\leq$ 1, LIWORK $\geq$ 1.
If JOBZ = 'V' and N > 1, LIWORK $\geq$ 3+5*N.

If LIWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the IWORK array, returns this value as the first entry of the IWORK array, and no error message related to LIWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero.

SSBEVX/CHBEVX

```
      SUBROUTINE SSBEVX( JOBZ, RANGE, UPLO, N, KD, AB, LDAB, Q, LDQ, VL,
     $                   VU, IL, IU, ABSTOL, M, W, Z, LDZ, WORK, IWORK,
     $                   IFAIL, INFO )
      CHARACTER          JOBZ, RANGE, UPLO
      INTEGER            IL, INFO, IU, KD, LDAB, LDQ, LDZ, M, N
      REAL               ABSTOL, VL, VU
      INTEGER            IFAIL( * ), IWORK( * )
      REAL               AB( LDAB, * ), Q( LDQ, * ), W( * ), WORK( * ),
     $                   Z( LDZ, * )

      SUBROUTINE CHBEVX( JOBZ, RANGE, UPLO, N, KD, AB, LDAB, Q, LDQ, VL,
     $                   VU, IL, IU, ABSTOL, M, W, Z, LDZ, WORK, RWORK,
     $                   IWORK, IFAIL, INFO )
      CHARACTER          JOBZ, RANGE, UPLO
      INTEGER            IL, INFO, IU, KD, LDAB, LDQ, LDZ, M, N
      REAL               ABSTOL, VL, VU
      INTEGER            IFAIL( * ), IWORK( * )
      REAL               RWORK( * ), W( * )
      COMPLEX            AB( LDAB, * ), Q( LDQ, * ), WORK( * ),
     $                   Z( LDZ, * )
```

Purpose

SSBEVX/CHBEVX computes selected eigenvalues and, optionally, eigenvectors of a real/complex symmetric/Hermitian band matrix A. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

Arguments

JOBZ (input) CHARACTER*1
= 'N': Compute eigenvalues only;
= 'V': Compute eigenvalues and eigenvectors.

RANGE (input) CHARACTER*1
= 'A': all eigenvalues will be found;
= 'V': all eigenvalues in the half-open interval (VL,VU] will be found;
= 'I': the ILth through IU^{th} eigenvalues will be found.

UPLO (input) CHARACTER*1
= 'U': Upper triangle of A is stored;
= 'L': Lower triangle of A is stored.

N (input) INTEGER
The order of the matrix A. N $\geq$ 0.

KD (input) INTEGER
The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD $\geq$ 0.

AB (input/output) REAL/COMPLEX array, dimension (LDAB, N)
On entry, the upper or lower triangle of the symmetric/Hermitian band matrix A, stored in the first kd+1 rows of the array. The jth column of A is stored in the jth column of the array AB as follows:
if UPLO = 'U', AB(kd+1+i−j,j) = A(i,j) for max(1,j−kd)$\leq$i$\leq$j;
if UPLO = 'L', AB(1+i−j,j) = A(i,j) for j$\leq$i$\leq$min(n,j+kd).
On exit, AB is overwritten by values generated during the reduction to tridiagonal form. If UPLO = 'U', the first superdiagonal and the diagonal of the tridiagonal matrix T are returned in rows kd and kd+1 of AB, and if UPLO = 'L', the diagonal and first subdiagonal of T are returned in the first two rows of AB.

LDAB (input) INTEGER
The leading dimension of the array AB. LDAB $\geq$ KD + 1.

Q (output) REAL/COMPLEX array, dimension (LDQ, N)
If JOBZ = 'V', the n-by-n orthogonal/unitary matrix used in the reduction to tridiagonal form.
If JOBZ = 'N', the array Q is not referenced.

LDQ (input) INTEGER
The leading dimension of the array Q. If JOBZ = 'V', then LDQ $\geq$ max(1,N).

VL, VU (input) REAL
If RANGE='V', the lower and upper bounds of the interval to be

searched for eigenvalues. VL < VU.
Not referenced if RANGE = 'A' or 'I'.

IL, IU (input) INTEGER
If RANGE='I', the indices (in ascending order) of the smallest and largest eigenvalues to be returned.
$1 \leq IL \leq IU \leq N$, if $N > 0$; IL = 1 and IU = 0 if N = 0.
Not referenced if RANGE = 'A' or 'V'.

ABSTOL (input) REAL
The absolute error tolerance for the eigenvalues.
An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to ABSTOL + EPS*max(|a|,|b|), where EPS is the machine precision. If ABSTOL ≤ 0, then EPS*$||T||_1$ will be used in its place, where T is the tridiagonal matrix obtained by reducing AB to tridiagonal form.
Eigenvalues will be computed most accurately when ABSTOL is set to twice the underflow threshold 2*SLAMCH('S'), not zero. If this routine returns with INFO>0, indicating that some eigenvectors did not converge, try setting ABSTOL to 2*SLAMCH('S').

M (output) INTEGER
The total number of eigenvalues found. $0 \leq M \leq N$.
If RANGE = 'A', M = N, and if RANGE = 'I', M = IU−IL+1.

W (output) REAL array, dimension (N)
The first M elements contain the selected eigenvalues in ascending order.

Z (output) REAL/COMPLEX array, dimension (LDZ, max(1,M))
If JOBZ = 'V', then if INFO = 0, the first M columns of Z contain the orthonormal eigenvectors of the matrix A corresponding to the selected eigenvalues, with the ith column of Z holding the eigenvector asociated with W(i). If an eigenvector fails to converge, then that column of Z contains the latest approximation to the eigenvector, and the index of the eigenvector is returned in IFAIL.
If JOBZ = 'N', then Z is not referenced.
Note: the user must ensure that at least max(1,M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and an upper bound must be used.

LDZ (input) INTEGER
The leading dimension of the array Z. LDZ $\geq$ 1, and if JOBZ = 'V', LDZ $\geq$ max(1,N).

WORK *SSBEVX* (workspace) REAL array, dimension (7*N)
CHBEVX (workspace) COMPLEX array, dimension (N)

RWORK *CHBEVX only* (workspace) REAL array, dimension (7*N)

IWORK (workspace) INTEGER array, dimension (5*N)

IFAIL (output) INTEGER array, dimension (N)
If JOBZ = 'V', then if INFO = 0, the first M elements of IFAIL are zero; if INFO > 0, then IFAIL contains the indices of the eigenvectors that failed to converge.
If JOBZ = 'N', then IFAIL is not referenced.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, then i eigenvectors failed to converge. Their indices are stored in array IFAIL.

SSBGST/CHBGST

```
      SUBROUTINE SSBGST( VECT, UPLO, N, KA, KB, AB, LDAB, BB, LDBB, X,
     $                   LDX, WORK, INFO )
      CHARACTER          UPLO, VECT
      INTEGER            INFO, KA, KB, LDAB, LDBB, LDX, N
      REAL               AB( LDAB, * ), BB( LDBB, * ), WORK( * ),
     $                   X( LDX, * )

      SUBROUTINE CHBGST( VECT, UPLO, N, KA, KB, AB, LDAB, BB, LDBB, X,
     $                   LDX, WORK, RWORK, INFO )
      CHARACTER          UPLO, VECT
      INTEGER            INFO, KA, KB, LDAB, LDBB, LDX, N
      REAL               RWORK( * )
      COMPLEX            AB( LDAB, * ), BB( LDBB, * ), WORK( * ),
     $                   X( LDX, * )
```

Purpose

SSBGST/CHBGST reduces a real/complex symmetric-definite/Hermitian-definite banded generalized eigenproblem $A*x = \lambda*B*x$ to standard form $C*y = \lambda*y$, such that C has the same bandwidth as A.

B must have been previously factorized as S^H*S by SPBSTF/CPBSTF, using a split Cholesky factorization. A is overwritten by $C = X^H*A*X$, where $X = S^{-1}*Q$ and Q is an/a orthogonal/unitary matrix chosen to preserve the bandwidth of A.

Arguments

VECT (input) CHARACTER*1
= 'N': do not form the transformation matrix X;
= 'V': form X.

UPLO (input) CHARACTER*1
= 'U': Upper triangle of A is stored;
= 'L': Lower triangle of A is stored.

N (input) INTEGER
The order of the matrices A and B. $N \geq 0$.

KA (input) INTEGER
The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KA $\geq$ 0.

KB (input) INTEGER
The number of superdiagonals of the matrix B if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KA $\geq$ KB $\geq$ 0.

AB (input/output) REAL/COMPLEX array, dimension (LDAB,N)
On entry, the upper or lower triangle of the symmetric/Hermitian band matrix A, stored in the first ka+1 rows of the array. The jth column of A is stored in the jth column of the array AB as follows:
if UPLO = 'U', AB(ka+1+i−j,j) = A(i,j) for max(1,j−ka)$\leq$i$\leq$j;
if UPLO = 'L', AB(1+i−j,j) = A(i,j) for j$\leq$i$\leq$min(n,j+ka).
On exit, the transformed matrix X^H*A*X, stored in the same format as A.

LDAB (input) INTEGER
The leading dimension of the array AB. LDAB $\geq$ KA+1.

BB (input) REAL/COMPLEX array, dimension (LDBB,N)
The banded factor S from the split Cholesky factorization of B, as returned by SPBSTF/CPBSTF, stored in the first kb+1 rows of the array.

LDBB (input) INTEGER
The leading dimension of the array BB. LDBB $\geq$ KB+1.

X (output) REAL/COMPLEX array, dimension (LDX,N)
If VECT = 'V', the n-by-n matrix X.
If VECT = 'N', the array X is not referenced.

LDX (input) INTEGER
The leading dimension of the array X.
LDX $\geq$ max(1,N) if VECT = 'V'; LDX $\geq$ 1 otherwise.

WORK *SSBGST* (workspace) REAL array, dimension (2*N)
CHBGST (workspace) COMPLEX array, dimension (N)

RWORK *CHBGST only* (workspace) REAL array, dimension (N)

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SSBGV/CHBGV

```
      SUBROUTINE SSBGV( JOBZ, UPLO, N, KA, KB, AB, LDAB, BB, LDBB, W, Z,
     $                  LDZ, WORK, INFO )
      CHARACTER          JOBZ, UPLO
      INTEGER            INFO, KA, KB, LDAB, LDBB, LDZ, N
      REAL               AB( LDAB, * ), BB( LDBB, * ), W( * ),
     $                   WORK( * ), Z( LDZ, * )
```

```
      SUBROUTINE CHBGV( JOBZ, UPLO, N, KA, KB, AB, LDAB, BB, LDBB, W, Z,
     $                  LDZ, WORK, RWORK, INFO )
      CHARACTER          JOBZ, UPLO
      INTEGER            INFO, KA, KB, LDAB, LDBB, LDZ, N
      REAL               RWORK( * ), W( * )
      COMPLEX            AB( LDAB, * ), BB( LDBB, * ), WORK( * ),
     $                   Z( LDZ, * )
```

Purpose

SSBGV/CHBGV computes all the eigenvalues, and optionally, the eigenvectors of a real/complex generalized symmetric-definite/Hermitian-definite banded eigenproblem, of the form A*x = λ*B*x. Here A and B are assumed to be symmetric/Hermitian and banded, and B is also positive definite.

Arguments

JOBZ (input) CHARACTER*1
= 'N': Compute eigenvalues only;
= 'V': Compute eigenvalues and eigenvectors.

UPLO (input) CHARACTER*1
= 'U': Upper triangles of A and B are stored;
= 'L': Lower triangles of A and B are stored.

N (input) INTEGER
The order of the matrices A and B. N $\geq$ 0.

KA (input) INTEGER
The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KA $\geq$ 0.

KB (input) INTEGER
The number of superdiagonals of the matrix B if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KB $\geq$ 0.

AB (input/output) REAL/COMPLEX array, dimension (LDAB, N)
On entry, the upper or lower triangle of the symmetric/Hermitian band matrix A, stored in the first ka+1 rows of the array. The jth column of A is stored in the jth column of the array AB as follows:
if UPLO = 'U', AB(ka+1+i−j,j) = A(i,j) for max(1,j−ka)$\leq$i$\leq$j;
if UPLO = 'L', AB(1+i−j,j) = A(i,j) for j$\leq$i$\leq$min(n,j+ka).
On exit, the contents of AB are destroyed.

LDAB (input) INTEGER
The leading dimension of the array AB. LDAB $\geq$ KA+1.

BB (input/output) REAL/COMPLEX array, dimension (LDBB, N)
On entry, the upper or lower triangle of the symmetric/Hermitian band matrix B, stored in the first kb+1 rows of the array. The jth column of B is stored in the jth column of the array BB as follows:
if UPLO = 'U', BB(kb+1+i−j,j) = B(i,j) for max(1,j−kb)$\leq$i$\leq$j;
if UPLO = 'L', BB(1+i−j,j) = B(i,j) for j$\leq$i$\leq$min(n,j+kb).
On exit, the factor S from the split Cholesky factorization B = S^H*S, as returned by SPBSTF/CPBSTF.

LDBB (input) INTEGER
The leading dimension of the array BB. LDBB $\geq$ KB+1.

W (output) REAL array, dimension (N)
If INFO = 0, the eigenvalues in ascending order.

Z (output) REAL/COMPLEX array, dimension (LDZ, N)
If JOBZ = 'V', then if INFO = 0, Z contains the matrix Z of eigenvectors, with the ith column of Z holding the eigenvector associated with W(i). The eigenvectors are normalized so that $Z^H*B*Z = I$.
If JOBZ = 'N', then Z is not referenced.

LDZ (input) INTEGER
The leading dimension of the array Z. LDZ $\geq$ 1, and if JOBZ = 'V', LDZ $\geq$ N.

WORK *SSBGV* (workspace) REAL array, dimension (3*N)
CHBGV (workspace) COMPLEX array, dimension (N)

RWORK *CHBGV only* (workspace) REAL array, dimension (3*N)

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, and i is:
$\leq$ N: the algorithm failed to converge: i off-diagonal elements of an intermediate tridiagonal form did not converge to zero;
> N: if INFO = N + i, for 1 $\leq$ i $\leq$ N, then SPBSTF/CPBSTF returned INFO = i: B is not positive definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed.

SSBGVD/CHBGVD

```
      SUBROUTINE SSBGVD( JOBZ, UPLO, N, KA, KB, AB, LDAB, BB, LDBB, W,
     $                   Z, LDZ, WORK, LWORK, IWORK, LIWORK, INFO )
      CHARACTER          JOBZ, UPLO
      INTEGER            INFO, KA, KB, LDAB, LDBB, LDZ, LIWORK, LWORK, N
      INTEGER            IWORK( * )
      REAL               AB( LDAB, * ), BB( LDBB, * ), W( * ),
     $                   WORK( * ), Z( LDZ, * )
```

```
      SUBROUTINE CHBGVD( JOBZ, UPLO, N, KA, KB, AB, LDAB, BB, LDBB, W,
     $                   Z, LDZ, WORK, LWORK, RWORK, LRWORK, IWORK,
     $                   LIWORK, INFO )
      CHARACTER          JOBZ, UPLO
      INTEGER            INFO, KA, KB, LDAB, LDBB, LDZ, LIWORK, LRWORK,
     $                   LWORK, N
      INTEGER            IWORK( * )
      REAL               RWORK( * ), W( * )
      COMPLEX            AB( LDAB, * ), BB( LDBB, * ), WORK( * ),
     $                   Z( LDZ, * )
```

Purpose

SSBGVD/CHBGVD computes all the eigenvalues, and optionally, the eigenvectors of a real/complex generalized symmetric-definite/Hermitian-definite banded eigenproblem, of the form A*x = λ*B*x. Here A and B are assumed to be symmetric/Hermitian and banded, and B is also positive definite. If eigenvectors are desired, it uses a divide and conquer algorithm.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Arguments

JOBZ (input) CHARACTER*1
= 'N': Compute eigenvalues only;
= 'V': Compute eigenvalues and eigenvectors.

UPLO (input) CHARACTER*1
= 'U': Upper triangles of A and B are stored;
= 'L': Lower triangles of A and B are stored.

N (input) INTEGER
The order of the matrices A and B. N $\geq$ 0.

KA (input) INTEGER
The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KA $\geq$ 0.

KB (input) INTEGER
The number of superdiagonals of the matrix B if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KB $\geq$ 0.

AB (input/output) REAL/COMPLEX array, dimension (LDAB, N)
On entry, the upper or lower triangle of the symmetric/Hermitian band matrix A, stored in the first ka+1 rows of the array. The jth column of A is stored in the jth column of the array AB as follows:
if UPLO = 'U', AB(ka+1+i−j,j) = A(i,j) for max(1,j−ka)$\leq$i$\leq$j;
if UPLO = 'L', AB(1+i−j,j) = A(i,j) for j$\leq$i$\leq$min(n,j+ka).
On exit, the contents of AB are destroyed.

LDAB (input) INTEGER
The leading dimension of the array AB. LDAB $\geq$ KA+1.

BB (input/output) REAL/COMPLEX array, dimension (LDBB, N)
On entry, the upper or lower triangle of the symmetric/Hermitian band matrix B, stored in the first kb+1 rows of the array. The jth column of B is stored in the jth column of the array BB as follows:
if UPLO = 'U', BB(kb+1+i−j,j) = B(i,j) for max(1,j−kb)$\leq$i$\leq$j;
if UPLO = 'L', BB(1+i−j,j) = B(i,j) for j$\leq$i$\leq$min(n,j+kb).
On exit, the factor S from the split Cholesky factorization $B = S^H*S$, as returned by SPBSTF/CPBSTF.

LDBB (input) INTEGER
The leading dimension of the array BB. LDBB $\geq$ KB+1.

W (output) REAL array, dimension (N)
If INFO = 0, the eigenvalues in ascending order.

Z (output) REAL/COMPLEX array, dimension (LDZ, N)
If JOBZ = 'V', then if INFO = 0, Z contains the matrix Z of eigenvectors, with the ith column of Z holding the eigenvector associated with W(i). The eigenvectors are normalized so that $Z^H*B*Z = I$.
If JOBZ = 'N', then Z is not referenced.

LDZ (input) INTEGER
The leading dimension of the array Z. LDZ $\geq$ 1, and if JOBZ = 'V', LDZ $\geq$ N.

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK.
If N $\leq$ 1, LWORK $\geq$ 1.
SSBGVD
If JOBZ = 'N' and N > 1, LWORK $\geq$ 3*N.
If JOBZ = 'V' and N > 1, LWORK $\geq$ 1 + 5*N + $2*N^2$.
CHBGVD
If JOBZ = 'N' and N > 1, LWORK $\geq$ N. If JOBZ = 'V' and N > 1, LWORK $\geq$ $2*N^2$.

If LWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK *CHBGVD only* (workspace) REAL array, dimension (LRWORK)

LRWORK *CHBGVD only* (input) INTEGER
The dimension of the array RWORK.
If N $\leq$ 1, LRWORK $\geq$ 1.
If JOBZ = 'N' and N > 1, LRWORK $\geq$ N.
If JOBZ = 'V' and N > 1, LRWORK $\geq$ 1 + 5*N + $2*N^2$.

If LRWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the RWORK array, returns this value as the first entry of the RWORK array, and no error message related to LRWORK is issued by XERBLA.

IWORK (workspace/output) INTEGER array, dimension (LIWORK)
On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK (input) INTEGER
The dimension of the array IWORK.
If JOBZ = 'N' or N $\leq$ 1, LIWORK $\geq$ 1.
If JOBZ = 'V' and N > 1, LIWORK $\geq$ 3 + 5*N.

If LIWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the IWORK array, returns this value as the first entry of the IWORK array, and no error message related to LIWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, and i is:
$\leq$ N: the algorithm failed to converge: i off-diagonal elements of an intermediate tridiagonal form did not converge to zero;
> N: if INFO = N + i, for 1 $\leq$ i $\leq$ N, then SPBSTF/CPBSTF returned INFO = i: B is not positive definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed.

SSBGVX/CHBGVX

```
      SUBROUTINE SSBGVX( JOBZ, RANGE, UPLO, N, KA, KB, AB, LDAB, BB,
     $                   LDBB, Q, LDQ, VL, VU, IL, IU, ABSTOL, M, W, Z,
     $                   LDZ, WORK, IWORK, IFAIL, INFO )
      CHARACTER          JOBZ, RANGE, UPLO
      INTEGER            IL, INFO, IU, KA, KB, LDAB, LDBB, LDQ, LDZ, M,
     $                   N
      REAL               ABSTOL, VL, VU
      INTEGER            IFAIL( * ), IWORK( * )
      REAL               AB( LDAB, * ), BB( LDBB, * ), Q( LDQ, * ),
     $                   W( * ), WORK( * ), Z( LDZ, * )
```

```
      SUBROUTINE CHBGVX( JOBZ, RANGE, UPLO, N, KA, KB, AB, LDAB, BB,
     $                   LDBB, Q, LDQ, VL, VU, IL, IU, ABSTOL, M, W, Z,
     $                   LDZ, WORK, RWORK, IWORK, IFAIL, INFO )
      CHARACTER          JOBZ, RANGE, UPLO
      INTEGER            IL, INFO, IU, KA, KB, LDAB, LDBB, LDQ, LDZ, M,
     $                   N
      REAL               ABSTOL, VL, VU
      INTEGER            IFAIL( * ), IWORK( * )
      REAL               RWORK( * ), W( * )
      COMPLEX            AB( LDAB, * ), BB( LDBB, * ), Q( LDQ, * ),
     $                   WORK( * ), Z( LDZ, * )
```

Purpose

SSBGVX/CHBGVX computes selected eigenvalues, and optionally, eigenvectors of a real generalized symmetric-definite/Hermitian-definite banded eigenproblem, of the form $A*x = \lambda*B*x$. Here A and B are assumed to be symmetric/Hermitian and banded, and B is also positive definite. Eigenvalues and eigenvectors can be selected by specifying either all eigenvalues, a range of values or a range of indices for the desired eigenvalues.

Arguments

JOBZ (input) CHARACTER*1
= 'N': Compute eigenvalues only;
= 'V': Compute eigenvalues and eigenvectors.

RANGE (input) CHARACTER*1
= 'A': all eigenvalues will be found.
= 'V': all eigenvalues in the half-open interval (VL,VU] will be found.
= 'I': the ILth through IU^{th} eigenvalues will be found.

UPLO (input) CHARACTER*1
= 'U': Upper triangles of A and B are stored;
= 'L': Lower triangles of A and B are stored.

N (input) INTEGER
The order of the matrices A and B. $N \geq 0$.

KA (input) INTEGER
The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. $KA \geq 0$.

KB (input) INTEGER
The number of superdiagonals of the matrix B if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. $KB \geq 0$.

AB (input/output) REAL/COMPLEX array, dimension (LDAB, N)
On entry, the upper or lower triangle of the symmetric/Hermitian band matrix A, stored in the first ka+1 rows of the array. The jth column of A is stored in the jth column of the array AB as follows:
if UPLO = 'U', $AB(ka+1+i-j,j) = A(i,j)$ for $\max(1,j-ka) \leq i \leq j$;
if UPLO = 'L', $AB(1+i-j,j) = A(i,j)$ for $j \leq i \leq \min(n,j+ka)$.
On exit, the contents of AB are destroyed.

LDAB (input) INTEGER
The leading dimension of the array AB. $LDAB \geq KA+1$.

BB (input/output) REAL/COMPLEX array, dimension (LDBB, N)
On entry, the upper or lower triangle of the symmetric/Hermitian band matrix B, stored in the first kb+1 rows of the array. The jth column of B is stored in the jth column of the array BB as follows:
if UPLO = 'U', $BB(kb+1+i-j,j) = B(i,j)$ for $\max(1,j-kb) \leq i \leq j$;
if UPLO = 'L', $BB(1+i-j,j) = B(i,j)$ for $j \leq i \leq \min(n,j+kb)$.
On exit, the factor S from the split Cholesky factorization $B = S^H*S$, as returned by SPBSTF/CPBSTF.

LDBB (input) INTEGER
The leading dimension of the array BB. $LDBB \geq KB+1$.

W (output) REAL array, dimension (N)
If INFO = 0, the eigenvalues in ascending order.

Q (output) REAL/COMPLEX array, dimension (LDQ, N)
If JOBZ = 'V', the n-by-n matrix used in the reduction of $A*x = \lambda*B*x$ to standard form, i.e. $C*x = \lambda*x$, and consequently C to tridiagonal form. If JOBZ = 'N', the array Q is not referenced.

LDQ (input) INTEGER
The leading dimension of the array Q. If JOBZ = 'N', $LDQ \geq 1$. If JOBZ = 'V', $LDQ \geq \max(1,N)$.

VL,VU (input) REAL
If RANGE='V', the lower and upper bounds of the interval to be searched for eigenvalues. $VL < VU$. Not referenced if RANGE = 'A' or 'I'.

IL,IU (input) INTEGER
If RANGE='I', the indices (in ascending order) of the smallest and largest eigenvalues to be returned. $1 \leq IL \leq IU \leq N$, if $N > 0$; $IL = 1$ and $IU = 0$ if $N = 0$. Not referenced if RANGE = 'A' or 'V'.

ABSTOL (input) REAL
The absolute error tolerance for the eigenvalues.
An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to ABSTOL + EPS*max(|a|,|b|), where EPS is the machine precision. If ABSTOL ≤ 0, then $EPS*||T||_1$ will be used in its place, where T is the tridiagonal matrix obtained by reducing AB to tridiagonal form.
Eigenvalues will be computed most accurately when ABSTOL is set to twice the underflow threshold 2*SLAMCH('S'), not zero. If this routine returns with INFO>0, indicating that some eigenvectors did not converge, try setting ABSTOL to 2*SLAMCH('S').

M (output) INTEGER
The total number of eigenvalues found. $0 \leq M \leq N$. If RANGE = 'A', M = N, and if RANGE = 'I', M = IU−IL+1.

W (output) REAL array, dimension (N)
If INFO = 0, the eigenvalues in ascending order.

Z (output) REAL/COMPLEX array, dimension (LDZ, N)
If JOBZ = 'V', then if INFO = 0, Z contains the matrix Z of eigenvectors, with the ith column of Z holding the eigenvector associated with W(i). The eigenvectors are normalized so that $Z^H*B*Z = I$.
If JOBZ = 'N', then Z is not referenced.

LDZ (input) INTEGER
The leading dimension of the array Z. LDZ $\geq$ 1, and if JOBZ = 'V', LDZ $\geq$ max(1,N).

WORK *SSBGVX* (workspace) REAL array, dimension (7*N)
CHBGVX (workspace) COMPLEX array, dimension (N)

RWORK *CHBGVX only* (workspace) REAL array, dimension (7*N)

IWORK (workspace/output) INTEGER array, dimension (5*N)

IFAIL (input) INTEGER array, dimension (M)
If JOBZ = 'V', then if INFO = 0, the first M elements of IFAIL are zero. If INFO > 0, then IFAIL contains the indices of the eigenvalues that failed to converge. If JOBZ = 'N', then IFAIL is not referenced.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, and i is:
$\leq$ N: the algorithm failed to converge: i off-diagonal elements of an intermediate tridiagonal form did not converge to zero;
> N: if INFO = N + i, for 1 $\leq$ i $\leq$ N, then SPBSTF/CPBSTF returned INFO = i: B is not positive definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed.

SSBTRD/CHBTRD

```
 SUBROUTINE SSBTRD( VECT, UPLO, N, KD, AB, LDAB, D, E, Q, LDQ,
$                   WORK, INFO )
 CHARACTER          UPLO, VECT
 INTEGER            INFO, KD, LDAB, LDQ, N
 REAL               AB( LDAB, * ), D( * ), E( * ), Q( LDQ, * ),
$                   WORK( * )

 SUBROUTINE CHBTRD( VECT, UPLO, N, KD, AB, LDAB, D, E, Q, LDQ,
$                   WORK, INFO )
 CHARACTER          UPLO, VECT
 INTEGER            INFO, KD, LDAB, LDQ, N
 REAL               D( * ), E( * )
 COMPLEX            AB( LDAB, * ), Q( LDQ, * ), WORK( * )
```

Purpose

SSBTRD/CHBTRD reduces a real/complex symmetric/Hermitian band matrix A to real symmetric tridiagonal form T by an orthogonal/unitary similarity transformation: $Q^H*A*Q = T$.

Arguments

VECT (input) CHARACTER*1
= 'N': do not form Q;
= 'V': form Q;
= 'U': update a matrix X, by forming X*Q.

UPLO (input) CHARACTER*1
= 'U': Upper triangle of A is stored;
= 'L': Lower triangle of A is stored.

N (input) INTEGER
The order of the matrix A. N $\geq$ 0.

KD (input) INTEGER
The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD $\geq$ 0.

AB (input/output) REAL/COMPLEX array, dimension (LDAB,N)
On entry, the upper or lower triangle of the symmetric/Hermitian band matrix A, stored in the first kd+1 rows of the array. The jth column of A is stored in the jth column of the array AB as follows:
if UPLO = 'U', AB(kd+1+i−j,j) = A(i,j) for max(1,j−kd)$\leq$i$\leq$j;
if UPLO = 'L', AB(1+i−j,j) = A(i,j) for j$\leq$i$\leq$min(n,j+kd).
On exit, the diagonal elements of AB are overwritten by the diagonal elements of the tridiagonal matrix T; if KD > 0, the elements on the first superdiagonal (if UPLO = 'U') or the first subdiagonal (if UPLO = 'L') are overwritten by the off-diagonal elements of T; the rest of AB is overwritten by values generated during the reduction.

LDAB (input) INTEGER
The leading dimension of the array AB. LDAB $\geq$ KD+1.

D (output) REAL array, dimension (N)
The diagonal elements of the tridiagonal matrix T.

E (output) REAL array, dimension (N−1)
The off-diagonal elements of the tridiagonal matrix T:
E(i) = T(i,i+1) if UPLO = 'U'; E(i) = T(i+1,i) if UPLO = 'L'.

Q (output) REAL/COMPLEX array, dimension (LDQ,N)
On entry:
if VECT = 'U', then Q must contain an N-by-N matrix X;
if VECT = 'N' or 'V', then Q need not be set.
On exit:
if VECT = 'V', Q contains the N-by-N orthogonal matrix Q;
if VECT = 'U', Q contains the product X*Q;
if VECT = 'N', the array Q is not referenced.

LDQ (input) INTEGER
The leading dimension of the array Q.
LDQ $\geq$ 1, and LDQ $\geq$ N if VECT = 'V' or 'U'.

WORK (workspace) REAL/COMPLEX array, dimension (N)

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SSPCON/CSPCON/CHPCON

```
      SUBROUTINE SSPCON( UPLO, N, AP, IPIV, ANORM, RCOND, WORK, IWORK,
     $                   INFO )
      CHARACTER          UPLO
      INTEGER            INFO, N
      REAL               ANORM, RCOND
      INTEGER            IPIV( * ), IWORK( * )
      REAL               AP( * ), WORK( * )

      SUBROUTINE CSPCON( UPLO, N, AP, IPIV, ANORM, RCOND, WORK, INFO )
      CHARACTER          UPLO
      INTEGER            INFO, N
      REAL               ANORM, RCOND
      INTEGER            IPIV( * )
      COMPLEX            AP( * ), WORK( * )

      SUBROUTINE CHPCON( UPLO, N, AP, IPIV, ANORM, RCOND, WORK, INFO )
      CHARACTER          UPLO
      INTEGER            INFO, N
      REAL               ANORM, RCOND
      INTEGER            IPIV( * )
      COMPLEX            AP( * ), WORK( * )
```

Purpose

SSPCON/CSPCON estimates the reciprocal of the condition number (in the 1-norm) of a real/complex symmetric packed matrix A using the factorization $A = U*D*U^T$ or $A = L*D*L^T$ computed by SSPTRF/CSPTRF.

CHPCON estimates the reciprocal of the condition number of a complex Hermitian packed matrix A using the factorization $A = U*D*U^H$ or $A = L*D*L^H$ computed by CHPTRF.

An estimate is obtained for $\|A^{-1}\|$, and the reciprocal of the condition number is computed as RCOND $= 1/(\|A\| * \|A^{-1}\|)$.

Arguments

UPLO (input) CHARACTER*1
Specifies whether the details of the factorization are stored as an upper or lower triangular matrix.
= 'U': Upper triangular, form is $A = U*D*U^T$ (SSPCON/CSPCON) or $A = U*D*U^H$ (CHPCON);
= 'L': Lower triangular, form is $A = L*D*L^T$ (SSPCON/CSPCON) or $A = L*D*L^H$ (CHPCON).

N (input) INTEGER
The order of the matrix A. N $\geq$ 0.

AP (input) REAL/COMPLEX/COMPLEX array, dimension (N*(N+1)/2)
The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by SSPTRF/CSPTRF/CHPTRF, stored as a packed triangular matrix.

IPIV (input) INTEGER array, dimension (N)
Details of the interchanges and the block structure of D as determined by SSPTRF/CSPTRF/CHPTRF.

ANORM (input) REAL
The 1-norm of the original matrix A.

RCOND (output) REAL
The reciprocal of the condition number of the matrix A, computed as RCOND $= 1/(\|A\| * \|A^{-1}\|)$.

WORK (workspace) REAL/COMPLEX/COMPLEX array, dimension (2*N)

IWORK *SSPCON only* (workspace) INTEGER array, dimension (N)

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SSPEV/CHPEV

```
      SUBROUTINE SSPEV( JOBZ, UPLO, N, AP, W, Z, LDZ, WORK, INFO )
      CHARACTER          JOBZ, UPLO
      INTEGER            INFO, LDZ, N
      REAL               AP( * ), W( * ), WORK( * ), Z( LDZ, * )

      SUBROUTINE CHPEV( JOBZ, UPLO, N, AP, W, Z, LDZ, WORK, RWORK,
     $                  INFO )
      CHARACTER          JOBZ, UPLO
      INTEGER            INFO, LDZ, N
      REAL               RWORK( * ), W( * )
      COMPLEX            AP( * ), WORK( * ), Z( LDZ, * )
```

Purpose

SSPEV/CHPEV computes all the eigenvalues and, optionally, eigenvectors of a real/complex symmetric/Hermitian matrix A in packed storage.

Arguments

JOBZ (input) CHARACTER*1
= 'N': Compute eigenvalues only;
= 'V': Compute eigenvalues and eigenvectors.

UPLO (input) CHARACTER*1
= 'U': Upper triangle of A is stored;
= 'L': Lower triangle of A is stored.

N (input) INTEGER
The order of the matrix A. $N \geq 0$.

AP (input/output) REAL/COMPLEX array, dimension (N*(N+1)/2)
On entry, the upper or lower triangle of the symmetric/Hermitian matrix A, packed columnwise in a linear array. The jth column of A is stored in the array AP as follows:
if UPLO = 'U', AP(i + (j−1)*j/2) = A(i,j) for $1 \leq i \leq j$;
if UPLO = 'L', AP(i + (j−1)*(2*n−j)/2) = A(i,j) for $j \leq i \leq n$.
On exit, AP is overwritten by values generated during the reduction to tridiagonal form. If UPLO = 'U', the diagonal and first superdiagonal of the tridiagonal matrix T overwrite the corresponding elements of A, and if UPLO = 'L', the diagonal and first subdiagonal of T overwrite the corresponding elements of A.

W (output) REAL array, dimension (N)
If INFO = 0, the eigenvalues in ascending order.

Z (output) REAL/COMPLEX array, dimension (LDZ, N)
If JOBZ = 'V', then if INFO = 0, Z contains the orthonormal eigenvectors of the matrix A, with the ith column of Z holding the eigenvector associated with W(i).
If JOBZ = 'N', then Z is not referenced.

LDZ (input) INTEGER
The leading dimension of the array Z. $LDZ \geq 1$, and if JOBZ = 'V', $LDZ \geq \max(1,N)$.

WORK *SSPEV* (workspace) REAL array, dimension (3*N)
CHPEV (workspace) COMPLEX array, dimension (max(1, 2*N−1))

RWORK *CHPEV only* (workspace) REAL array, dimension (max(1,3*N−2))

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value
> 0: the algorithm failed to converge; if INFO = i, i off-diagonal elements of an intermediate tridiagonal form did not converge to zero.

SSPEVD/CHPEVD

```
      SUBROUTINE SSPEVD( JOBZ, UPLO, N, AP, W, Z, LDZ, WORK, LWORK,
     $                   IWORK, LIWORK, INFO )
      CHARACTER          JOBZ, UPLO
      INTEGER            INFO, LDZ, LIWORK, LWORK, N
      INTEGER            IWORK( * )
      REAL               AP( * ), W( * ), WORK( * ), Z( LDZ, * )

      SUBROUTINE CHPEVD( JOBZ, UPLO, N, AP, W, Z, LDZ, WORK, LWORK,
     $                   RWORK, LRWORK, IWORK, LIWORK, INFO )
      CHARACTER          JOBZ, UPLO
      INTEGER            INFO, LDZ, LIWORK, LRWORK, LWORK, N
      INTEGER            IWORK( * )
      REAL               RWORK( * ), W( * )
      COMPLEX            AP( * ), WORK( * ), Z( LDZ, * )
```

Purpose

SSPEVD/CHPEVD computes all the eigenvalues and, optionally, eigenvectors of a real/complex symmetric/Hermitian matrix A in packed storage. If eigenvectors are desired, it uses a divide and conquer algorithm.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Arguments

JOBZ (input) CHARACTER*1
= 'N': Compute eigenvalues only;
= 'V': Compute eigenvalues and eigenvectors.

UPLO (input) CHARACTER*1
= 'U': Upper triangle of A is stored;
= 'L': Lower triangle of A is stored.

N (input) INTEGER
The order of the matrix A. $N \geq 0$.

AP (input/output) REAL/COMPLEX array, dimension (N*(N+1)/2)
On entry, the upper or lower triangle of the symmetric/Hermitian matrix A, packed columnwise in a linear array. The jth column of A is stored in the array AP as follows:
if UPLO = 'U', AP(i + (j−1)*j/2) = A(i,j) for $1 \leq i \leq j$;
if UPLO = 'L', AP(i + (j−1)*(2*n−j)/2) = A(i,j) for $j \leq i \leq n$.
On exit, AP is overwritten by values generated during the reduction to tridiagonal form. If UPLO = 'U', the diagonal and first superdiagonal

of the tridiagonal matrix T overwrite the corresponding elements of A, and if UPLO = 'L', the diagonal and first subdiagonal of T overwrite the corresponding elements of A.

W (output) REAL array, dimension (N)
If INFO = 0, the eigenvalues in ascending order.

Z (output) REAL/COMPLEX array, dimension (LDZ, N)
If JOBZ = 'V', then if INFO = 0, Z contains the orthonormal eigenvectors of the matrix A, with the ith column of Z holding the eigenvector associated with W(i).
If JOBZ = 'N', then Z is not referenced.

LDZ (input) INTEGER
The leading dimension of the array Z. LDZ $\geq$ 1, and if JOBZ = 'V', LDZ $\geq$ max(1,N).

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK.
If N $\leq$ 1, LWORK $\geq$ 1.
SSPEVD
If JOBZ = 'N' and N > 1, LWORK $\geq$ 2*N.
If JOBZ = 'V' and N > 1, LWORK $\geq$ $(1 + 6*N + 2*N^2)$.
CHPEVD
If JOBZ = 'N' and N > 1, LWORK $\geq$ N.
If JOBZ = 'V' and N > 1, LWORK $\geq$ 2*N.

If LWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK *CHPEVD only* (workspace/output) REAL array, dimension (LRWORK)
On exit, if INFO = 0, RWORK(1) returns the optimal LRWORK.

LRWORK *CHPEVD only* (input) INTEGER
The dimension of the array RWORK.
If N $\leq$ 1, LRWORK $\geq$ 1.
If JOBZ = 'N' and N > 1, LRWORK $\geq$ N.
If JOBZ = 'V' and N > 1, LRWORK $\geq$ $(1 + 5*N + 2*N^2)$.

If LRWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the RWORK array, returns this value as the first entry of the RWORK array, and no error message related to LRWORK is issued by XERBLA.

IWORK (workspace/output) INTEGER array, dimension (LIWORK)
On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK (input) INTEGER
The dimension of the array IWORK.
If JOBZ = 'N' or N $\leq$ 1, LIWORK $\geq$ 1.
If JOBZ = 'V' and N > 1, LIWORK $\geq$ 3 + 5*N.

If LIWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the IWORK array, returns this value as the first entry of the IWORK array, and no error message related to LIWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero.

SSPEVX/CHPEVX

```
      SUBROUTINE SSPEVX( JOBZ, RANGE, UPLO, N, AP, VL, VU, IL, IU,
     $                   ABSTOL, M, W, Z, LDZ, WORK, IWORK, IFAIL,
     $                   INFO )
      CHARACTER          JOBZ, RANGE, UPLO
      INTEGER            IL, INFO, IU, LDZ, M, N
      REAL               ABSTOL, VL, VU
      INTEGER            IFAIL( * ), IWORK( * )
      REAL               AP( * ), W( * ), WORK( * ), Z( LDZ, * )

      SUBROUTINE CHPEVX( JOBZ, RANGE, UPLO, N, AP, VL, VU, IL, IU,
     $                   ABSTOL, M, W, Z, LDZ, WORK, RWORK, IWORK,
     $                   IFAIL, INFO )
      CHARACTER          JOBZ, RANGE, UPLO
      INTEGER            IL, INFO, IU, LDZ, M, N
      REAL               ABSTOL, VL, VU
      INTEGER            IFAIL( * ), IWORK( * )
      REAL               RWORK( * ), W( * )
      COMPLEX            AP( * ), WORK( * ), Z( LDZ, * )
```

Purpose

SSPEVX/CHPEVX computes selected eigenvalues and, optionally, eigenvectors of a real/complex symmetric/Hermitian matrix A in packed storage. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

Arguments

JOBZ (input) CHARACTER*1

= 'N': Compute eigenvalues only;
= 'V': Compute eigenvalues and eigenvectors.

RANGE (input) CHARACTER*1
= 'A': all eigenvalues will be found;
= 'V': all eigenvalues in the half-open interval (VL,VU] will be found;
= 'I': the ILth through IU^{th} eigenvalues will be found.

UPLO (input) CHARACTER*1
= 'U': Upper triangle of A is stored;
= 'L': Lower triangle of A is stored.

N (input) INTEGER
The order of the matrix A. $N \geq 0$.

AP (input/output) REAL/COMPLEX array, dimension (N*(N+1)/2)
On entry, the upper or lower triangle of the symmetric/Hermitian matrix A, packed columnwise in a linear array. The jth column of A is stored in the array AP as follows:
if UPLO = 'U', AP(i + (j−1)*j/2) = A(i,j) for $1 \leq i \leq j$;
if UPLO = 'L', AP(i + (j−1)*(2*n−j)/2) = A(i,j) for $j \leq i \leq n$.
On exit, AP is overwritten by values generated during the reduction to tridiagonal form. If UPLO = 'U', the diagonal and first superdiagonal of the tridiagonal matrix T overwrite the corresponding elements of A, and if UPLO = 'L', the diagonal and first subdiagonal of T overwrite the corresponding elements of A.

VL, VU (input) REAL
If RANGE='V', the lower and upper bounds of the interval to be searched for eigenvalues. VL < VU.
Not referenced if RANGE = 'A' or 'I'.

IL, IU (input) INTEGER
If RANGE='I', the indices (in ascending order) of the smallest and largest eigenvalues to be returned.
$1 \leq IL \leq IU \leq N$, if $N > 0$; IL = 1 and IU = 0 if N = 0.
Not referenced if RANGE = 'A' or 'V'.

ABSTOL (input) REAL
The absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to ABSTOL + EPS*max(|a|,|b|), where EPS is the machine precision. If ABSTOL is less than or equal to zero, then $EPS*||T||_1$ will be used in its place, where T is the tridiagonal matrix obtained by reducing A to tridiagonal form.
Eigenvalues will be computed most accurately when ABSTOL is set to twice the underflow threshold 2*SLAMCH('S'), not zero. If this routine returns with INFO>0, indicating that some eigenvectors did not converge, try setting ABSTOL to 2*SLAMCH('S').

M (output) INTEGER
The total number of eigenvalues found. $0 \leq M \leq N$.
If RANGE = 'A', M = N, and if RANGE = 'I', M = IU−IL+1.

W (output) REAL array, dimension (N)
If INFO = 0, the selected eigenvalues in ascending order.

Z (output) REAL/COMPLEX array, dimension (LDZ, max(1,M))
If JOBZ = 'V', then if INFO = 0, the first M columns of Z contain the orthonormal eigenvectors of the matrix A corresponding to the selected eigenvalues, with the ith column of Z holding the eigenvector asociated with W(i). If an eigenvector fails to converge, then that column of Z contains the latest approximation to the eigenvector, and the index of the eigenvector is returned in IFAIL.
If JOBZ = 'N', then Z is not referenced.
Note: the user must ensure that at least max(1,M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and an upper bound must be used.

LDZ (input) INTEGER
The leading dimension of the array Z. $LDZ \geq 1$, and if JOBZ = 'V', $LDZ \geq \max(1,N)$.

WORK *SSPEVX* (workspace) REAL array, dimension (8*N)
CHPEVX (workspace) COMPLEX array, dimension (2*N)

RWORK *CHPEVX only* (workspace) REAL array, dimension (7*N)

IWORK (workspace) INTEGER array, dimension (5*N)

IFAIL (output) INTEGER array, dimension (N)
If JOBZ = 'V', then if INFO = 0, the first M elements of IFAIL are zero; if INFO > 0, then IFAIL contains the indices of the eigenvectors that failed to converge.
If JOBZ = 'N', then IFAIL is not referenced.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, then i eigenvectors failed to converge. Their indices are stored in array IFAIL.

SSPGST/CHPGST

```
SUBROUTINE SSPGST( ITYPE, UPLO, N, AP, BP, INFO )
CHARACTER          UPLO
INTEGER            INFO, ITYPE, N
REAL               AP( * ), BP( * )

SUBROUTINE CHPGST( ITYPE, UPLO, N, AP, BP, INFO )
CHARACTER          UPLO
INTEGER            INFO, ITYPE, N
COMPLEX            AP( * ), BP( * )
```

Purpose

SSPGST/CHPGST reduces a real/complex symmetric-definite/Hermitian-definite generalized eigenproblem to standard form, using packed storage.

If ITYPE = 1, the problem is $A*x = \lambda*B*x$, and A is overwritten by $(U^H)^{-1}*A*U^{-1}$ or $L^{-1}*A*(L^H)^{-1}$

If ITYPE = 2 or 3, the problem is $A*B*x = \lambda*x$ or $B*A*x = \lambda*x$, and A is overwritten by $U*A*U^H$ or L^H*A*L.

B must have been previously factorized as U^H*U or $L*L^H$ by SPPTRF/CPPTRF.

Arguments

ITYPE (input) INTEGER
= 1: compute $(U^H)^{-1}*A*U^{-1}$ or $L^{-1}*A*(L^H)^{-1}$;
= 2 or 3: compute $U*A*U^H$ or L^H*A*L.

UPLO (input) CHARACTER
= 'U': Upper triangle of A is stored and B is factored as U^H*U;
= 'L': Lower triangle of A is stored and B is factored as $L*L^H$.

N (input) INTEGER
The order of the matrices A and B. $N \geq 0$.

AP (input/output) REAL/COMPLEX array, dimension (N*(N+1)/2)
On entry, the upper or lower triangle of the symmetric/Hermitian matrix A, packed columnwise in a linear array. The jth column of A is stored in the array AP as follows:
if UPLO = 'U', AP(i + (j−1)*j/2) = A(i,j) for $1 \leq i \leq j$;
if UPLO = 'L', AP(i + (j−1)*(2*n−j)/2) = A(i,j) for $j \leq i \leq n$.
On exit, if INFO = 0, the transformed matrix, stored in the same format as A.

BP (input) REAL/COMPLEX array, dimension (N*(N+1)/2)
The triangular factor from the Cholesky factorization of B, stored in the same format as A, as returned by SPPTRF/CPPTRF.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SSPGV/CHPGV

```
      SUBROUTINE SSPGV( ITYPE, JOBZ, UPLO, N, AP, BP, W, Z, LDZ, WORK,
     $                  INFO )
      CHARACTER          JOBZ, UPLO
      INTEGER            INFO, ITYPE, LDZ, N
      REAL               AP( * ), BP( * ), W( * ), WORK( * ),
     $                   Z( LDZ, * )
```

```
      SUBROUTINE CHPGV( ITYPE, JOBZ, UPLO, N, AP, BP, W, Z, LDZ, WORK,
     $                  RWORK, INFO )
      CHARACTER          JOBZ, UPLO
      INTEGER            INFO, ITYPE, LDZ, N
      REAL               RWORK( * ), W( * )
      COMPLEX            AP( * ), BP( * ), WORK( * ), Z( LDZ, * )
```

Purpose

SSPGV/CHPGV computes all the eigenvalues and, optionally, the eigenvectors of a real/complex generalized symmetric-definite/Hermitian-definite eigenproblem, of the form

$$A*x=\lambda*B*x,\ A*B*x=\lambda*x,\ \text{or}\ B*A*x=\lambda*x.$$

Here A and B are assumed to be symmetric/Hermitian, stored in packed format, and B is also positive definite.

Arguments

ITYPE (input) INTEGER
Specifies the problem type to be solved:
= 1: $A*x = \lambda*B*x$
= 2: $A*B*x = \lambda*x$
= 3: $B*A*x = \lambda*x$

JOBZ (input) CHARACTER*1
= 'N': Compute eigenvalues only;
= 'V': Compute eigenvalues and eigenvectors.

UPLO (input) CHARACTER*1
= 'U': Upper triangles of A and B are stored;
= 'L': Lower triangles of A and B are stored.

N (input) INTEGER
The order of the matrices A and B. $N \geq 0$.

AP (input/output) REAL/COMPLEX array, dimension (N*(N+1)/2)
On entry, the upper or lower triangle of the symmetric/Hermitian matrix A, packed columnwise in a linear array. The jth column of A is stored in the array AP as follows:
if UPLO = 'U', AP(i + (j−1)*j/2) = A(i,j) for $1 \leq i \leq j$;
if UPLO = 'L', AP(i + (j−1)*(2*n−j)/2) = A(i,j) for $j \leq i \leq n$.
On exit, the contents of AP are destroyed.

BP (input/output) REAL/COMPLEX array, dimension (N*(N+1)/2)
On entry, the upper or lower triangle of the symmetric/Hermitian matrix B, packed columnwise in a linear array. The jth column of B is stored in the array BP as follows:
if UPLO = 'U', BP(i + (j−1)*j/2) = B(i,j) for $1 \leq i \leq j$;
if UPLO = 'L', BP(i + (j−1)*(2*n−j)/2) = B(i,j) for $j \leq i \leq n$.
On exit, the triangular factor U or L from the Cholesky factorization $B = U^H*U$ or $B = L*L^H$, in the same storage format as B.

W (output) REAL array, dimension (N)
If INFO = 0, the eigenvalues in ascending order.

Z (output) REAL/COMPLEX array, dimension (LDZ, N)
If JOBZ = 'V', then if INFO = 0, Z contains the matrix Z of eigenvectors. The eigenvectors are normalized as follows:
if ITYPE = 1 or 2, Z^H*B*Z = I;
if ITYPE = 3, Z^H*B^{-1}*Z = I.
If JOBZ = 'N', then Z is not referenced.

LDZ (input) INTEGER
The leading dimension of the array Z. LDZ $\geq$ 1, and if JOBZ = 'V', LDZ $\geq$ max(1,N).

WORK *SSPGV* (workspace) REAL array, dimension (3*N)
CHPGV (workspace) COMPLEX array, dimension (max(1, 2*N−1))

RWORK *CHPGV only* (workspace) REAL array, dimension (max(1,3*N−2))

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: SPPTRF/CPPTRF or SSPEV/CHPEV returned an error code:
$\leq$ N: SSPEV/CHPEV failed to converge; if INFO = i, i off-diagonal elements of an intermediate tridiagonal form did not converge to zero;
> N: if INFO = N + i, for 1 $\leq$ i $\leq$ N, then the leading minor of order i of B is not positive definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed.

SSPGVD/CHPGVD

```
      SUBROUTINE SSPGVD( ITYPE, JOBZ, UPLO, N, AP, BP, W, Z, LDZ, WORK,
     $                   LWORK, IWORK, LIWORK, INFO )
      CHARACTER          JOBZ, UPLO
      INTEGER            INFO, ITYPE, LDZ, LIWORK, LWORK, N
      INTEGER            IWORK( * )
      REAL               AP( * ), BP( * ), W( * ), WORK( * ),
     $                   Z( LDZ, * )

      SUBROUTINE CHPGVD( ITYPE, JOBZ, UPLO, N, AP, BP, W, Z, LDZ, WORK,
     $                   LWORK, RWORK, LRWORK, IWORK, LIWORK, INFO )
      CHARACTER          JOBZ, UPLO
      INTEGER            INFO, ITYPE, LDZ, LIWORK, LRWORK, LWORK, N
      INTEGER            IWORK( * )
      REAL               RWORK( * ), W( * )
      COMPLEX            AP( * ), BP( * ), WORK( * ), Z( LDZ, * )
```

Purpose

SSPGVD/CHPGVD computes all the eigenvalues, and optionally, the eigenvectors of a real generalized symmetric-definite/Hermitian-definite eigenproblem, of the form $A*x = \lambda*B*x$, $A*Bx = \lambda*x$, or $B*A*x = \lambda*x$. Here A and B are assumed to be symmetric/Hermitian, stored in packed format, and B is also positive definite. If eigenvectors are desired, it uses a divide and conquer algorithm.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Arguments

ITYPE (input) INTEGER
Specifies the problem type to be solved:
= 1: A*x = λ*B*x
= 2: A*B*x = λ*x
= 3: B*A*x = λ*x

JOBZ (input) CHARACTER*1
= 'N': Compute eigenvalues only;
= 'V': Compute eigenvalues and eigenvectors.

UPLO (input) CHARACTER*1
= 'U': Upper triangles of A and B are stored;
= 'L': Lower triangles of A and B are stored.

N (input) INTEGER
The order of the matrices A and B. N $\geq$ 0.

AP (input/output) REAL/COMPLEX array, dimension (N*(N+1)/2)
On entry, the upper or lower triangle of the symmetric/Hermitian matrix A, packed columnwise in a linear array. The jth column of A is stored in the array AP as follows:
if UPLO = 'U', AP(i + (j−1)*j/2) = A(i,j) for 1$\leq$i$\leq$j;
if UPLO = 'L', AP(i + (j−1)*(2*n−j)/2) = A(i,j) for j$\leq$i$\leq$n.
On exit, the contents of AP are destroyed.

BP (input/output) REAL/COMPLEX array, dimension (N*(N+1)/2)
On entry, the upper or lower triangle of the symmetric/Hermitian matrix A, packed columnwise in a linear array. The jth column of B is stored in the array BP as follows:
if UPLO = 'U', BP(i + (j−1)*j/2) = B(i,j) for 1$\leq$i$\leq$j;
if UPLO = 'L', BP(i + (j−1)*(2*n−j)/2) = B(i,j) for j$\leq$i$\leq$n.

On exit, the triangular factor U or L from the Cholesky factorization $B = U^H*U$ or $B = L*L^H$, in the same storage format as B.

W (output) REAL array, dimension (N)
If INFO = 0, the eigenvalues in ascending order.

Z (output) REAL/COMPLEX array, dimension (LDZ, N)
If JOBZ = 'V', then if INFO = 0, Z contains the matrix Z of eigenvectors. The eigenvectors are normalized as follows:
if ITYPE = 1 or 2, $Z^H * B * Z = I$;
if ITYPE = 3, $Z^H * B^{-1} * Z = I$.
If JOBZ = 'N', then Z is not referenced.

LDZ (input) INTEGER
The leading dimension of the array Z. LDZ $\geq$ 1, and if JOBZ = 'V', LDZ $\geq$ max(1,N).

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK. If N $\leq$ 1, LWORK $\geq$ 1.
SSPGVD
If JOBZ = 'N' and N > 1, LWORK $\geq$ 2*N.
If JOBZ = 'V' and N > 1, LWORK $\geq$ 1 + 6*N + 2*N^2.
CHPGVD
If JOBZ = 'N' and N > 1, LWORK $\geq$ N.
If JOBZ = 'V' and N > 1, LWORK $\geq$ 2*N.

If LWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK *CHPGVD only* (workspace) REAL array, dimension (LRWORK)
On exit, if INFO = 0, RWORK(1) returns the optimal LRWORK.

LRWORK *CHPGVD only* (input) INTEGER
The dimension of array RWORK.
If N $\leq$ 1, LRWORK $\geq$ 1.
If JOBZ = 'N' and N > 1, LRWORK $\geq$ N. If JOBZ = 'V' and N > 1, LRWORK $\geq$ 1 + 5*N + 2*N^2.

If LRWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the RWORK array, returns this value as the first entry of the RWORK array, and no error message related to LRWORK is issued by XERBLA.

IWORK (workspace/output) INTEGER array, dimension (LIWORK)
On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK (input) INTEGER
The dimension of the array IWORK.
If JOBZ = 'N' or N $\leq$ 1, LIWORK $\geq$ 1. If JOBZ = 'V' and N > 1, LIWORK $\geq$ 3 + 5*N.

If LIWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the IWORK array, returns this value as the first entry of the IWORK array, and no error message related to LIWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: SPPTRF/CPPTRF or SSPEVD/CHPEVD returned an error code:
$\leq$ N: if INFO = i, SSPEVD/CHPEVD failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero;
> N: if INFO = N + i, for 1 $\leq$ i $\leq$ N, then the leading minor of order i of B is not positive definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed.

SSPGVX/CHPGVX

```
      SUBROUTINE SSPGVX( ITYPE, JOBZ, RANGE, UPLO, N, AP, BP, VL, VU,
     $                   IL, IU, ABSTOL, M, W, Z, LDZ, WORK, IWORK,
     $                   IFAIL, INFO )
      CHARACTER          JOBZ, RANGE, UPLO
      INTEGER            IL, INFO, ITYPE, IU, LDZ, M, N
      REAL               ABSTOL, VL, VU
      INTEGER            IFAIL( * ), IWORK( * )
      REAL               AP( * ), BP( * ), W( * ), WORK( * ),
     $                   Z( LDZ, * )

      SUBROUTINE CHPGVX( ITYPE, JOBZ, RANGE, UPLO, N, AP, BP, VL, VU,
     $                   IL, IU, ABSTOL, M, W, Z, LDZ, WORK, RWORK,
     $                   IWORK, IFAIL, INFO )
      CHARACTER          JOBZ, RANGE, UPLO
      INTEGER            IL, INFO, ITYPE, IU, LDZ, M, N
      REAL               ABSTOL, VL, VU
      INTEGER            IFAIL( * ), IWORK( * )
      REAL               RWORK( * ), W( * )
      COMPLEX            AP( * ), BP( * ), WORK( * ), Z( LDZ, * )
```

Purpose

SSPGVX/CHPGVX computes selected eigenvalues and, optionally, eigenvectors of a complex generalized symmetric-definite/Hermitian-definite eigenproblem, of the form $A*x = \lambda*B*x$, $A*Bx = \lambda*x$, or $B*A*x = \lambda*x$. Here A and B are assumed to be symmetric/Hermitian, stored in packed format, and B is also positive definite. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

Arguments

ITYPE (input) INTEGER
Specifies the problem type to be solved:
= 1: A*x = λ*B*x
= 2: A*B*x = λ*x
= 3: B*A*x = λ*x

JOBZ (input) CHARACTER*1
= 'N': Compute eigenvalues only;
= 'V': Compute eigenvalues and eigenvectors.

RANGE (input) CHARACTER*1
= 'A': all eigenvalues will be found.
= 'V': all eigenvalues in the half-open interval (VL,VU] will be found.
= 'I': the ILth through IU^{th} eigenvalues will be found.

UPLO (input) CHARACTER*1
= 'U': Upper triangles of A and B are stored;
= 'L': Lower triangles of A and B are stored.

N (input) INTEGER
The order of the matrices A and B. N ≥ 0.

AP (input/output) REAL/COMPLEX array, dimension (N*(N+1)/2)
On entry, the upper or lower triangle of the symmetric/Hermitian matrix A, packed columnwise in a linear array. The jth column of A is stored in the array AP as follows:
if UPLO = 'U', AP(i + (j−1)*j/2) = A(i,j) for 1≤i≤j;
if UPLO = 'L', AP(i + (j−1)*(2*n−j)/2) = A(i,j) for j≤i≤n.
On exit, the contents of AP are destroyed.

BP (input/output) REAL/COMPLEX array, dimension (N*(N+1)/2)
On entry, the upper or lower triangle of the symmetric/Hermitian matrix A, packed columnwise in a linear array. The jth column of B is stored in the array BP as follows:
if UPLO = 'U', BP(i + (j−1)*j/2) = B(i,j) for 1≤i≤j;
if UPLO = 'L', BP(i + (j−1)*(2*n−j)/2) = B(i,j) for j≤i≤n.

On exit, the triangular factor U or L from the Cholesky factorization $B = U^H * U$ or $B = L * L^H$, in the same storage format as B.

VL,VU (input) REAL
If RANGE='V', the lower and upper bounds of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

IL,IU (input) INTEGER
If RANGE='I', the indices (in ascending order) of the smallest and largest eigenvalues to be returned. 1 ≤ IL ≤ IU ≤ N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

ABSTOL (input) REAL
The absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to ABSTOL + EPS*max(|a|,|b|), where EPS is the machine precision. If ABSTOL is less than or equal to zero, then EPS*$||T||_1$ will be used in its place, where T is the tridiagonal matrix obtained by reducing A to tridiagonal form.
Eigenvalues will be computed most accurately when ABSTOL is set to twice the underflow threshold 2*SLAMCH('S'), not zero. If this routine returns with INFO>0, indicating that some eigenvectors did not converge, try setting ABSTOL to 2*SLAMCH('S').

M (output) INTEGER
The total number of eigenvalues found. 0 ≤ M ≤ N.
If RANGE = 'A', M = N, and if RANGE = 'I', M = IU−IL+1.

W (output) REAL array, dimension (N)
If INFO = 0, the eigenvalues in ascending order.

Z (output) REAL/COMPLEX array, dimension (LDZ, N)
If JOBZ = 'N', then Z is not referenced.
If JOBZ = 'V', then if INFO = 0, the first M columns of Z contain the orthonormal eigenvectors of the matrix A corresponding to the selected eigenvalues, with the ith column of Z holding the eigenvector associated with W(i). The eigenvectors are normalized as follows:
if ITYPE = 1 or 2, $Z^H * B * Z = I$;
if ITYPE = 3, $Z^H * B^{-1} * Z = I$.

If an eigenvector fails to converge, then that column of Z contains the latest approximation to the eigenvector, and the index of the eigenvector is returned in IFAIL. Note: the user must ensure that at least max(1,M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and an upper bound must be used.

LDZ (input) INTEGER
The leading dimension of the array Z. LDZ ≥ 1, and if JOBZ = 'V', LDZ ≥ max(1,N).

WORK *SSPGVX* (workspace) REAL array, dimension (8*N)
CHPGVX (workspace) COMPLEX array, dimension (2*N)

RWORK *CHPGVX only* (workspace) REAL array, dimension (7*N)

IWORK (workspace) INTEGER array, dimension (5*N)

IFAIL (output) INTEGER array, dimension (N)
If JOBZ = 'V', then if INFO = 0, the first M elements of IFAIL are zero. If INFO > 0, then IFAIL contains the indices of the eigenvectors that failed to converge. If JOBZ = 'N', then IFAIL is not referenced.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = -i, the i-th argument had an illegal value.
> 0: SPPTRF/CPPTRF or SSPEVX/CHPEVX returned an error code:

$\leq$ N: if INFO = i, SSPEVX/CHPEVX failed to converge; i eigenvectors failed to converge. Their indices are stored in array IFAIL.

> N: if INFO = N + i, for $1 \leq i \leq n$, then the leading minor of order i of B is not positive definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed.

SSPRFS/CSPRFS/CHPRFS

```
      SUBROUTINE SSPRFS( UPLO, N, NRHS, AP, AFP, IPIV, B, LDB, X, LDX,
     $                   FERR, BERR, WORK, IWORK, INFO )
      CHARACTER          UPLO
      INTEGER            INFO, LDB, LDX, N, NRHS
      INTEGER            IPIV( * ), IWORK( * )
      REAL               AFP( * ), AP( * ), B( LDB, * ), BERR( * ),
     $                   FERR( * ), WORK( * ), X( LDX, * )

      SUBROUTINE CSPRFS( UPLO, N, NRHS, AP, AFP, IPIV, B, LDB, X, LDX,
     $                   FERR, BERR, WORK, RWORK, INFO )
      CHARACTER          UPLO
      INTEGER            INFO, LDB, LDX, N, NRHS
      INTEGER            IPIV( * )
      REAL               BERR( * ), FERR( * ), RWORK( * )
      COMPLEX            AFP( * ), AP( * ), B( LDB, * ), WORK( * ),
     $                   X( LDX, * )

      SUBROUTINE CHPRFS( UPLO, N, NRHS, AP, AFP, IPIV, B, LDB, X, LDX,
     $                   FERR, BERR, WORK, RWORK, INFO )
      CHARACTER          UPLO
      INTEGER            INFO, LDB, LDX, N, NRHS
      INTEGER            IPIV( * )
      REAL               BERR( * ), FERR( * ), RWORK( * )
      COMPLEX            AFP( * ), AP( * ), B( LDB, * ), WORK( * ),
     $                   X( LDX, * )
```

Purpose

SSPRFS/CSPRFS/CHPRFS improves the computed solution to a system of linear equations when the coefficient matrix is real/complex/complex symmetric/-symmetric/Hermitian indefinite and packed, and provides error bounds and backward error estimates for the solution.

Arguments

UPLO (input) CHARACTER*1
= 'U': Upper triangle of A is stored;
= 'L': Lower triangle of A is stored.

N (input) INTEGER
The order of the matrix A. $N \geq 0$.

NRHS (input) INTEGER
The number of right hand sides, i.e., the number of columns of the matrices B and X. $NRHS \geq 0$.

AP (input) REAL/COMPLEX/COMPLEX array, dimension (N*(N+1)/2)
The upper or lower triangle of the symmetric/symmetric/Hermitian matrix A, packed columnwise in a linear array. The jth column of A is stored in the array AP as follows:
if UPLO = 'U', AP(i + (j−1)*j/2) = A(i,j) for $1 \leq i \leq j$;
if UPLO = 'L', AP(i + (j−1)*(2*n−j)/2) = A(i,j) for $j \leq i \leq n$.

AFP (input) REAL/COMPLEX/COMPLEX array, dimension (N*(N+1)/2)
The factored form of the matrix A. AFP contains the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization $A = U*D*U^T$ or $A = L*D*L^T$ as computed by SSPTRF/CSPTRF or the factorization $A = U*D*U^H$ or $A = L*D*L^H$ as computed by CHPTRF, stored as a packed triangular matrix.

IPIV (input) INTEGER array, dimension (N)
Details of the interchanges and the block structure of D as determined by SSPTRF/CSPTRF/CHPTRF.

B (input) REAL/COMPLEX/COMPLEX array, dimension (LDB,NRHS)
The right hand side matrix B.

LDB (input) INTEGER
The leading dimension of the array B. $LDB \geq \max(1,N)$.

X (input/output)
REAL/COMPLEX/COMPLEX array, dimension (LDX,NRHS)
On entry, the solution matrix X, as computed by SSPTRS/CSPTRS/CHPTRS.
On exit, the improved solution matrix X.

LDX (input) INTEGER
The leading dimension of the array X. $LDX \geq \max(1,N)$.

FERR (output) REAL array, dimension (NRHS)
The estimated forward error bound for each solution vector X(j) (the jth column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) − XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR (output) REAL array, dimension (NRHS)
The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK *SSPRFS* (workspace) REAL array, dimension (3*N)
CSPRFS/CHPRFS (workspace) COMPLEX array, dimension (2*N)

IWORK *SSPRFS only* (workspace) INTEGER array, dimension (N)

RWORK *CSPRFS/CHPRFS only* (workspace) REAL array, dimension (N)

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SSPSV/CSPSV/CHPSV

```
SUBROUTINE SSPSV( UPLO, N, NRHS, AP, IPIV, B, LDB, INFO )
CHARACTER          UPLO
INTEGER            INFO, LDB, N, NRHS
INTEGER            IPIV( * )
REAL               AP( * ), B( LDB, * )

SUBROUTINE CSPSV( UPLO, N, NRHS, AP, IPIV, B, LDB, INFO )
CHARACTER          UPLO
INTEGER            INFO, LDB, N, NRHS
INTEGER            IPIV( * )
COMPLEX            AP( * ), B( LDB, * )

SUBROUTINE CHPSV( UPLO, N, NRHS, AP, IPIV, B, LDB, INFO )
CHARACTER          UPLO
INTEGER            INFO, LDB, N, NRHS
INTEGER            IPIV( * )
COMPLEX            AP( * ), B( LDB, * )
```

Purpose

SSPSV/CSPSV/CHPSV computes the solution to a real/complex/complex system of linear equations A*X = B, where A is an n-by-n symmetric (SSPSV/CSPSV) or Hermitian (CHPSV) matrix stored in packed format and X and B are n-by-nrhs matrices.

The diagonal pivoting method is used to factor A as

$A = U*D*U^T$ or $A = L*D*L^T$ (SSPSV/CSPSV) or

$A = U*D*U^H$ or $A = L*D*L^H$ (CHPSV),

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and D is symmetric (SSPSV/CSPSV) or Hermitian (CHPSV) and block diagonal with 1-by-1 and 2-by-2 diagonal blocks. The factored form of A is then used to solve the system of equations A*X = B.

Arguments

UPLO (input) CHARACTER*1
= 'U': Upper triangle of A is stored;
= 'L': Lower triangle of A is stored.

N (input) INTEGER
The number of linear equations, i.e., the order of the matrix A. $N \geq 0$.

NRHS (input) INTEGER
The number of right hand sides, i.e., the number of columns of the matrix B. $NRHS \geq 0$.

AP (input/output)
REAL/COMPLEX/COMPLEX array, dimension (N*(N+1)/2)
On entry, the upper or lower triangle of the symmetric/symmetric/Hermitian matrix A, packed columnwise in a linear array. The jth column of A is stored in the array AP as follows:
if UPLO = 'U', AP(i + (j−1)*j/2) = A(i,j) for $1 \leq i \leq j$;
if UPLO = 'L', AP(i + (j−1)*(2*n−j)/2) = A(i,j) for $j \leq i \leq n$.
On exit, the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization $A = U*D*U^T$ or $A = L*D*L^T$ as computed by SSPTRF/CSPTRF or the factorization $A = U*D*U^H$ or $A = L*D*L^H$ as computed by CHPTRF, stored as a packed triangular matrix in the same storage format as A.

IPIV (output) INTEGER array, dimension (N)
Details of the interchanges and the block structure of D, as determined by SSPTRF/CSPTRF/CHPTRF. If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged, and D(k,k) is a 1-by-1 diagonal block. If UPLO = 'U' and IPIV(k) = IPIV(k−1) < 0, then rows and columns k−1 and −IPIV(k) were interchanged and D(k−1:k,k−1:k) is a 2-by-2 diagonal block. If UPLO = 'L' and IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and −IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

B (input/output)
REAL/COMPLEX/COMPLEX array, dimension (LDB,NRHS)
On entry, the n-by-nrhs right hand side matrix B.
On exit, if INFO = 0, the n-by-nrhs solution matrix X.

LDB (input) INTEGER
The leading dimension of the array B. $LDB \geq \max(1,N)$.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, D(i,i) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, so the solution could not be computed.

SSPSVX/CSPSVX/CHPSVX

```
      SUBROUTINE SSPSVX( FACT, UPLO, N, NRHS, AP, AFP, IPIV, B, LDB, X,
     $                   LDX, RCOND, FERR, BERR, WORK, IWORK, INFO )
      CHARACTER          FACT, UPLO
      INTEGER            INFO, LDB, LDX, N, NRHS
      REAL               RCOND
      INTEGER            IPIV( * ), IWORK( * )
      REAL               AFP( * ), AP( * ), B( LDB, * ), BERR( * ),
     $                   FERR( * ), WORK( * ), X( LDX, * )

      SUBROUTINE CSPSVX( FACT, UPLO, N, NRHS, AP, AFP, IPIV, B, LDB, X,
     $                   LDX, RCOND, FERR, BERR, WORK, RWORK, INFO )
      CHARACTER          FACT, UPLO
      INTEGER            INFO, LDB, LDX, N, NRHS
      REAL               RCOND
      INTEGER            IPIV( * )
      REAL               BERR( * ), FERR( * ), RWORK( * )
      COMPLEX            AFP( * ), AP( * ), B( LDB, * ), WORK( * ),
     $                   X( LDX, * )

      SUBROUTINE CHPSVX( FACT, UPLO, N, NRHS, AP, AFP, IPIV, B, LDB, X,
     $                   LDX, RCOND, FERR, BERR, WORK, RWORK, INFO )
      CHARACTER          FACT, UPLO
      INTEGER            INFO, LDB, LDX, N, NRHS
      REAL               RCOND
      INTEGER            IPIV( * )
      REAL               BERR( * ), FERR( * ), RWORK( * )
      COMPLEX            AFP( * ), AP( * ), B( LDB, * ), WORK( * ),
     $                   X( LDX, * )
```

Purpose

SSPSVX/CSPSVX/CHPSVX uses the diagonal pivoting factorization to compute the solution to a real/complex/complex system of linear equations A*X = B, where A is an n-by-n symmetric (SSPSVX/CSPSVX) or Hermitian (CHPSVX) matrix stored in packed format and X and B are n-by-nrhs matrices.

Error bounds on the solution and a condition estimate are also computed.

Description

The following steps are performed:

1. If FACT = 'N', the diagonal pivoting method is used to factor A as

   $A = U*D*U^T$ or $A = L*D*L^T$ (SSPSVX/CSPSVX) or

   $A = U*D*U^H$ or $A = L*D*L^H$ (CHPSVX),

   where U (or L) is a product of permutation and unit upper (lower) triangular matrices and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

2. If some D(i,i)=0, so that D is exactly singular, then the routine returns with INFO = i. Otherwise, the factored form of A is used to estimate the condition number of the matrix A. If the reciprocal of the condition number is less than machine precision, INFO = N+1 is returned as a warning, but the routine still goes on to solve for X and compute error bounds as described below.

3. The system of equations is solved for X using the factored form of A.

4. Iterative refinement is applied to improve the computed solution matrix and calculate error bounds and backward error estimates for it.

Arguments

FACT (input) CHARACTER*1
Specifies whether or not the factored form of A has been supplied on entry.
= 'F': On entry, AFP and IPIV contain the factored form of A. AP, AFP and IPIV will not be modified.
= 'N': The matrix A will be copied to AFP and factored.

UPLO (input) CHARACTER*1
= 'U': Upper triangle of A is stored;
= 'L': Lower triangle of A is stored.

N (input) INTEGER
The number of linear equations, i.e., the order of the matrix A. $N \geq 0$.

NRHS (input) INTEGER
The number of right hand sides, i.e., the number of columns of the matrices B and X. $NRHS \geq 0$.

AP (input) REAL/COMPLEX/COMPLEX array, dimension (N*(N+1)/2)
The upper or lower triangle of the symmetric/symmetric/Hermitian matrix A, packed columnwise in a linear array. The jth column of A is stored in the array AP as follows:
if UPLO = 'U', AP(i + (j−1)*j/2) = A(i,j) for $1 \leq i \leq j$;
if UPLO = 'L', AP(i + (j−1)*(2*n−j)/2) = A(i,j) for $j \leq i \leq n$.

AFP (input or output) REAL/COMPLEX/COMPLEX array, dimension (N*(N+1)/2)
If FACT = 'F', then AFP is an input argument and on entry contains the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization $A = U*D*U^T$ or $A = L*D*L^T$ as computed by SSPTRF/CSPTRF or the factorization $A = U*D*U^H$ or $A = L*D*L^H$ as computed by CHPTRF, stored as a packed triangular matrix in the same storage format as A.
If FACT = 'N', then AFP is an output argument and on exit returns the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization.

IPIV (input or output) INTEGER array, dimension (N)
If FACT = 'F', then IPIV is an input argument and on entry contains details of the interchanges and the block structure of D, as determined by SSPTRF/CSPTRF/CHPTRF.

If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.
If UPLO = 'U' and IPIV(k) = IPIV(k−1) < 0, then rows and columns k−1 and −IPIV(k) were interchanged and D(k−1:k,k−1:k) is a 2-by-2 diagonal block. If UPLO = 'L' and IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and −IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.
If FACT = 'N', then IPIV is an output argument and on exit contains details of the interchanges and the block structure of D, as determined by SSPTRF/CSPTRF/CHPTRF.

B (input) REAL/COMPLEX/COMPLEX array, dimension (LDB,NRHS)
The n-by-nrhs right hand side matrix B.

LDB (input) INTEGER
The leading dimension of the array B. LDB ≥ max(1,N).

X (output)
REAL/COMPLEX/COMPLEX array, dimension (LDX,NRHS)
If INFO = 0 or INFO = N+1, the N-by-NRHS solution matrix X.

LDX (input) INTEGER
The leading dimension of the array X. LDX ≥ max(1,N).

RCOND (output) REAL
The estimate of the reciprocal condition number of the matrix A. If RCOND is less than the machine precision (in particular, if RCOND = 0), the matrix is singular to working precision. This condition is indicated by a return code of INFO > 0.

FERR (output) REAL array, dimension (NRHS)
The estimated forward error bound for each solution vector X(j) (the jth column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) − XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR (output) REAL array, dimension (NRHS)
The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK *SSPSVX* (workspace) REAL array, dimension (3*N)
CSPSVX/CHPSVX (workspace) COMPLEX array, dimension (2*N)

IWORK *SSPSVX only* (workspace) INTEGER array, dimension (N)

RWORK *CSPSVX/CHPSVX only* (workspace) REAL array, dimension (N)

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0 and ≤ N: if INFO = i, D(i,i) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned.
= N+1: D is nonsingular, but RCOND is less than machine precision, meaning that the matrix is singular to working precision. Nevertheless, the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the value of RCOND would suggest.

SSPTRD/CHPTRD

```
SUBROUTINE SSPTRD( UPLO, N, AP, D, E, TAU, INFO )
CHARACTER          UPLO
INTEGER            INFO, N
REAL               AP( * ), D( * ), E( * ), TAU( * )

SUBROUTINE CHPTRD( UPLO, N, AP, D, E, TAU, INFO )
CHARACTER          UPLO
INTEGER            INFO, N
REAL               D( * ), E( * )
COMPLEX            AP( * ), TAU( * )
```

Purpose

SSPTRD/CHPTRD reduces a real/complex symmetric/Hermitian matrix A stored in packed form to real symmetric tridiagonal form T by an orthogonal/unitary similarity transformation: $Q^H*A*Q = T$.

Arguments

UPLO (input) CHARACTER*1
= 'U': Upper triangle of A is stored;
= 'L': Lower triangle of A is stored.

N (input) INTEGER
The order of the matrix A. N ≥ 0.

AP (input/output) REAL/COMPLEX array, dimension (N*(N+1)/2)
On entry, the upper or lower triangle of the symmetric/Hermitian matrix A, packed columnwise in a linear array. The jth column of A is stored in the array AP as follows:
if UPLO = 'U', AP(i + (j−1)*j/2) = A(i,j) for 1≤i≤j;
if UPLO = 'L', AP(i + (j−1)*(2*n−j)/2) = A(i,j) for j≤i≤n.
On exit, if UPLO = 'U', the diagonal and first superdiagonal of A are overwritten by the corresponding elements of the tridiagonal matrix T, and the elements above the first superdiagonal, with the array TAU, represent the orthogonal/unitary matrix Q as a product of elementary reflectors; if UPLO = 'L', the diagonal and first subdiagonal of A are

overwritten by the corresponding elements of the tridiagonal matrix T, and the elements below the first subdiagonal, with the array TAU, represent the orthogonal/unitary matrix Q as a product of elementary reflectors.

D (output) REAL array, dimension (N)
The diagonal elements of the tridiagonal matrix T: D(i) = A(i,i).

E (output) REAL array, dimension (N−1)
The off-diagonal elements of the tridiagonal matrix T:
E(i) = A(i,i+1) if UPLO = 'U'; E(i) = A(i+1,i) if UPLO = 'L'.

TAU (output) REAL/COMPLEX array, dimension (N−1)
The scalar factors of the elementary reflectors.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SSPTRF/CSPTRF/CHPTRF

```
SUBROUTINE SSPTRF( UPLO, N, AP, IPIV, INFO )
CHARACTER          UPLO
INTEGER            INFO, N
INTEGER            IPIV( * )
REAL               AP( * )

SUBROUTINE CSPTRF( UPLO, N, AP, IPIV, INFO )
CHARACTER          UPLO
INTEGER            INFO, N
INTEGER            IPIV( * )
COMPLEX            AP( * )

SUBROUTINE CHPTRF( UPLO, N, AP, IPIV, INFO )
CHARACTER          UPLO
INTEGER            INFO, N
INTEGER            IPIV( * )
COMPLEX            AP( * )
```

Purpose

SSPTRF/CSPTRF/CHPTRF computes the factorization of a real/complex/complex symmetric/symmetric/Hermitian matrix A stored in packed format, using the diagonal pivoting method. The form of the factorization is

$A = U*D*U^T$ or $A = L*D*L^T$ (SSPTRF/CSPTRF), or

$A = U*D*U^H$ or $A = L*D*L^H$ (CHPTRF),

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and D is symmetric (SSPTRF/CSPTRF) or Hermitian (CHPTRF) and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

Arguments

UPLO (input) CHARACTER*1
= 'U': Upper triangle of A is stored;
= 'L': Lower triangle of A is stored.

N (input) INTEGER
The order of the matrix A. N ≥ 0.

AP (input/output)
REAL/COMPLEX/COMPLEX array, dimension (N*(N+1)/2)
On entry, the upper or lower triangle of the symmetric/svmmetric/Hermitian matrix A, packed columnwise in a linear array. The jth column of A is stored in the array AP as follows:
if UPLO = 'U', AP(i + (j−1)*j/2) = A(i,j) for 1≤i≤j;
if UPLO = 'L', AP(i + (j−1)*(2*n−j)/2) = A(i,j) for j≤i≤n.
On exit, the block diagonal matrix D and the multipliers used to obtain the factor U or L, stored as a packed triangular matrix overwriting A.

IPIV (output) INTEGER array, dimension (N)
Details of the interchanges and the block structure of D. If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block. If UPLO = 'U' and IPIV(k) = IPIV(k−1) < 0, then rows and columns k−1 and −IPIV(k) were interchanged and D(k−1:k,k−1:k) is a 2-by-2 diagonal block. If UPLO = 'L' and IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and −IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, D(i,i) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

SSPTRI/CSPTRI/CHPTRI

```
SUBROUTINE SSPTRI( UPLO, N, AP, IPIV, WORK, INFO )
CHARACTER          UPLO
INTEGER            INFO, N
INTEGER            IPIV( * )
REAL               AP( * ), WORK( * )
```

```
SUBROUTINE CSPTRI( UPLO, N, AP, IPIV, WORK, INFO )
CHARACTER          UPLO
INTEGER            INFO, N
INTEGER            IPIV( * )
COMPLEX            AP( * ), WORK( * )

SUBROUTINE CHPTRI( UPLO, N, AP, IPIV, WORK, INFO )
CHARACTER          UPLO
INTEGER            INFO, N
INTEGER            IPIV( * )
COMPLEX            AP( * ), WORK( * )
```

Purpose

SSPTRI/CSPTRI/CHPTRI computes the inverse of a real/complex/complex symmetric/symmetric/Hermitian indefinite matrix A in packed storage using the factorization $A = U*D*U^T$ or $A = L*D*L^T$ computed by SSPTRF/CSPTRF or the factorization $A = U*D*U^H$ or $A = L*D*L^H$ computed by CHPTRF.

Arguments

UPLO (input) CHARACTER*1
Specifies whether the details of the factorization are stored as an upper or lower triangular matrix.
= 'U': Upper triangular, form is $A = U*D*U^T$ (SSPTRI/CSPTRI) or $A = U*D*U^H$ (CHPTRI);
= 'L': Lower triangular, form is $A = L*D*L^T$ (SSPTRI/CSPTRI) or $A = L*D*L^H$ (CHPTRI).

N (input) INTEGER
The order of the matrix A. $N \geq 0$.

AP (input/output)
REAL/COMPLEX/COMPLEX array, dimension (N*(N+1)/2)
On entry, the block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by SSPTRF/CSPTRF/CHPTRF, stored as a packed triangular matrix.
On exit, if INFO = 0, the (symmetric/symmetric/Hermitian) inverse of the original matrix, stored as a packed triangular matrix. The jth column of A^{-1} is stored in the array AP as follows:
if UPLO = 'U', $AP(i + (j-1)*j/2) = A^{-1}(i,j)$ for $1 \leq i \leq j$;
if UPLO = 'L', $AP(i + (j-1)*(2*n-j)/2) = A^{-1}(i,j)$ for $j \leq i \leq n$.

IPIV (input) INTEGER array, dimension (N)
Details of the interchanges and the block structure of D as determined by SSPTRF/CSPTRF/CHPTRF.

WORK (workspace) REAL/COMPLEX/COMPLEX array, dimension (N)

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, D(i,i) = 0; the matrix is singular and its inverse could not be computed.

SSPTRS/CSPTRS/CHPTRS

```
SUBROUTINE SSPTRS( UPLO, N, NRHS, AP, IPIV, B, LDB, INFO )
CHARACTER          UPLO
INTEGER            INFO, LDB, N, NRHS
INTEGER            IPIV( * )
REAL               AP( * ), B( LDB, * )

SUBROUTINE CSPTRS( UPLO, N, NRHS, AP, IPIV, B, LDB, INFO )
CHARACTER          UPLO
INTEGER            INFO, LDB, N, NRHS
INTEGER            IPIV( * )
COMPLEX            AP( * ), B( LDB, * )

SUBROUTINE CHPTRS( UPLO, N, NRHS, AP, IPIV, B, LDB, INFO )
CHARACTER          UPLO
INTEGER            INFO, LDB, N, NRHS
INTEGER            IPIV( * )
COMPLEX            AP( * ), B( LDB, * )
```

Purpose

SSPTRS/CSPTRS/CHPTRS solves a system of linear equations $A*X = B$ with a real/complex/complex symmetric/symmetric/Hermitian matrix A stored in packed format using the factorization $A = U*D*U^T$ or $A = L*D*L^T$ computed by SSPTRF/CSPTRF or the factorization $A = U*D*U^H$ or $A = L*D*L^H$ computed by CHPTRF.

Arguments

UPLO (input) CHARACTER*1
Specifies whether the details of the factorization are stored as an upper or lower triangular matrix.
= 'U': Upper triangular, form is $A = U*D*U^T$ (SSPTRS/CSPTRS) or $A = U*D*U^H$ (CHPTRS);
= 'L': Lower triangular, form is $A = L*D*L^T$ (SSPTRS/CSPTRS) or $A = L*D*L^H$ (CHPTRS).

N (input) INTEGER
The order of the matrix A. $N \geq 0$.

NRHS (input) INTEGER
The number of right hand sides, i.e., the number of columns of the matrix B. $NRHS \geq 0$.

AP (input) REAL/COMPLEX/COMPLEX array, dimension (N*(N+1)/2)
The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by SSPTRF/CSPTRF/CHPTRF, stored as a packed triangular matrix.

IPIV (input) INTEGER array, dimension (N)
Details of the interchanges and the block structure of D as determined by SSPTRF/CSPTRF/CHPTRF.

B (input/output)
REAL/COMPLEX/COMPLEX array, dimension (LDB,NRHS)
On entry, the right hand side matrix B.
On exit, the solution matrix X.

LDB (input) INTEGER
The leading dimension of the array B. LDB $\geq$ max(1,N).

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SSTEBZ

```
      SUBROUTINE SSTEBZ( RANGE, ORDER, N, VL, VU, IL, IU, ABSTOL, D, E,
     $                   M, NSPLIT, W, IBLOCK, ISPLIT, WORK, IWORK,
     $                   INFO )
      CHARACTER          ORDER, RANGE
      INTEGER            IL, INFO, IU, M, N, NSPLIT
      REAL               ABSTOL, VL, VU
      INTEGER            IBLOCK( * ), ISPLIT( * ), IWORK( * )
      REAL               D( * ), E( * ), W( * ), WORK( * )
```

Purpose

SSTEBZ computes the eigenvalues of a symmetric tridiagonal matrix T. The user may ask for all eigenvalues, all eigenvalues in the half-open interval (VL, VU], or the ILth through IU^{th} eigenvalues.

To avoid overflow, the matrix must be scaled so that its largest element is no greater than $overflow^{1/2} * underflow^{1/4}$ in absolute value, and for greater accuracy, it should not be much smaller than that.

Arguments

RANGE (input) CHARACTER*1
= 'A': ("All") all eigenvalues will be found.
= 'V': ("Value") all eigenvalues in the half-open interval (VL, VU] will be found.
= 'I': ("Index") the ILth through IU^{th} eigenvalues will be found.

ORDER (input) CHARACTER*1
= 'B': ("By Block") the eigenvalues will be grouped by split-off block (see IBLOCK, ISPLIT) and ordered from smallest to largest within the block.
= 'E': ("Entire matrix") the eigenvalues for the entire matrix will be ordered from smallest to largest.

N (input) INTEGER
The order of the tridiagonal matrix T. N $\geq$ 0.

VL, VU (input) REAL
If RANGE='V', the lower and upper bounds of the interval to be searched for eigenvalues. Eigenvalues less than or equal to VL, or greater than VU, will not be returned. VL < VU.
Not referenced if RANGE = 'A' or 'I'.

IL, IU (input) INTEGER
If RANGE='I', the indices (in ascending order) of the smallest and largest eigenvalues to be returned.
$1 \leq IL \leq IU \leq N$, if N > 0; IL = 1 and IU = 0 if N = 0.
Not referenced if RANGE = 'A' or 'V'.

ABSTOL (input) REAL
The absolute tolerance for the eigenvalues. An eigenvalue (or cluster) is considered to be located if it has been determined to lie in an interval whose width is ABSTOL or less. If ABSTOL is less than or equal to zero, then $EPS*||T||_1$ will be used in its place, where EPS is the machine precision.
Eigenvalues will be computed most accurately when ABSTOL is set to twice the underflow threshold 2*SLAMCH('S'), not zero.

D (input) REAL array, dimension (N)
The n diagonal elements of the tridiagonal matrix T.

E (input) REAL array, dimension (N−1)
The (n−1) off-diagonal elements of the tridiagonal matrix T.

M (output) INTEGER
The actual number of eigenvalues found ($0 \leq M \leq N$). (See also the description of INFO = 2,3.)

NSPLIT (output) INTEGER
The number of diagonal blocks in the matrix T. ($1 \leq NSPLIT \leq N$).

W (output) REAL array, dimension (N)
On exit, the first M elements of W will contain the eigenvalues. (SSTEBZ may use the remaining N−M elements as workspace.)

IBLOCK (output) INTEGER array, dimension (N)
At each row/column j where E(j) is zero or small, the matrix T is considered to split into a block diagonal matrix. On exit, if INFO = 0, IBLOCK(i) specifies to which block (from 1 to NSPLIT) the eigenvalue W(i) belongs. If INFO > 0, IBLOCK(i) is set to a negative block number if the ith eigenvalue did not converge. (SSTEBZ may use the remaining N−M elements as workspace.)

ISPLIT (output) INTEGER array, dimension (N)
The splitting points, at which T breaks up into submatrices. The

first submatrix consists of rows/columns 1 to ISPLIT(1), the second of rows/columns ISPLIT(1)+1 through ISPLIT(2), etc., and the NSPLITth consists of rows/columns ISPLIT(NSPLIT−1)+1 through ISPLIT(NSPLIT)=N. (Only the first NSPLIT elements will actually be used, but since the user cannot know a priori what value NSPLIT will have, N words must be reserved for ISPLIT.)

WORK (workspace) REAL array, dimension (4*N)

IWORK (workspace) INTEGER array, dimension (3*N)

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: some or all of the eigenvalues failed to converge or were not computed:
= 1 or 3: Bisection failed to converge for some eigenvalues; these eigenvalues are flagged by a negative block number. The effect is that the eigenvalues may not be as accurate as the absolute and relative tolerances. This is generally caused by unexpectedly inaccurate arithmetic.
= 2 or 3: RANGE='I' only: Not all of the eigenvalues IL:IU were found.
Effect: M < IU+1−IL.
Cause: non-monotonic arithmetic, causing the Sturm sequence to be non-monotonic.
Cure: recalculate, using RANGE='A', and pick out eigenvalues IL:IU. In some cases, increasing the internal parameter FUDGE may make things work.
= 4: RANGE='I', and the Gershgorin interval initially used was too small. No eigenvalues were computed.
Probable cause: your machine has sloppy floating-point arithmetic.
Cure: Increase the internal parameter FUDGE, recompile, and try again.

SSTEDC/CSTEDC

```
      SUBROUTINE SSTEDC( COMPZ, N, D, E, Z, LDZ, WORK, LWORK, IWORK,
     $                   LIWORK, INFO )
      CHARACTER          COMPZ
      INTEGER            INFO, LDZ, LIWORK, LWORK, N
      INTEGER            IWORK( * )
      REAL               D( * ), E( * ), WORK( * ), Z( LDZ, * )
```

```
      SUBROUTINE CSTEDC( COMPZ, N, D, E, Z, LDZ, WORK, LWORK, RWORK,
     $                   LRWORK, IWORK, LIWORK, INFO )
      CHARACTER          COMPZ
      INTEGER            INFO, LDZ, LIWORK, LRWORK, LWORK, N
      INTEGER            IWORK( * )
      REAL               D( * ), E( * ), RWORK( * )
      COMPLEX            WORK( * ), Z( LDZ, * )
```

Purpose

SSTEDC/CSTEDC computes all eigenvalues and, optionally, eigenvectors of a symmetric tridiagonal matrix using the divide and conquer method.

The eigenvectors of a full or band real/complex symmetric/Hermitian matrix can also be found if SSYTRD/CHETRD or SSPTRD/CHPTRD or SSBTRD/CHBTRD has been used to reduce this matrix to tridiagonal form.

This code makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Arguments

COMPZ (input) CHARACTER*1
= 'N': Compute eigenvalues only.
= 'I': Compute eigenvectors of tridiagonal matrix also.
= 'V': Compute eigenvectors of original symmetric/Hermitian matrix also. On entry, Z contains the orthogonal/unitary matrix used to reduce the original matrix to tridiagonal form.

N (input) INTEGER
The dimension of the symmetric tridiagonal matrix. N ≥ 0.

D (input/output) REAL array, dimension (N)
On entry, the diagonal elements of the tridiagonal matrix.
On exit, if INFO = 0, the eigenvalues in ascending order.

E (input/output) REAL array, dimension (N−1)
On entry, the subdiagonal elements of the tridiagonal matrix.
On exit, E has been destroyed.

Z (input/output) REAL/COMPLEX array, dimension (LDZ,N)
On entry, if COMPZ = 'V', then Z contains the orthogonal/unitary matrix used in the reduction to tridiagonal form.
On exit, if INFO = 0, then if COMPZ = 'V', Z contains the orthonormal eigenvectors of the original symmetric/Hermitian matrix, and if COMPZ = 'I', Z contains the orthonormal eigenvectors of the symmetric tridiagonal matrix.
If COMPZ = 'N', then Z is not referenced.

LDZ (input) INTEGER
The leading dimension of the array Z. LDZ ≥ 1.

If eigenvectors are desired, then LDZ $\geq$ max(1,N).

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK.
SSTEDC
If COMPZ = 'N' or N $\leq$ 1 then LWORK $\geq$ 1.
If COMPZ = 'V' and N > 1 then LWORK $\geq$ (1 + 3*N + 2*N*lg N + 3*N^2), where lg N = smallest integer k such that $2^k \geq$N. If COMPZ = 'I' and N > 1 then LWORK $\geq$ (1 + 3*N + 2*N*lg N + 2*N^2).
CSTEDC
If COMPZ = 'N' or 'I', or N $\leq$ 1, LWORK $\geq$ 1.
If COMPZ = 'V' and N > 1, LWORK $\geq$ N*N.

If LWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK *CSTEDC only* (workspace/output) REAL array, dimension (LRWORK) On exit, if INFO = 0, RWORK(1) returns the optimal LRWORK.

LRWORK *CSTEDC only* (input) INTEGER
The dimension of the array RWORK.
If COMPZ = 'N' or N $\leq$ 1, LRWORK $\geq$ 1.
If COMPZ = 'V' and N > 1, LRWORK $\geq$ (1 + 3*N + 2*N*lg N + 3*N^2), where lg N = smallest integer k such that $2^k \geq$N.
If COMPZ = 'I' and N > 1, LRWORK $\geq$ (1 + 3*N + 2*N*lg N + 3*N^2).

If LRWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the RWORK array, returns this value as the first entry of the RWORK array, and no error message related to LRWORK is issued by XERBLA.

IWORK (workspace/output) INTEGER array, dimension (LIWORK)
On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK (input) INTEGER
The dimension of the array IWORK.
If COMPZ = 'N' or N $\leq$ 1 then LIWORK $\geq$ 1.
If COMPZ = 'V' and N > 1 then LIWORK $\geq$ (6 + 6*N + 5*N*lg N).
If COMPZ = 'I' and N > 1 then LIWORK $\geq$ 3 + 5*N.

If LIWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the IWORK array, returns this value as the first entry of the IWORK array, and no error message related to LIWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: The algorithm failed to compute an eigenvalue while working on the submatrix lying in rows and columns INFO/(N+1) through mod(INFO,N+1)

SSTEGR/CSTEGR

```
 SUBROUTINE SSTEGR( JOBZ, RANGE, N, D, E, VL, VU, IL, IU, ABSTOL,
$                   M, W, Z, LDZ, ISUPPZ, WORK, LWORK, IWORK,
$                   LIWORK, INFO )
 CHARACTER          JOBZ, RANGE
 INTEGER            IL, INFO, IU, LDZ, LWORK, LIWORK, M, N
 REAL               ABSTOL, VL, VU
 INTEGER            ISUPPZ( * ), IWORK( * )
 REAL               D( * ), E( * ), W( * ), WORK( * ), Z( LDZ, * )

 SUBROUTINE CSTEGR( JOBZ, RANGE, N, D, E, VL, VU, IL, IU, ABSTOL,
$                   M, W, Z, LDZ, ISUPPZ, WORK, LWORK, IWORK,
$                   LIWORK, INFO )
 CHARACTER          JOBZ, RANGE
 INTEGER            IL, INFO, IU, LDZ, LWORK, LIWORK, M, N
 REAL               ABSTOL, VL, VU
 INTEGER            ISUPPZ( * ), IWORK( * )
 REAL               D( * ), E( * ), W( * ), WORK( * )
 COMPLEX            Z( LDZ, * )
```

Purpose

SSTEGR/CSTEGR computes selected eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix T. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues. The eigenvalues are computed by the dqds algorithm, while orthogonal eigenvectors are computed from various "good" LDL^T representations (also known as Relatively Robust Representations). Gram-Schmidt orthogonalization is avoided as far as possible. More specifically, the various steps of the algorithm are as follows. For the ith unreduced block of T,

(a) Compute $T - \sigma_i = L_i D_i L_i^T$, such that $L_i D_i L_i^T$ is a relatively robust representation,

(b) Compute the eigenvalues, λ_j, of $L_i D_i L_i^T$ to high relative accuracy by the dqds algorithm,

(c) If there is a cluster of close eigenvalues, "choose" σ_i close to the cluster, and go to step (a),

(d) Given the approximate eigenvalue λ_j of $L_i D_i L_i^T$, compute the corresponding eigenvector by forming a rank-revealing twisted factorization.

The desired accuracy of the output can be specified by the input parameter ABSTOL.

Note 1: Currently SSTEGR/CSTEGR is only set up to find ALL the n eigenvalues and eigenvectors of T in $O(n^2)$ time.

Note 2 : Currently the routine SSTEIN/CSTEIN is called when an appropriate σ_i cannot be chosen in step (c) above. SSTEIN/CSTEIN invokes modified Gram-Schmidt when eigenvalues are close.

Note 3 : SSTEGR/CSTEGR works only on machines which follow ieee-754 floating-point standard in their handling of infinities and NaNs. Normal execution of SSTEGR/CSTEGR may create NaNs and infinities and hence may abort due to a floating point exception in environments which do not conform to the ieee standard.

Arguments

JOBZ (input) CHARACTER*1
= 'N': Compute eigenvalues only;
= 'V': Compute eigenvalues and eigenvectors.

RANGE (input) CHARACTER*1
= 'A': all eigenvalues will be found;
= 'V': all eigenvalues in the half-open interval (VL,VU] will be found;
= 'I': the ILth through IU^{th} eigenvalues will be found.
Only RANGE = 'A' is currently supported.

N (input) INTEGER
The order of the matrix A. N $\geq$ 0.

D (input/output) REAL array, dimension (N)
On entry, the n diagonal elements of the tridiagonal matrix T.
On exit, D is overwritten.

E (input/output) REAL array, dimension (N)
On entry, the (n−1) subdiagonal elements of the tridiagonal matrix T in elements 1 to N−1 of E; E(N) need not be set.
On exit, E is overwritten.

VL,VR (input) REAL
If RANGE='V', the lower and upper bounds of the interval to be searched for eigenvalues. VL < VU.
Not referenced if RANGE = 'A' or 'I'.

IL,IU (input) INTEGER
If RANGE='I', the indices (in ascending order) of the smallest and largest eigenvalues to be returned. 1 $\leq$ IL $\leq$ IU $\leq$ N, if N > 0; IL = 1 and IU = 0 if N = 0.
Not referenced if RANGE = 'A' or 'V'.

ABSTOL (input) REAL
The absolute error tolerance for the eigenvalues/eigenvectors.
IF JOBZ = 'V', the eigenvalues and eigenvectors output have residual norms bounded by ABSTOL, and the dot products between different eigenvectors are bounded by ABSTOL. If ABSTOL < N*EPS*$\|T\|_1$, then N*EPS*$\|T\|_1$ will be used in its place, where EPS is the machine precision. The eigenvalues are computed to an accuracy of EPS*$\|T\|_1$ irrespective of ABSTOL. If high relative accuracy is important, set ABSTOL to xLAMCH('S').

M (output) INTEGER
The total number of eigenvalues found. 0 $\leq$ M $\leq$ N.
If RANGE = 'A', M = N, and if RANGE = 'I', M = IU−IL+1.

W (output) REAL array, dimension (N)
The first M elements contain the selected eigenvalues in ascending order.

Z (output) REAL/COMPLEX array, dimension (LDZ, max(1,M)) .
If JOBZ = 'V', then if INFO = 0, the first M columns of Z contain the orthonormal eigenvectors of the matrix T corresponding to the selected eigenvalues, with the ith column of Z holding the eigenvector associated with W(i).
If JOBZ = 'N', then Z is not referenced.
Note: the user must ensure that at least max(1,M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and an upper bound must be used.

LDZ (input) INTEGER
The leading dimension of the array Z. LDZ $\geq$ 1, and if JOBZ = 'V', LDZ $\geq$ max(1,N).

ISUPPZ (output) INTEGER array, dimension (2*max(1,M))
The support of the eigenvectors in Z, i.e., the indices indicating the nonzero elements in Z. The ith eigenvector is nonzero only in elements ISUPPZ(2*i−1) through ISUPPZ(2*i).

WORK (workspace/output) REAL array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal (and minimal) LWORK.

LWORK (input) INTEGER
The dimension of the array WORK. LWORK $\geq$ max(1,18*N).

If LWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

IWORK (workspace/output) INTEGER array, dimension (LIWORK)
On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK (input) INTEGER
The dimension of the array IWORK. LIWORK $\geq$ max(1, 10*N).

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the IWORK array, returns this value as the first entry of the IWORK array, and no error message related to LIWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = $-$i, the ith argument had an illegal value.
> 0: if INFO = 1, internal error in SLARRE
if INFO = 2, internal error in SLARRV/CLARRV

SSTEIN/CSTEIN

```
      SUBROUTINE SSTEIN( N, D, E, M, W, IBLOCK, ISPLIT, Z, LDZ, WORK,
     $                   IWORK, IFAIL, INFO )
      INTEGER            INFO, LDZ, M, N
      INTEGER            IBLOCK( * ), IFAIL( * ), ISPLIT( * ),
     $                   IWORK( * )
      REAL               D( * ), E( * ), W( * ), WORK( * ), Z( LDZ, * )

      SUBROUTINE CSTEIN( N, D, E, M, W, IBLOCK, ISPLIT, Z, LDZ, WORK,
     $                   IWORK, IFAIL, INFO )
      INTEGER            INFO, LDZ, M, N
      INTEGER            IBLOCK( * ), IFAIL( * ), ISPLIT( * ),
     $                   IWORK( * )
      REAL               D( * ), E( * ), W( * ), WORK( * )
      COMPLEX            Z( LDZ, * )
```

Purpose

SSTEIN/CSTEIN computes the eigenvectors of a real symmetric tridiagonal matrix T corresponding to specified eigenvalues, using inverse iteration.

The maximum number of iterations allowed for each eigenvector is specified by an internal parameter MAXITS (currently set to 5).

CSTEIN only
Although the eigenvectors are real, they are stored in a complex array, which may be passed to CUNMTR or CUPMTR for back transformation to the eigenvectors of a complex Hermitian matrix which was reduced to tridiagonal form.

Arguments

N (input) INTEGER
The order of the matrix. N $\geq$ 0.

D (input) REAL array, dimension (N)
The n diagonal elements of the tridiagonal matrix T.

E (input) REAL array, dimension (N)
The (n$-$1) subdiagonal elements of the tridiagonal matrix T, stored in elements 1 to n$-$1; E(n) need not be set.

M (input) INTEGER
The number of eigenvectors to be found. 0 $\leq$ M $\leq$ N.

W (input) REAL array, dimension (N)
The first M elements of W contain the eigenvalues for which eigenvectors are to be computed. The eigenvalues should be grouped by split-off block and ordered from smallest to largest within the block. (The output array W from SSTEBZ with ORDER = 'B' is expected here.)

IBLOCK (input) INTEGER array, dimension (N)
The submatrix indices associated with the corresponding eigenvalues in W; IBLOCK(i)=1 if eigenvalue W(i) belongs to the first submatrix from the top, =2 if W(i) belongs to the second submatrix, etc. (The output array IBLOCK from SSTEBZ is expected here.)

ISPLIT (input) INTEGER array, dimension (N)
The splitting points, at which T breaks up into submatrices. The first submatrix consists of rows/columns 1 to ISPLIT(1), the second of rows/columns ISPLIT(1)+1 through ISPLIT(2), etc. (The output array ISPLIT from SSTEBZ is expected here.)

Z (output) REAL/COMPLEX array, dimension (LDZ, M)
The computed eigenvectors. The eigenvector associated with the eigenvalue W(i) is stored in the ith column of Z. Any vector which fails to converge is set to its current iterate after MAXITS iterations.
CSTEIN only
The imaginary parts of the eigenvectors are set to zero.

LDZ (input) INTEGER
The leading dimension of the array Z. LDZ $\geq$ max(1,N).

WORK (workspace) REAL array, dimension (5*N)

IWORK (workspace) INTEGER array, dimension (N)

IFAIL (output) INTEGER array, dimension (M)
On normal exit, all elements of IFAIL are zero. If one or more eigenvectors fail to converge after MAXITS iterations, then their indices are stored in array IFAIL.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = $-$i, the ith argument had an illegal value.

> 0: if INFO = i, then i eigenvectors failed to converge in MAXITS iterations. Their indices are stored in array IFAIL.

SSTEQR/CSTEQR

```
SUBROUTINE SSTEQR( COMPZ, N, D, E, Z, LDZ, WORK, INFO )
CHARACTER          COMPZ
INTEGER            INFO, LDZ, N
REAL               D( * ), E( * ), WORK( * ), Z( LDZ, * )

SUBROUTINE CSTEQR( COMPZ, N, D, E, Z, LDZ, WORK, INFO )
CHARACTER          COMPZ
INTEGER            INFO, LDZ, N
REAL               D( * ), E( * ), WORK( * )
COMPLEX            Z( LDZ, * )
```

Purpose

SSTEQR/CSTEQR computes all eigenvalues and, optionally, eigenvectors of a symmetric tridiagonal matrix using the implicit QL or QR method. The eigenvectors of a full or band symmetric or Hermitian matrix can also be found if SSYTRD/CHETRD, SSPTRD/CHPTRD, or SSBTRD/CHBTRD has been used to reduce this matrix to tridiagonal form.

Arguments

COMPZ (input) CHARACTER*1
= 'N': Compute eigenvalues only.
= 'V': Compute eigenvalues and eigenvectors of the original symmetric/Hermitian matrix. On entry, Z must contain the orthogonal/unitary matrix used to reduce the original matrix to tridiagonal form.
= 'I': Compute eigenvalues and eigenvectors of the tridiagonal matrix. Z is initialized to the identity matrix.

N (input) INTEGER
The order of the matrix. $N \geq 0$.

D (input/output) REAL array, dimension (N)
On entry, the diagonal elements of the tridiagonal matrix.
On exit, if INFO = 0, the eigenvalues in ascending order.

E (input/output) REAL array, dimension (N−1)
On entry, the (n−1) subdiagonal elements of the tridiagonal matrix.
On exit, E has been destroyed.

Z (input/output) REAL/COMPLEX array, dimension (LDZ, N)
On entry, if COMPZ = 'V', then Z contains the orthogonal/unitary matrix used in the reduction to tridiagonal form.
On exit, if INFO = 0, then if COMPZ = 'V', Z contains the orthonormal eigenvectors of the original symmetric/Hermitian matrix, and if COMPZ = 'I', Z contains the orthonormal eigenvectors of the symmetric tridiagonal matrix.
If COMPZ = 'N', then Z is not referenced.

LDZ (input) INTEGER
The leading dimension of the array Z. $LDZ \geq 1$, and if eigenvectors are desired, then $LDZ \geq \max(1,N)$.

WORK (workspace) REAL array, dimension (max(1,2*N−2))
If COMPZ = 'N', then WORK is not referenced.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: the algorithm has failed to find all the eigenvalues in a total of 30*N iterations; if INFO = i, then i elements of E have not converged to zero; on exit, D and E contain the elements of a symmetric tridiagonal matrix which is orthogonally/unitarily similar to the original matrix.

SSTERF

```
SUBROUTINE SSTERF( N, D, E, INFO )
INTEGER            INFO, N
REAL               D( * ), E( * )
```

Purpose

SSTERF computes all eigenvalues of a symmetric tridiagonal matrix using the Pal-Walker-Kahan variant of the QL or QR algorithm.

Arguments

N (input) INTEGER
The order of the matrix. $N \geq 0$.

D (input/output) REAL array, dimension (N)
On entry, the n diagonal elements of the tridiagonal matrix.
On exit, if INFO = 0, D contains the eigenvalues in ascending order.

E (input/output) REAL array, dimension (N−1)
On entry, the (n−1) subdiagonal elements of the tridiagonal matrix.
On exit, E has been destroyed.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

> 0: the algorithm has failed to find all the eigenvalues in a total of 30*N iterations; if INFO = i, then i elements of E have not converged to zero.

SSTEV

```
SUBROUTINE SSTEV( JOBZ, N, D, E, Z, LDZ, WORK, INFO )
CHARACTER          JOBZ
INTEGER            INFO, LDZ, N
REAL               D( * ), E( * ), WORK( * ), Z( LDZ, * )
```

Purpose

SSTEV computes all eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix A.

Arguments

JOBZ (input) CHARACTER*1
= 'N': Compute eigenvalues only;
= 'V': Compute eigenvalues and eigenvectors.

N (input) INTEGER
The order of the matrix A. $N \geq 0$.

D (input/output) REAL array, dimension (N)
On entry, the n diagonal elements of the tridiagonal matrix A.
On exit, if INFO = 0, the eigenvalues in ascending order.

E (input/output) REAL array, dimension (N)
On entry, the (n−1) subdiagonal elements of the tridiagonal matrix A, stored in elements 1 to n−1 of E; E(n) need not be set, but is used by the routine.
On exit, the contents of E are destroyed.

Z (output) REAL array, dimension (LDZ,N)
If JOBZ = 'V', then if INFO = 0, Z contains the orthonormal eigenvectors of the matrix A, with the ith column of Z holding the eigenvector associated with the eigenvalue returned in D(i).
If JOBZ = 'N', then Z is not referenced.

LDZ (input) INTEGER
The leading dimension of the array Z. $LDZ \geq 1$, and if JOBZ = 'V', $LDZ \geq \max(1,N)$.

WORK (workspace) REAL array, dimension (max(1,2*N−2))
If JOBZ = 'N', WORK is not referenced.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, the algorithm failed to converge; i elements of E did not converge to zero.

SSTEVD

```
SUBROUTINE SSTEVD( JOBZ, N, D, E, Z, LDZ, WORK, LWORK, IWORK,
$                  LIWORK, INFO )
CHARACTER          JOBZ
INTEGER            INFO, LDZ, LIWORK, LWORK, N
INTEGER            IWORK( * )
REAL               D( * ), E( * ), WORK( * ), Z( LDZ, * )
```

Purpose

SSTEVD computes all eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix. If eigenvectors are desired, it uses a divide and conquer algorithm.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Arguments

JOBZ (input) CHARACTER*1
= 'N': Compute eigenvalues only;
= 'V': Compute eigenvalues and eigenvectors.

N (input) INTEGER
The order of the matrix. $N \geq 0$.

D (input/output) REAL array, dimension (N)
On entry, the n diagonal elements of the tridiagonal matrix A.
On exit, if INFO = 0, the eigenvalues in ascending order.

E (input/output) REAL array, dimension (N)
On entry, the (n−1) subdiagonal elements of the tridiagonal matrix A, stored in elements 1 to n−1 of E; E(n) need not be set, but is used by the routine.
On exit, the contents of E are destroyed.

Z (output) REAL array, dimension (LDZ, N)
If JOBZ = 'V', then if INFO = 0, Z contains the orthonormal eigenvectors of the matrix A, with the ith column of Z holding the eigenvector associated with D(i).
If JOBZ = 'N', then Z is not referenced.

LDZ (input) INTEGER
The leading dimension of the array Z. LDZ $\geq$ 1, and if JOBZ = 'V', LDZ $\geq$ max(1,N).

WORK (workspace/output) REAL array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK.
If JOBZ = 'N' or N $\leq$ 1 then LWORK $\geq$ 1.
If JOBZ = 'V' and N > 1 then LWORK $\geq$ (1 + 3*N + 2*N*lg N + 2*N^2), where lg N = smallest integer k such that $2^k \geq$ N).

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

IWORK (workspace/output) INTEGER array, dimension (LIWORK)
On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK (input) INTEGER
The dimension of the array IWORK.
If JOBZ = 'N' or N $\leq$ 1 then LIWORK $\geq$ 1. If JOBZ = 'V' and N > 1 then LIWORK $\geq$ 3+5*N.

If LIWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the IWORK array, returns this value as the first entry of the IWORK array, and no error message related to LIWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = $-$i, the ith argument had an illegal value.
> 0: if INFO = i, the algorithm failed to converge; i off-diagonal elements of E did not converge to zero.

SSTEVR

```
      SUBROUTINE SSTEVR( JOBZ, RANGE, N, D, E, VL, VU, IL, IU, ABSTOL,
     $                   M, W, Z, LDZ, ISUPPZ, WORK, LWORK, IWORK,
     $                   LIWORK, INFO )
      CHARACTER          JOBZ, RANGE
      INTEGER            IL, INFO, IU, LDZ, M, N
      REAL               ABSTOL, VL, VU
      INTEGER            ISUPPZ( * ), IWORK( * )
      REAL               D( * ), E( * ), W( * ), WORK( * ), Z( LDZ, * )
```

Purpose

SSTEVR computes selected eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix T. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

Whenever possible, SSTEVR calls SSTEGR to compute the eigenspectrum using Relatively Robust Representations. SSTEGR computes eigenvalues by the dqds algorithm, while orthogonal eigenvectors are computed from various "good" LDL^T representations (also known as Relatively Robust Representations). Gram-Schmidt orthogonalization is avoided as far as possible. More specifically, the various steps of the algorithm are as follows. For the ith unreduced block of T,

(a) Compute $T - \sigma_i = L_i D_i L_i^T$, such that $L_i D_i L_i^T$ is a relatively robust representation,

(b) Compute the eigenvalues, λ_j, of $L_i D_i L_i^T$ to high relative accuracy by the dqds algorithm,

(c) If there is a cluster of close eigenvalues, "choose" σ_i close to the cluster, and go to step (a),

(d) Given the approximate eigenvalue λ_j of $L_i D_i L_i^T$, compute the corresponding eigenvector by forming a rank-revealing twisted factorization.

The desired accuracy of the output can be specified by the input parameter ABSTOL.

Note 1: SSTEVR calls SSTEGR when the full spectrum is requested on machines which conform to the ieee-754 floating point standard. SSTEVR calls SSTEBZ and SSTEIN on non-ieee machines and when partial spectrum requests are made.

Normal execution of SSTEGR may create NaNs and infinities and hence may abort due to a floating point exception in environments which do not handle NaNs and infinities in the ieee standard default manner.

Arguments

JOBZ (input) CHARACTER*1
= 'N': Compute eigenvalues only;
= 'V': Compute eigenvalues and eigenvectors.

RANGE (input) CHARACTER*1
= 'A': all eigenvalues will be found;
= 'V': all eigenvalues in the half-open interval (VL,VU] will be found;
= 'I': the ILth through IUth eigenvalues will be found.

For RANGE = 'V' or 'I', SSTEBZ and SSTEIN are called.

N (input) INTEGER
The order of the matrix A. $N \geq 0$.

D (input/output) REAL array, dimension (N)
On entry, the n diagonal elements of the tridiagonal matrix T.
On exit, D may be multiplied by a constant factor chosen to avoid over/underflow in computing the eigenvalues.

E (input/output) REAL array, dimension (N)
On entry, the (n−1) subdiagonal elements of the tridiagonal matrix T in elements 1 to N−1 of E; E(N) need not be set.
On exit, E may be multiplied by a constant factor chosen to avoid over/underflow in computing the eigenvalues.

VL,VR (input) REAL
If RANGE='V', the lower and upper bounds of the interval to be searched for eigenvalues. VL < VU.
Not referenced if RANGE = 'A' or 'I'.

IL,IU (input) INTEGER
If RANGE='I', the indices (in ascending order) of the smallest and largest eigenvalues to be returned. $1 \leq IL \leq IU \leq N$, if $N > 0$; IL = 1 and IU = 0 if N = 0.
Not referenced if RANGE = 'A' or 'V'.

ABSTOL (input) REAL
The absolute error tolerance for the eigenvalues/eigenvectors.
IF JOBZ = 'V', the eigenvalues and eigenvectors output have residual norms bounded by ABSTOL, and the dot products between different eigenvectors are bounded by ABSTOL. If ABSTOL $< N*EPS*\|T\|_1$, then $N*EPS*\|T\|_1$ will be used in its place, where EPS is the machine precision. The eigenvalues are computed to an accuracy of $EPS*\|T\|_1$ irrespective of ABSTOL. If high relative accuracy is important, set ABSTOL to SLAMCH('S'). Doing so will guarantee that eigenvalues are computed to high relative accuracy when possible in future releases. The current code does not make any guarantees about high relative accuracy, but future releases will.

M (output) INTEGER
The total number of eigenvalues found. $0 \leq M \leq N$.
If RANGE = 'A', M = N, and if RANGE = 'I', M = IU−IL+1.

W (output) REAL array, dimension (N)
The first M elements contain the selected eigenvalues in ascending order.

Z (output) REAL array, dimension (LDZ, max(1,M))
If JOBZ = 'V', then if INFO = 0, the first M columns of Z contain the orthonormal eigenvectors of the matrix T corresponding to the selected eigenvalues, with the ith column of Z holding the eigenvector associated with W(i).
If JOBZ = 'N', then Z is not referenced.
Note: the user must ensure that at least max(1,M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and an upper bound must be used.

LDZ (input) INTEGER
The leading dimension of the array Z. $LDZ \geq 1$, and if JOBZ = 'V', $LDZ \geq \max(1,N)$.

ISUPPZ (output) INTEGER array, dimension (2*max(1,M))
The support of the eigenvectors in Z, i.e., the indices indicating the nonzero elements in Z. The ith eigenvector is nonzero only in elements ISUPPZ(2*i−1) through ISUPPZ(2*i).

WORK (workspace/output) REAL array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal (and minimal) LWORK.

LWORK (input) INTEGER
The dimension of the array WORK. $LWORK \geq \max(1,20*N)$.

If LWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

IWORK (workspace/output) INTEGER array, dimension (LIWORK)
On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK (input) INTEGER
The dimension of the array IWORK. $LIWORK \geq \max(1, 10*N)$.

If LIWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the IWORK array, returns this value as the first entry of the IWORK array, and no error message related to LIWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: Internal error

SSTEVX

```
      SUBROUTINE SSTEVX( JOBZ, RANGE, N, D, E, VL, VU, IL, IU, ABSTOL,
     $                   M, W, Z, LDZ, WORK, IWORK, IFAIL, INFO )
      CHARACTER          JOBZ, RANGE
      INTEGER            IL, INFO, IU, LDZ, M, N
      REAL               ABSTOL, VL, VU
      INTEGER            IFAIL( * ), IWORK( * )
      REAL               D( * ), E( * ), W( * ), WORK( * ), Z( LDZ, * )
```

Purpose

SSTEVX computes selected eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix A. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

Arguments

JOBZ (input) CHARACTER*1
= 'N': Compute eigenvalues only;
= 'V': Compute eigenvalues and eigenvectors.

RANGE (input) CHARACTER*1
= 'A': all eigenvalues will be found.
= 'V': all eigenvalues in the half-open interval (VL,VU] will be found.
= 'I': the ILth through IU^{th} eigenvalues will be found.

N (input) INTEGER
The order of the matrix A. $N \geq 0$.

D (input/output) REAL array, dimension (N)
On entry, the n diagonal elements of the tridiagonal matrix A.
On exit, D may be multiplied by a constant factor chosen to avoid over/underflow in computing the eigenvalues.

E (input/output) REAL array, dimension (N)
On entry, the (n−1) subdiagonal elements of the tridiagonal matrix A, stored in elements 1 to n−1 of E; E(n) need not be set.
On exit, E may be multiplied by a constant factor chosen to avoid over/underflow in computing the eigenvalues.

VL, VU (input) REAL
If RANGE='V', the lower and upper bounds of the interval to be searched for eigenvalues. VL < VU.
Not referenced if RANGE = 'A' or 'I'.

IL, IU (input) INTEGER
If RANGE='I', the indices (in ascending order) of the smallest and largest eigenvalues to be returned.
$1 \leq IL \leq IU \leq N$, if $N > 0$; IL = 1 and IU = 0 if N = 0.
Not referenced if RANGE = 'A' or 'V'.

ABSTOL (input) REAL
The absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to ABSTOL + EPS*max(|a|,|b|), where EPS is the machine precision. If ABSTOL is less than or equal to zero, then $EPS*||A||_1$ will be used in its place.
Eigenvalues will be computed most accurately when ABSTOL is set to twice the underflow threshold 2*SLAMCH('S'), not zero. If this routine returns with INFO>0, indicating that some eigenvectors did not converge, try setting ABSTOL to 2*SLAMCH('S').

M (output) INTEGER
The total number of eigenvalues found. $0 \leq M \leq N$.
If RANGE = 'A', M = N, and if RANGE = 'I', M = IU−IL+1.

W (output) REAL array, dimension (N)
The first M elements contain the selected eigenvalues in ascending order.

Z (output) REAL array, dimension (LDZ,max(1,M))
If JOBZ = 'V', then if INFO = 0, the first M columns of Z contain the orthonormal eigenvectors of the matrix A corresponding to the selected eigenvalues, with the ith column of Z holding the eigenvector associated with W(i). If an eigenvector fails to converge (INFO > 0), then that column of Z contains the latest approximation to the eigenvector, and the index of the eigenvector is returned in IFAIL.
If JOBZ = 'N', then Z is not referenced.
Note: the user must ensure that at least max(1,M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and an upper bound must be used.

LDZ (input) INTEGER
The leading dimension of the array Z. $LDZ \geq 1$, and if JOBZ = 'V', $LDZ \geq \max(1,N)$.

WORK (workspace) REAL array, dimension (5*N)

IWORK (workspace) INTEGER array, dimension (5*N)

IFAIL (output) INTEGER array, dimension (N)
If JOBZ = 'V', then if INFO = 0, the first M elements of IFAIL are zero; if INFO > 0, then IFAIL contains the indices of the eigenvectors that failed to converge.
If JOBZ = 'N', then IFAIL is not referenced.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, then i eigenvectors failed to converge. Their indices are stored in array IFAIL.

SSYCON/CSYCON/CHECON

```
      SUBROUTINE SSYCON( UPLO, N, A, LDA, IPIV, ANORM, RCOND, WORK,
     $                   IWORK, INFO )
      CHARACTER          UPLO
      INTEGER            INFO, LDA, N
      REAL               ANORM, RCOND
      INTEGER            IPIV( * ), IWORK( * )
      REAL               A( LDA, * ), WORK( * )
```

```
      SUBROUTINE CSYCON( UPLO, N, A, LDA, IPIV, ANORM, RCOND, WORK,
     $                   INFO )
      CHARACTER          UPLO
      INTEGER            INFO, LDA, N
      REAL               ANORM, RCOND
      INTEGER            IPIV( * )
      COMPLEX            A( LDA, * ), WORK( * )

      SUBROUTINE CHECON( UPLO, N, A, LDA, IPIV, ANORM, RCOND, WORK,
     $                   INFO )
      CHARACTER          UPLO
      INTEGER            INFO, LDA, N
      REAL               ANORM, RCOND
      INTEGER            IPIV( * )
      COMPLEX            A( LDA, * ), WORK( * )
```

Purpose

SSYCON/CSYCON estimates the reciprocal of the condition number (in the 1-norm) of a real/complex symmetric matrix A using the factorization A = U*D*U^T or A = L*D*L^T computed by SSYTRF/CSYTRF.

CHECON estimates the reciprocal of the condition number of a complex Hermitian matrix A using the factorization A = U*D*U^H or A = L*D*L^H computed by CHETRF.

An estimate is obtained for $||A^{-1}||$, and the reciprocal of the condition number is computed as RCOND = $1/(||A|| * ||A^{-1}||)$.

Arguments

UPLO (input) CHARACTER*1
Specifies whether the details of the factorization are stored as an upper or lower triangular matrix.
= 'U': Upper triangular, form is A = U*D*U^T (SSYCON/CSYCON) or A = U*D*U^H (CHECON);
= 'L': Lower triangular, form is A = L*D*L^T (SSYCON/CSYCON) or A = L*D*L^H (CHECON).

N (input) INTEGER
The order of the matrix A. N $\geq$ 0.

A (input) REAL/COMPLEX/COMPLEX array, dimension (LDA,N)
The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by SSYTRF/CSYTRF/CHETRF.

LDA (input) INTEGER
The leading dimension of the array A. LDA $\geq$ max(1,N).

IPIV (input) INTEGER array, dimension (N)
Details of the interchanges and the block structure of D as determined by SSYTRF/CSYTRF/CHETRF.

ANORM (input) REAL
The 1-norm of the original matrix A.

RCOND (output) REAL
The reciprocal of the condition number of the matrix A, computed as RCOND = $1/(||A|| * ||A^{-1}||)$.

WORK (workspace) REAL/COMPLEX/COMPLEX array, dimension (2*N)

IWORK *SSYCON only* (workspace) INTEGER array, dimension (N)

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SSYEV/CHEEV

```
      SUBROUTINE SSYEV( JOBZ, UPLO, N, A, LDA, W, WORK, LWORK, INFO )
      CHARACTER          JOBZ, UPLO
      INTEGER            INFO, LDA, LWORK, N
      REAL               A( LDA, * ), W( * ), WORK( * )

      SUBROUTINE CHEEV( JOBZ, UPLO, N, A, LDA, W, WORK, LWORK, RWORK,
     $                  INFO )
      CHARACTER          JOBZ, UPLO
      INTEGER            INFO, LDA, LWORK, N
      REAL               RWORK( * ), W( * )
      COMPLEX            A( LDA, * ), WORK( * )
```

Purpose

SSYEV/CHEEV computes all eigenvalues and, optionally, eigenvectors of a real/complex symmetric/Hermitian matrix A.

Arguments

JOBZ (input) CHARACTER*1
= 'N': Compute eigenvalues only;
= 'V': Compute eigenvalues and eigenvectors.

UPLO (input) CHARACTER*1
= 'U': Upper triangle of A is stored;
= 'L': Lower triangle of A is stored.

N (input) INTEGER
The order of the matrix A. N $\geq$ 0.

A (input/output) REAL/COMPLEX array, dimension (LDA, N)
On entry, the symmetric/Hermitian matrix A. If UPLO = 'U', the leading n-by-n upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading n-by-n lower triangular part of A contains the lower triangular part of the matrix A.
On exit, if JOBZ = 'V', then if INFO = 0, A contains the orthonormal eigenvectors of the matrix A.

If JOBZ = 'N', then on exit the lower triangle (if UPLO='L') or the upper triangle (if UPLO='U') of A, including the diagonal, is destroyed.

LDA (input) INTEGER
The leading dimension of the array A. LDA $\geq$ max(1,N).

W (output) REAL array, dimension (N)
If INFO = 0, the eigenvalues in ascending order.

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The length of the array WORK.
SSYEV
LWORK $\geq$ max(1,3*N−1). For optimal efficiency, LWORK $\geq$ (NB+2)*N, where NB is the block size for SSYTRD returned by ILAENV.
CHEEV
LWORK $\geq$ max(1,2*N−1). For optimal efficiency, LWORK $\geq$ (NB+1)*N, where NB is the block size for CHETRD returned by ILAENV.

RWORK *CHEEV only* (workspace) REAL array, dimension (max(1,3*N−2))

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value
> 0: the algorithm failed to converge; if INFO = i, i off-diagonal elements of an intermediate tridiagonal form did not converge to zero.

SSYEVD/CHEEVD

```
      SUBROUTINE SSYEVD( JOBZ, UPLO, N, A, LDA, W, WORK, LWORK, IWORK,
     $                   LIWORK, INFO )
      CHARACTER          JOBZ, UPLO
      INTEGER            INFO, LDA, LIWORK, LWORK, N
      INTEGER            IWORK( * )
      REAL               A( LDA, * ), W( * ), WORK( * )

      SUBROUTINE CHEEVD( JOBZ, UPLO, N, A, LDA, W, WORK, LWORK, RWORK,
     $                   LRWORK, IWORK, LIWORK, INFO )
      CHARACTER          JOBZ, UPLO
      INTEGER            INFO, LDA, LIWORK, LRWORK, LWORK, N
      INTEGER            IWORK( * )
      REAL               RWORK( * ), W( * )
      COMPLEX            A( LDA, * ), WORK( * )
```

Purpose

SSYEVD/CHEEVD computes all eigenvalues and, optionally, eigenvectors of a real/complex symmetric/Hermitian matrix A. If eigenvectors are desired, it uses a divide and conquer algorithm.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Arguments

JOBZ (input) CHARACTER*1
= 'N': Compute eigenvalues only;
= 'V': Compute eigenvalues and eigenvectors.

UPLO (input) CHARACTER*1
= 'U': Upper triangle of A is stored;
= 'L': Lower triangle of A is stored.

N (input) INTEGER
The order of the matrix A. N $\geq$ 0.

A (input/output) REAL/COMPLEX array, dimension (LDA, N)
On entry, the symmetric/Hermitian matrix A. If UPLO = 'U', the leading n-by-n upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading n-by-n lower triangular part of A contains the lower triangular part of the matrix A.
On exit, if JOBZ = 'V', then if INFO = 0, A contains the orthonormal eigenvectors of the matrix A.
If JOBZ = 'N', then on exit the lower triangle (if UPLO='L') or the upper triangle (if UPLO='U') of A, including the diagonal, is destroyed.

LDA (input) INTEGER
The leading dimension of the array A. LDA $\geq$ max(1,N).

W (output) REAL array, dimension (N)
If INFO = 0, the eigenvalues in ascending order.

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK.
If N $\leq$ 1, LWORK $\geq$ 1.
SSYEVD
If JOBZ = 'N' and N > 1, LWORK $\geq$ 2*N+1.
If JOBZ = 'V' and N > 1, LWORK $\geq$ $(1 + 6*N + 2*N^2)$.
CHEEVD
If JOBZ = 'N' and N > 1, LWORK $\geq$ N + 1.
If JOBZ = 'V' and N > 1, LWORK $\geq$ $2*N + N^2$.

If LWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK *CHEEVD only* (workspace/output) REAL array, dimension (LRWORK)
On exit, if INFO = 0, RWORK(1) returns the optimal LRWORK.

LRWORK *CHEEVD only* (input) INTEGER
The dimension of the array RWORK.
If N ≤ 1, LRWORK ≥ 1.
If JOBZ = 'N' and N > 1, LRWORK ≥ N.
If JOBZ = 'V' and N > 1, LRWORK ≥ $(1 + 5*N + 2*N^2)$.

If LRWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the RWORK array, returns this value as the first entry of the RWORK array, and no error message related to LRWORK is issued by XERBLA.

IWORK (workspace/output) INTEGER array, dimension (LIWORK)
On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK (input) INTEGER
The dimension of the array IWORK.
If N ≤ 1, LIWORK ≥ 1.
If JOBZ = 'N' and N > 1, LIWORK ≥ 1.
If JOBZ = 'V' and N > 1, LIWORK ≥ 3 + 5*N.

If LIWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the IWORK array, returns this value as the first entry of the IWORK array, and no error message related to LIWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero.

SSYEVR/CHEEVR

```
 SUBROUTINE SSYEVR( JOBZ, RANGE, UPLO, N, A, LDA, VL, VU, IL, IU,
$                   ABSTOL, M, W, Z, LDZ, ISUPPZ, WORK, LWORK,
$                   IWORK, LIWORK, INFO )
 CHARACTER          JOBZ, RANGE, UPLO
 INTEGER            IL, INFO, IU, LDA, LDZ, LIWORK, LWORK,
$                   M, N
 REAL               ABSTOL, VL, VU
 INTEGER            ISUPPZ( * ), IWORK( * )
 REAL               A( LDA, * ), W( * ), WORK( * ), Z( LDZ, * )

 SUBROUTINE CHEEVR( JOBZ, RANGE, UPLO, N, A, LDA, VL, VU, IL, IU,
$                   ABSTOL, M, W, Z, LDZ, ISUPPZ, WORK, LWORK,
$                   RWORK, LRWORK, IWORK, LIWORK, INFO )
 CHARACTER          JOBZ, RANGE, UPLO
 INTEGER            IL, INFO, IU, LDA, LDZ, LIWORK, LWORK,
$                   M, N, LRWORK
 REAL               ABSTOL, VL, VU
 INTEGER            ISUPPZ( * ), IWORK( * )
 REAL               RWORK( * ), W( * )
 COMPLEX            A( LDA, * ), WORK( * ), Z( LDZ, * )
```

Purpose

SSYEVR/CHEEVR computes selected eigenvalues and, optionally, eigenvectors of a real/complex symmetric/Hermitian tridiagonal matrix T. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

Whenever possible, SSYEVR/CHEEVR calls SSTEGR/CSTEGR to compute the eigenspectrum using Relatively Robust Representations. SSTEGR/CSTEGR computes eigenvalues by the dqds algorithm, while orthogonal eigenvectors are computed from various "good" LDL^T representations (also known as Relatively Robust Representations). Gram-Schmidt orthogonalization is avoided as far as possible. More specifically, the various steps of the algorithm are as follows. For the ith unreduced block of T,

(a) Compute $T - \sigma_i = L_i D_i L_i^T$, such that $L_i D_i L_i^T$ is a relatively robust representation,

(b) Compute the eigenvalues, λ_j, of $L_i D_i L_i^T$ to high relative accuracy by the dqds algorithm,

(c) If there is a cluster of close eigenvalues, "choose" σ_i close to the cluster, and go to step (a),

(d) Given the approximate eigenvalue λ_j of $L_i D_i L_i^T$, compute the corresponding eigenvector by forming a rank-revealing twisted factorization.

The desired accuracy of the output can be specified by the input parameter ABSTOL.

Note 1: SSYEVR/CHEEVR calls SSTEGR/CSTEGR when the full spectrum is requested on machines which conform to the ieee-754 floating point standard. SSYEVR/CHEEVR calls SSTEBZ and SSTEIN/CSTEIN on non-ieee machines and when partial spectrum requests are made.

Normal execution of SSTEGR/CSTEGR may create NaNs and infinities and hence may abort due to a floating point exception in environments which do not handle NaNs and infinities in the ieee standard default manner.

Arguments

JOBZ (input) CHARACTER*1
= 'N': Compute eigenvalues only;
= 'V': Compute eigenvalues and eigenvectors.

RANGE (input) CHARACTER*1
= 'A': all eigenvalues will be found;
= 'V': all eigenvalues in the half-open interval (VL,VU] will be found;
= 'I': the ILth through IU^{th} eigenvalues will be found.
For RANGE = 'V' or 'I', SSTEBZ and SSTEIN are called.

N (input) INTEGER
The order of the matrix A. $N \geq 0$.

A (input/output) REAL/COMPLEX array, dimension (LDA, N)
On entry, the symmetric/Hermitian matrix A. If UPLO = 'U', the leading n-by-n upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading n-by-n lower triangular part of A contains the lower triangular part of the matrix A.
On exit, the lower triangle (if UPLO='L') or the upper triangle (if UPLO='U') of A, including the diagonal, is destroyed.

LDA (input) INTEGER
The leading dimension of the array A. $LDA \geq \max(1,N)$.

VL,VR (input) REAL
If RANGE='V', the lower and upper bounds of the interval to be searched for eigenvalues. $VL < VU$.
Not referenced if RANGE = 'A' or 'I'.

IL,IU (input) INTEGER
If RANGE='I', the indices (in ascending order) of the smallest and largest eigenvalues to be returned. $1 \leq IL \leq IU \leq N$, if $N > 0$; $IL = 1$ and $IU = 0$ if $N = 0$.
Not referenced if RANGE = 'A' or 'V'.

ABSTOL (input) REAL
The absolute error tolerance for the eigenvalues/eigenvectors.
IF JOBZ = 'V', the eigenvalues and eigenvectors output have residual norms bounded by ABSTOL, and the dot products between different eigenvectors are bounded by ABSTOL. If $ABSTOL < N*EPS*||T||_1$, then $N*EPS*||T||_1$ will be used in its place, where EPS is the machine precision. The eigenvalues are computed to an accuracy of $EPS*||T||_1$ irrespective of ABSTOL. If high relative accuracy is important, set ABSTOL to SLAMCH('S'). Doing so will guarantee that eigenvalues are computed to high relative accuracy when possible in future releases. The current code does not make any guarantees about high relative accuracy, but future releases will.

M (output) INTEGER
The total number of eigenvalues found. $0 \leq M \leq N$.
If RANGE = 'A', M = N, and if RANGE = 'I', M = IU−IL+1.

W (output) REAL array, dimension (N)
The first M elements contain the selected eigenvalues in ascending order.

Z (output) REAL/COMPLEX array, dimension (LDZ, max(1,M))
If JOBZ = 'V', then if INFO = 0, the first M columns of Z contain the orthonormal eigenvectors of the matrix T corresponding to the selected eigenvalues, with the ith column of Z holding the eigenvector associated with W(i).
If JOBZ = 'N', then Z is not referenced.
Note: the user must ensure that at least max(1,M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and an upper bound must be used.

LDZ (input) INTEGER
The leading dimension of the array Z. $LDZ \geq 1$, and if JOBZ = 'V', $LDZ \geq \max(1,N)$.

ISUPPZ (output) INTEGER array, dimension (2*max(1,M))
The support of the eigenvectors in Z, i.e., the indices indicating the nonzero elements in Z. The ith eigenvector is nonzero only in elements ISUPPZ(2*i−1) through ISUPPZ(2*i).

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal (and minimal) LWORK.

LWORK (input) INTEGER
The dimension of the array WORK. *SSYEVR*
$LWORK \geq \max(1,26*N)$.
For optimal efficiency, $LWORK \geq (NB+6)*N$, where NB is the max of the blocksize for SSYTRD and SORMTR returned by ILAENV.
CHEEVR
$LWORK \geq \max(1,2*N)$.
For optimal efficiency, $LWORK \geq (NB+1)*N$, where NB is the max of the blocksize for CHETRD and for CUNMTR as returned by ILAENV.

If LWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the

first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK *CHEEVR only* (workspace/output) REAL array, dimension (LRWORK)
On exit, if INFO = 0, RWORK(1) returns the optimal (and minimal) LRWORK.

LRWORK *CHEEVR only* (input) INTEGER
The length of the array RWORK. LRWORK $\geq$ max(1,24*N).

If LRWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the RWORK array, returns this value as the first entry of the RWORK array, and no error message related to LRWORK is issued by XERBLA.

IWORK (workspace/output) INTEGER array, dimension (LIWORK)
On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK (input) INTEGER
The dimension of the array IWORK. LIWORK $\geq$ max(1, 10*N).

If LIWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the IWORK array, returns this value as the first entry of the IWORK array, and no error message related to LIWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: Internal error

SSYEVX/CHEEVX

```
      SUBROUTINE SSYEVX( JOBZ, RANGE, UPLO, N, A, LDA, VL, VU, IL, IU,
     $                   ABSTOL, M, W, Z, LDZ, WORK, LWORK, IWORK,
     $                   IFAIL, INFO )
      CHARACTER          JOBZ, RANGE, UPLO
      INTEGER            IL, INFO, IU, LDA, LDZ, LWORK, M, N
      REAL               ABSTOL, VL, VU
      INTEGER            IFAIL( * ), IWORK( * )
      REAL               A( LDA, * ), W( * ), WORK( * ), Z( LDZ, * )
```

```
      SUBROUTINE CHEEVX( JOBZ, RANGE, UPLO, N, A, LDA, VL, VU, IL, IU,
     $                   ABSTOL, M, W, Z, LDZ, WORK, LWORK, RWORK,
     $                   IWORK, IFAIL, INFO )
      CHARACTER          JOBZ, RANGE, UPLO
      INTEGER            IL, INFO, IU, LDA, LDZ, LWORK, M, N
      REAL               ABSTOL, VL, VU
      INTEGER            IFAIL( * ), IWORK( * )
      REAL               RWORK( * ), W( * )
      COMPLEX            A( LDA, * ), WORK( * ), Z( LDZ, * )
```

Purpose

SSYEVX/CHEEVX computes selected eigenvalues and, optionally, eigenvectors of a real/complex symmetric/Hermitian matrix A. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

Arguments

JOBZ (input) CHARACTER*1
= 'N': Compute eigenvalues only;
= 'V': Compute eigenvalues and eigenvectors.

RANGE (input) CHARACTER*1
= 'A': all eigenvalues will be found.
= 'V': all eigenvalues in the half-open interval (VL,VU] will be found.
= 'I': the ILth through IU^{th} eigenvalues will be found.

UPLO (input) CHARACTER*1
= 'U': Upper triangle of A is stored;
= 'L': Lower triangle of A is stored.

N (input) INTEGER
The order of the matrix A. N $\geq$ 0.

A (input/output) REAL/COMPLEX array, dimension (LDA, N)
On entry, the symmetric/Hermitian matrix A. If UPLO = 'U', the leading n-by-n upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading n-by-n lower triangular part of A contains the lower triangular part of the matrix A.
On exit, the lower triangle (if UPLO='L') or the upper triangle (if UPLO='U') of A, including the diagonal, is destroyed.

LDA (input) INTEGER
The leading dimension of the array A. LDA $\geq$ max(1,N).

VL, VU (input) REAL
If RANGE='V', the lower and upper bounds of the interval to be searched for eigenvalues. VL < VU.
Not referenced if RANGE = 'A' or 'I'.

IL, IU (input) INTEGER
If RANGE='I', the indices (in ascending order) of the smallest and largest eigenvalues to be returned.

$1 \le IL \le IU \le N$, if $N > 0$; $IL = 1$ and $IU = 0$ if $N = 0$.
Not referenced if RANGE = 'A' or 'V'.

ABSTOL (input) REAL
The absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to ABSTOL + EPS*max(|a|,|b|), where EPS is the machine precision. If ABSTOL is less than or equal to zero, then $EPS*||T||_1$ will be used in its place, where T is the tridiagonal matrix obtained by reducing A to tridiagonal form.
Eigenvalues will be computed most accurately when ABSTOL is set to twice the underflow threshold 2*SLAMCH('S'), not zero. If this routine returns with INFO>0, indicating that some eigenvectors did not converge, try setting ABSTOL to 2*SLAMCH('S').

M (output) INTEGER
The total number of eigenvalues found. $0 \le M \le N$.
If RANGE = 'A', M = N, and if RANGE = 'I', M = IU−IL+1.

W (output) REAL array, dimension (N)
The first m elements contain the selected eigenvalues in ascending order.

Z (output) REAL/COMPLEX array, dimension (LDZ, max(1,M))
If JOBZ = 'V', then if INFO = 0, the first M columns of Z contain the orthonormal eigenvectors of the matrix A corresponding to the selected eigenvalues, with the ith column of Z holding the eigenvector associated with W(i). If an eigenvector fails to converge, then that column of Z contains the latest approximation to the eigenvector, and the index of the eigenvector is returned in IFAIL.
If JOBZ = 'N', then Z is not referenced.
Note: the user must ensure that at least max(1,M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and an upper bound must be used.

LDZ (input) INTEGER
The leading dimension of the array Z. $LDZ \ge 1$, and if JOBZ = 'V', $LDZ \ge \max(1,N)$.

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The length of the array WORK.
SSYEVX
$LWORK \ge \max(1,8*N)$. For optimal efficiency, $LWORK \ge (NB+3)*N$, where NB is the max of the block size for SSYTRD and SORMTR returned by ILAENV.
CHEEVX
$LWORK \ge \max(1,2*N-1)$. For optimal efficiency, $LWORK \ge (NB+1)*N$, where NB is the max of the block size for CHETRD and for CUNMTR as returned by ILAENV.

RWORK *CHEEVX only* (workspace) REAL array, dimension (7*N)

IWORK (workspace) INTEGER array, dimension (5*N)

IFAIL (output) INTEGER array, dimension (N)
If JOBZ = 'V', then if INFO = 0, the first M elements of IFAIL are zero; if INFO > 0, then IFAIL contains the indices of the eigenvectors that failed to converge.
If JOBZ = 'N', then IFAIL is not referenced.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, then i eigenvectors failed to converge. Their indices are stored in array IFAIL.

SSYGST/CHEGST

```
SUBROUTINE SSYGST( ITYPE, UPLO, N, A, LDA, B, LDB, INFO )
CHARACTER          UPLO
INTEGER            INFO, ITYPE, LDA, LDB, N
REAL               A( LDA, * ), B( LDB, * )

SUBROUTINE CHEGST( ITYPE, UPLO, N, A, LDA, B, LDB, INFO )
CHARACTER          UPLO
INTEGER            INFO, ITYPE, LDA, LDB, N
COMPLEX            A( LDA, * ), B( LDB, * )
```

Purpose

SSYGST/CHEGST reduces a real/complex symmetric/Hermitian definite generalized eigenproblem to standard form.

If ITYPE = 1, the problem is $A*x = \lambda*B*x$, and A is overwritten by $(U^H)^{-1}*A*U^{-1}$ or $L^{-1}*A*(L^H)^{-1}$.

If ITYPE = 2 or 3, the problem is $A*B*x = \lambda*x$ or $B*A*x = \lambda*x$, and A is overwritten by $U*A*U^H$ or L^H*A*L.

B must have been previously factorized as U^H*U or $L*L^H$ by SPOTRF/CPOTRF.

Arguments

ITYPE (input) INTEGER
= 1: compute $(U^H)^{-1}*A*U^{-1}$ or $L^{-1}*A*(L^H)^{-1}$;
= 2 or 3: compute $U*A*U^H$ or L^H*A*L.

UPLO (input) CHARACTER
= 'U': Upper triangle of A is stored and B is factored as U^H*U;
= 'L': Lower triangle of A is stored and B is factored as $L*L^H$.

N (input) INTEGER
The order of the matrices A and B. $N \ge 0$.

A (input/output) REAL/COMPLEX array, dimension (LDA,N)
On entry, the symmetric/Hermitian matrix A. If UPLO = 'U', the leading n-by-n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n-by-n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.
On exit, if INFO = 0, the transformed matrix, stored in the same format as A.

LDA (input) INTEGER
The leading dimension of the array A. LDA $\geq$ max(1,N).

B (input) REAL/COMPLEX array, dimension (LDB,N)
The triangular factor from the Cholesky factorization of B, as returned by SPOTRF/CPOTRF.

LDB (input) INTEGER
The leading dimension of the array B. LDB $\geq$ max(1,N).

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = $-$i, the ith argument had an illegal value.

SSYGV/CHEGV

```
      SUBROUTINE SSYGV( ITYPE, JOBZ, UPLO, N, A, LDA, B, LDB, W, WORK,
     $                  LWORK, INFO )
      CHARACTER          JOBZ, UPLO
      INTEGER            INFO, ITYPE, LDA, LDB, LWORK, N
      REAL               A( LDA, * ), B( LDB, * ), W( * ), WORK( * )

      SUBROUTINE CHEGV( ITYPE, JOBZ, UPLO, N, A, LDA, B, LDB, W, WORK,
     $                  LWORK, RWORK, INFO )
      CHARACTER          JOBZ, UPLO
      INTEGER            INFO, ITYPE, LDA, LDB, LWORK, N
      REAL               RWORK( * ), W( * )
      COMPLEX            A( LDA, * ), B( LDB, * ), WORK( * )
```

Purpose

SSYGV/CHEGV computes all the eigenvalues, and optionally, the eigenvectors of a real/complex generalized symmetric/Hermitian definite eigenproblem, of the form

$$A*x = \lambda*B*x,\ A*Bx = \lambda*x, \text{ or } B*A*x = \lambda*x.$$

Here A and B are assumed to be symmetric/Hermitian and B is also positive definite.

Arguments

ITYPE (input) INTEGER
Specifies the problem type to be solved:
= 1: $A*x = \lambda*B*x$
= 2: $A*B*x = \lambda*x$
= 3: $B*A*x = \lambda*x$

JOBZ (input) CHARACTER*1
= 'N': Compute eigenvalues only;
= 'V': Compute eigenvalues and eigenvectors.

UPLO (input) CHARACTER*1
= 'U': Upper triangles of A and B are stored;
= 'L': Lower triangles of A and B are stored.

N (input) INTEGER
The order of the matrices A and B. N $\geq$ 0.

A (input/output) REAL/COMPLEX array, dimension (LDA, N)
On entry, the symmetric/Hermitian matrix A. If UPLO = 'U', the leading n-by-n upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading n-by-n lower triangular part of A contains the lower triangular part of the matrix A.
On exit, if JOBZ = 'V', then if INFO = 0, A contains the matrix Z of eigenvectors. The eigenvectors are normalized as follows:
if ITYPE = 1 or 2, $Z^H*B*Z = I$;
if ITYPE = 3, $Z^H*B^{-1}*Z = I$.
If JOBZ = 'N', then on exit the upper triangle (if UPLO='U') or the lower triangle (if UPLO='L') of A, including the diagonal, is destroyed.

LDA (input) INTEGER
The leading dimension of the array A. LDA $\geq$ max(1,N).

B (input/output) REAL/COMPLEX array, dimension (LDB, N)
On entry, the symmetric/Hermitian positive definite matrix B. If UPLO = 'U', the leading n-by-n upper triangular part of B contains the upper triangular part of the matrix B. If UPLO = 'L', the leading n-by-n lower triangular part of B contains the lower triangular part of the matrix B.
On exit, if INFO $\leq$ N, the part of B containing the matrix is overwritten by the triangular factor U or L from the Cholesky factorization $B = U^H*U$ or $B = L*L^H$.

LDB (input) INTEGER
The leading dimension of the array B. LDB $\geq$ max(1,N).

W (output) REAL array, dimension (N)
If INFO = 0, the eigenvalues in ascending order.

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The length of the array WORK.

LWORK $\geq$ max(1,3*N−1). For optimal efficiency, LWORK $\geq$ (NB+2)*N, where NB is the block size for SSYTRD returned by ILAENV.
CHEGV
LWORK $\geq$ max(1,2*N−1). For optimal efficiency, LWORK $\geq$ (NB+1)*N, where NB is the block size for CHETRD returned by ILAENV.

If LWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK *CHEGV only* (workspace) REAL array, dimension (max(1,3*N−2))

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value
> 0: SPOTRF/CPOTRF or SSYEV/CHEEV returned an error code:
$\leq$ N: SSYEV/CHEEV failed to converge; if INFO = i, i off-diagonal elements of an intermediate tridiagonal form did not converge to zero.
> N: if INFO = N + i, for 1 $\leq$ i $\leq$ N, then the leading minor of order i of B is not positive definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed.

SSYGVD/CHEGVD

```
      SUBROUTINE SSYGVD( ITYPE, JOBZ, UPLO, N, A, LDA, B, LDB, W, WORK,
     $                   LWORK, IWORK, LIWORK, INFO )
      CHARACTER          JOBZ, UPLO
      INTEGER            INFO, ITYPE, LDA, LDB, LIWORK, LWORK, N
      INTEGER            IWORK( * )
      REAL               A( LDA, * ), B( LDB, * ), W( * ), WORK( * )

      SUBROUTINE CHEGVD( ITYPE, JOBZ, UPLO, N, A, LDA, B, LDB, W, WORK,
     $                   LWORK, RWORK, LRWORK, IWORK, LIWORK, INFO )
      CHARACTER          JOBZ, UPLO
      INTEGER            INFO, ITYPE, LDA, LDB, LIWORK, LRWORK, LWORK, N
      INTEGER            IWORK( * )
      REAL               RWORK( * ), W( * )
      COMPLEX            A( LDA, * ), B( LDB, * ), WORK( * )
```

Purpose

SSYGVD/CHEGVD computes all the eigenvalues, and optionally, the eigenvectors of a real/complex generalized symmetric-definite/Hermitian-definite eigenproblem, of the form $A*x = \lambda*B*x$, $A*Bx = \lambda*x$, or $B*A*x = \lambda*x$. Here A and B are assumed to be symmetric/Hermitian and B is also positive definite. If eigenvectors are desired, it uses a divide and conquer algorithm.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Arguments

ITYPE (input) INTEGER
Specifies the problem type to be solved:
= 1: A*x = λ*B*x
= 2: A*B*x = λ*x
= 3: B*A*x = λ*x

JOBZ (input) CHARACTER*1
= 'N': Compute eigenvalues only;
= 'V': Compute eigenvalues and eigenvectors.

UPLO (input) CHARACTER*1
= 'U': Upper triangles of A and B are stored;
= 'L': Lower triangles of A and B are stored.

N (input) INTEGER
The order of the matrices A and B. N $\geq$ 0.

A (input/output) REAL/COMPLEX array, dimension (LDA, N)
On entry, the symmetric/Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A.

On exit, if JOBZ = 'V', then if INFO = 0, A contains the matrix Z of eigenvectors. The eigenvectors are normalized as follows:
if ITYPE = 1 or 2, $Z^H * B * Z = I$;
if ITYPE = 3, $Z^H * B^{-1} * Z = I$.
If JOBZ = 'N', then on exit the upper triangle (if UPLO='U') or the lower triangle (if UPLO='L') of A, including the diagonal, is destroyed.

LDA (input) INTEGER
The leading dimension of the array A. LDA $\geq$ max(1,N).

B (input/output) REAL/COMPLEX array, dimension (LDB, N)
On entry, the symmetric/Hermitian matrix B. If UPLO = 'U', the leading N-by-N upper triangular part of B contains the upper triangular part of the matrix B. If UPLO = 'L', the leading N-by-N lower triangular part of B contains the lower triangular part of the matrix B.

On exit, if INFO $\leq$ N, the part of B containing the matrix is overwritten by the triangular factor U or L from the Cholesky factorization

$B = U^H * U$ or $B = L * L^H$.

LDB (input) INTEGER
The leading dimension of the array B. LDB $\geq$ max(1,N).

W (output) REAL array, dimension (N)
If INFO = 0, the eigenvalues in ascending order.

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The length of the array WORK. If N $\leq$ 1, LWORK $\geq$ 1.
SSYGVD
If JOBZ = 'N' and N > 1, LWORK $\geq$ 2*N+1.
If JOBZ = 'V' and N > 1, LWORK $\geq$ 1 + 6*N + 2*N^2.
CHEGVD
If JOBZ = 'N' and N > 1, LWORK $\geq$ N + 1. If JOBZ = 'V' and N > 1, LWORK $\geq$ 2*N + N^2.

If LWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK *CHEGVD only* (workspace/output) REAL array, dimension (LRWORK)
On exit, if INFO = 0, RWORK(1) returns the optimal LRWORK.

LRWORK *CHEGVD only* (input) INTEGER
The dimension of the array RWORK.
If N $\leq$ 1, LRWORK $\geq$ 1.
If JOBZ = 'N' and N > 1, LRWORK $\geq$ N.
If JOBZ = 'V' and N > 1, LRWORK $\geq$ 1 + 5*N + 2*N^2.

If LRWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the RWORK array, returns this value as the first entry of the RWORK array, and no error message related to LRWORK is issued by XERBLA.

IWORK (workspace/output) INTEGER array, dimension (LIWORK)
On exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK (input) INTEGER
The dimension of the array IWORK.
If N $\leq$ 1, LIWORK $\geq$ 1.
If JOBZ = 'N' and N > 1, LIWORK $\geq$ 1.
If JOBZ = 'V' and N > 1, LIWORK $\geq$ 3 + 5*N.

If LIWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the IWORK array, returns this value as the first entry of the IWORK array, and no error message related to LIWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: SPOTRF/CPOTRF or SSYEVD/CHEEVD returned an error code:
$\leq$ N: if INFO = i, SSYEVD/CHEEVD failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero;
> N: if INFO = N + i, for 1 $\leq$ i $\leq$ N, then the leading minor of order i of B is not positive definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed.

SSYGVX/CHEGVX

```
 SUBROUTINE SSYGVX( ITYPE, JOBZ, RANGE, UPLO, N, A, LDA, B, LDB,
$                   VL, VU, IL, IU, ABSTOL, M, W, Z, LDZ, WORK,
$                   LWORK, IWORK, IFAIL, INFO )
 CHARACTER          JOBZ, RANGE, UPLO
 INTEGER            IL, INFO, ITYPE, IU, LDA, LDB, LDZ, LWORK, M, N
 REAL               ABSTOL, VL, VU
 INTEGER            IFAIL( * ), IWORK( * )
 REAL               A( LDA, * ), B( LDB, * ), W( * ), WORK( * ),
$                   Z( LDZ, * )

 SUBROUTINE CHEGVX( ITYPE, JOBZ, RANGE, UPLO, N, A, LDA, B, LDB,
$                   VL, VU, IL, IU, ABSTOL, M, W, Z, LDZ, WORK,
$                   LWORK, RWORK, IWORK, IFAIL, INFO )
 CHARACTER          JOBZ, RANGE, UPLO
 INTEGER            IL, INFO, ITYPE, IU, LDA, LDB, LDZ, LWORK, M, N
 REAL               ABSTOL, VL, VU
 INTEGER            IFAIL( * ), IWORK( * )
 REAL               RWORK( * ), W( * )
 COMPLEX            A( LDA, * ), B( LDB, * ), WORK( * ),
$                   Z( LDZ, * )
```

Purpose

SSYGVX/CHEGVX computes selected eigenvalues, and optionally, eigenvectors of a real/complex generalized symmetric-definite/Hermitian-definite eigenproblem, of the form $A*x = \lambda*B*x$, $A*Bx = \lambda*x$, or $B*A*x = \lambda*x$. Here A and B are assumed to be symmetric/Hermitian and B is also positive definite. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

Arguments

ITYPE (input) INTEGER
Specifies the problem type to be solved:
= 1: A*x = λ*B*x
= 2: A*B*x = λ*x
= 3: B*A*x = λ*x

JOBZ (input) CHARACTER*1
= 'N': Compute eigenvalues only;
= 'V': Compute eigenvalues and eigenvectors.

RANGE (input) CHARACTER*1
= 'A': all eigenvalues will be found.
= 'V': all eigenvalues in the half-open interval (VL,VU] will be found.
= 'I': the ILth through IU^{th} eigenvalues will be found.

UPLO (input) CHARACTER*1
= 'U': Upper triangles of A and B are stored;
= 'L': Lower triangles of A and B are stored.

N (input) INTEGER
The order of the matrices A and B. $N \geq 0$.

A (input/output) REAL/COMPLEX array, dimension (LDA, N)
On entry, the symmetric/Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A.

On exit, the lower triangle (if UPLO='L') or the upper triangle (if UPLO='U') of A, including the diagonal, is destroyed.

LDA (input) INTEGER
The leading dimension of the array A. LDA $\geq$ max(1,N).

B (input/output) REAL/COMPLEX array, dimension (LDB, N)
On entry, the symmetric/Hermitian matrix B. If UPLO = 'U', the leading N-by-N upper triangular part of B contains the upper triangular part of the matrix B. If UPLO = 'L', the leading N-by-N lower triangular part of B contains the lower triangular part of the matrix B.

On exit, if INFO $\leq$ N, the part of B containing the matrix is overwritten by the triangular factor U or L from the Cholesky factorization $B = U^H * U$ or $B = L * L^H$.

LDB (input) INTEGER
The leading dimension of the array B. LDB $\geq$ max(1,N).

VL,VU (input) REAL
If RANGE='V', the lower and upper bounds of the interval to be searched for eigenvalues. VL < VU.
Not referenced if RANGE = 'A' or 'I'.

IL,IU (input) INTEGER
If RANGE='I', the indices (in ascending order) of the smallest and largest eigenvalues to be returned. $1 \leq IL \leq IU \leq N$, if $N > 0$; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

ABSTOL (input) REAL
The absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to ABSTOL + EPS*max(|a|,|b|), where EPS is the machine precision. If ABSTOL is less than or equal to zero, then $EPS*||T||_1$ will be used in its place, where T is the tridiagonal matrix obtained by reducing A to tridiagonal form.
Eigenvalues will be computed most accurately when ABSTOL is set to twice the underflow threshold 2*SLAMCH('S'), not zero. If this routine returns with INFO>0, indicating that some eigenvectors did not converge, try setting ABSTOL to 2*SLAMCH('S').

M (output) INTEGER
The total number of eigenvalues found. $0 \leq M \leq N$.
If RANGE = 'A', M = N, and if RANGE = 'I', M = IU−IL+1.

W (output) REAL array, dimension (N)
The first m elements contain the selected eigenvalues in ascending order.

Z (output) REAL/COMPLEX array, dimension (LDZ, max(1,M))
If JOBZ = 'N', then Z is not referenced.
If JOBZ = 'V', then if INFO = 0, the first M columns of Z contain the orthonormal eigenvectors of the matrix A corresponding to the selected eigenvalues, with the ith column of Z holding the eigenvector associated with W(i). The eigenvectors are normalized as follows:
if ITYPE = 1 or 2, $Z^H * B * Z = I$;
if ITYPE = 3, $Z^H * B^{-1} * Z = I$.

If an eigenvector fails to converge, then that column of Z contains the latest approximation to the eigenvector, and the index of the eigenvector is returned in IFAIL. Note: the user must ensure that at least max(1,M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and an upper bound must be used.

LDZ (input) INTEGER
The leading dimension of the array Z. LDZ $\geq$ 1, and if JOBZ = 'V', LDZ $\geq$ max(1,N).

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The length of the array WORK.
SSYGVX
LWORK $\geq$ max(1,8*N). For optimal efficiency, LWORK $\geq$ (NB+3)*N, where NB is the blocksize for SSYTRD returned by ILAENV.
CHEGVX
LWORK $\geq$ max(1,2*N−1).

For optimal efficiency, LWORK ¿= (NB+1)*N, where NB is the block-size for CHETRD returned by ILAENV.

If LWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

RWORK *CHEGVX only* (workspace) REAL array, dimension (7*N)

IWORK (workspace) INTEGER array, dimension (5*N)

IFAIL (output) INTEGER array, dimension (N)
If JOBZ = 'V', then if INFO = 0, the first M elements of IFAIL are zero. If INFO > 0, then IFAIL contains the indices of the eigenvectors that failed to converge. If JOBZ = 'N', then IFAIL is not referenced.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = -i, the i-th argument had an illegal value.
> 0: SPOTRF/CPOTRF or SSYEVX/CHEEVX returned an error code:
≤ N: if INFO = i, SSYEVX/CHEEVX failed to converge; i eigenvectors failed to converge. Their indices are stored in array IFAIL.
> N: if INFO = N + i, for 1 ≤ i ≤ n, then the leading minor of order i of B is not positive definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed.

SSYRFS/CSYRFS/CHERFS

```
      SUBROUTINE SSYRFS( UPLO, N, NRHS, A, LDA, AF, LDAF, IPIV, B, LDB,
     $                   X, LDX, FERR, BERR, WORK, IWORK, INFO )
      CHARACTER          UPLO
      INTEGER            INFO, LDA, LDAF, LDB, LDX, N, NRHS
      INTEGER            IPIV( * ), IWORK( * )
      REAL               A( LDA, * ), AF( LDAF, * ), B( LDB, * ),
     $                   BERR( * ), FERR( * ), WORK( * ), X( LDX, * )

      SUBROUTINE CSYRFS( UPLO, N, NRHS, A, LDA, AF, LDAF, IPIV, B, LDB,
     $                   X, LDX, FERR, BERR, WORK, RWORK, INFO )
      CHARACTER          UPLO
      INTEGER            INFO, LDA, LDAF, LDB, LDX, N, NRHS
      INTEGER            IPIV( * )
      REAL               BERR( * ), FERR( * ), RWORK( * )
      COMPLEX            A( LDA, * ), AF( LDAF, * ), B( LDB, * ),
     $                   WORK( * ), X( LDX, * )

      SUBROUTINE CHERFS( UPLO, N, NRHS, A, LDA, AF, LDAF, IPIV, B, LDB,
     $                   X, LDX, FERR, BERR, WORK, RWORK, INFO )
      CHARACTER          UPLO
      INTEGER            INFO, LDA, LDAF, LDB, LDX, N, NRHS
      INTEGER            IPIV( * )
      REAL               BERR( * ), FERR( * ), RWORK( * )
      COMPLEX            A( LDA, * ), AF( LDAF, * ), B( LDB, * ),
     $                   WORK( * ), X( LDX, * )
```

Purpose

SSYRFS/CSYRFS/CHERFS improves the computed solution to a system of linear equations when the coefficient matrix is real/complex/complex symmetric/-symmetric/Hermitian indefinite, and provides error bounds and backward error estimates for the solution.

Arguments

UPLO (input) CHARACTER*1
= 'U': Upper triangle of A is stored;
= 'L': Lower triangle of A is stored.

N (input) INTEGER
The order of the matrix A. N ≥ 0.

NRHS (input) INTEGER
The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS ≥ 0.

A (input) REAL/COMPLEX/COMPLEX array, dimension (LDA,N)
The symmetric/symmetric/Hermitian matrix A. If UPLO = 'U', the leading n-by-n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n-by-n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

LDA (input) INTEGER
The leading dimension of the array A. LDA ≥ max(1,N).

AF (input) REAL/COMPLEX/COMPLEX array, dimension (LDAF,N)
The factored form of the matrix A. AF contains the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization $A = U*D*U^T$ or $A = L*D*L^T$ as computed by SSYTRF/CSYTRF or the factorization $A = U*D*U^H$ or $A = L*D*L^H$ as computed by CHETRF.

LDAF (input) INTEGER
The leading dimension of the array AF. LDAF ≥ max(1,N).

IPIV (input) INTEGER array, dimension (N)
Details of the interchanges and the block structure of D as determined by SSYTRF/CSYTRF/CHETRF.

B (input) REAL/COMPLEX array, dimension (LDB,NRHS)
The right hand side matrix B.

LDB (input) INTEGER
The leading dimension of the array B. LDB $\geq$ max(1,N).

X (input/output)
REAL/COMPLEX/COMPLEX array, dimension (LDX,NRHS)
On entry, the solution matrix X, as computed by SSYTRS/CSYTRS/CHETRS.
On exit, the improved solution matrix X.

LDX (input) INTEGER
The leading dimension of the array X. LDX $\geq$ max(1,N).

FERR (output) REAL array, dimension (NRHS)
The estimated forward error bound for each solution vector X(j) (the jth column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) − XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR (output) REAL array, dimension (NRHS)
The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK *SSYRFS* (workspace) REAL array, dimension (3*N)
CSYRFS/CHERFS (workspace) COMPLEX array, dimension (2*N)

IWORK *SSYRFS only* (workspace) INTEGER array, dimension (N)

RWORK *CSYRFS/CHERFS only* (workspace) REAL array, dimension (N)

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SSYSV/CSYSV/CHESV

```
      SUBROUTINE SSYSV( UPLO, N, NRHS, A, LDA, IPIV, B, LDB, WORK,
     $                  LWORK, INFO )
      CHARACTER          UPLO
      INTEGER            INFO, LDA, LDB, LWORK, N, NRHS
      INTEGER            IPIV( * )
      REAL               A( LDA, * ), B( LDB, * ), WORK( LWORK )

      SUBROUTINE CSYSV( UPLO, N, NRHS, A, LDA, IPIV, B, LDB, WORK,
     $                  LWORK, INFO )
      CHARACTER          UPLO
      INTEGER            INFO, LDA, LDB, LWORK, N, NRHS
      INTEGER            IPIV( * )
      COMPLEX            A( LDA, * ), B( LDB, * ), WORK( LWORK )

      SUBROUTINE CHESV( UPLO, N, NRHS, A, LDA, IPIV, B, LDB, WORK,
     $                  LWORK, INFO )
      CHARACTER          UPLO
      INTEGER            INFO, LDA, LDB, LWORK, N, NRHS
      INTEGER            IPIV( * )
      COMPLEX            A( LDA, * ), B( LDB, * ), WORK( LWORK )
```

Purpose

SSYSV/CSYSV/CHESV computes the solution to a real/complex/complex system of linear equations A*X = B, where A is an n-by-n symmetric/symmetric/Hermitian matrix and X and B are n-by-nrhs matrices.

The diagonal pivoting method is used to factor A as

$$A = U*D*U^T \text{ or } A = L*D*L^T \text{ (SSYSV/CSYSV) or}$$

$$A = U*D*U^H \text{ or } A = L*D*L^H \text{ (CHESV),}$$

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and D is symmetric (SSYSV/CSYSV) or Hermitian (CHESV) and block diagonal with 1-by-1 and 2-by-2 diagonal blocks. The factored form of A is then used to solve the system of equations A*X = B.

Arguments

UPLO (input) CHARACTER*1
= 'U': Upper triangle of A is stored;
= 'L': Lower triangle of A is stored.

N (input) INTEGER
The number of linear equations, i.e., the order of the matrix A. N $\geq$ 0.

NRHS (input) INTEGER
The number of right hand sides, i.e., the number of columns of the matrix B. NRHS $\geq$ 0.

A (input/output) REAL/COMPLEX/COMPLEX array, dimension (LDA,N)
On entry, the symmetric/symmetric/Hermitian matrix A. If UPLO = 'U', the leading n-by-n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n-by-n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.
On exit, if INFO = 0, the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization $A = U*D*U^T$ or $A = L*D*L^T$ as computed by SSYTRF/CSYTRF or the factorization $A = U*D*U^H$ or $A = L*D*L^H$ as computed by CHETRF.

LDA (input) INTEGER
The leading dimension of the array A. LDA $\geq$ max(1,N).

IPIV (output) INTEGER array, dimension (N)
Details of the interchanges and the block structure of D, as deter-

mined by SSYTRF/CSYTRF/CHETRF. If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged, and D(k,k) is a 1-by-1 diagonal block. If UPLO = 'U' and IPIV(k) = IPIV(k−1) < 0, then rows and columns k−1 and −IPIV(k) were interchanged and D(k−1:k,k−1:k) is a 2-by-2 diagonal block. If UPLO = 'L' and IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and −IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

B (input/output)
REAL/COMPLEX/COMPLEX array, dimension (LDB,NRHS)
On entry, the n-by-nrhs right hand side matrix B.
On exit, if INFO = 0, the n-by-nrhs solution matrix X.

LDB (input) INTEGER
The leading dimension of the array B. LDB $\geq$ max(1,N).

WORK (workspace/output)
REAL/COMPLEX/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The length of WORK. LWORK $\geq$ 1, and for best performance LWORK $\geq$ N*NB, where NB is the optimal blocksize for SSYTRF/CSYTRF/CHETRF.

If LWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, D(i,i) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, so the solution could not be computed.

SSYSVX/CSYSVX/CHESVX

```
      SUBROUTINE SSYSVX( FACT, UPLO, N, NRHS, A, LDA, AF, LDAF, IPIV, B,
     $                   LDB, X, LDX, RCOND, FERR, BERR, WORK, LWORK,
     $                   IWORK, INFO )
      CHARACTER          FACT, UPLO
      INTEGER            INFO, LDA, LDAF, LDB, LDX, LWORK, N, NRHS
      REAL               RCOND
      INTEGER            IPIV( * ), IWORK( * )
      REAL               A( LDA, * ), AF( LDAF, * ), B( LDB, * ),
     $                   BERR( * ), FERR( * ), WORK( * ), X( LDX, * )
      SUBROUTINE CSYSVX( FACT, UPLO, N, NRHS, A, LDA, AF, LDAF, IPIV, B,
     $                   LDB, X, LDX, RCOND, FERR, BERR, WORK, LWORK,
     $                   RWORK, INFO )
      CHARACTER          FACT, UPLO
      INTEGER            INFO, LDA, LDAF, LDB, LDX, LWORK, N, NRHS
      REAL               RCOND
      INTEGER            IPIV( * )
      REAL               BERR( * ), FERR( * ), RWORK( * )
      COMPLEX            A( LDA, * ), AF( LDAF, * ), B( LDB, * ),
     $                   WORK( * ), X( LDX, * )
      SUBROUTINE CHESVX( FACT, UPLO, N, NRHS, A, LDA, AF, LDAF, IPIV, B,
     $                   LDB, X, LDX, RCOND, FERR, BERR, WORK, LWORK,
     $                   RWORK, INFO )
      CHARACTER          FACT, UPLO
      INTEGER            INFO, LDA, LDAF, LDB, LDX, LWORK, N, NRHS
      REAL               RCOND
      INTEGER            IPIV( * )
      REAL               BERR( * ), FERR( * ), RWORK( * )
      COMPLEX            A( LDA, * ), AF( LDAF, * ), B( LDB, * ),
     $                   WORK( * ), X( LDX, * )
```

Purpose

SSYSVX/CSYSVX/CHESVX uses the diagonal pivoting factorization to compute the solution to a real/complex/complex system of linear equations A*X = B, where A is an n-by-n symmetric (SSYSVX/CSYSVX) or Hermitian (CHESVX) matrix and X and B are n-by-nrhs matrices.

Error bounds on the solution and a condition estimate are also computed.

Description

The following steps are performed:

1. If FACT = 'N', the diagonal pivoting method is used to factor A. The form of the factorization is

   $A = U*D*U^T$ or $A = L*D*L^T$ (SSYSVX/CSYSVX) or

   $A = U*D*U^H$ or $A = L*D*L^H$ (CHESVX),

   where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and D is symmetric (SSYSVX/CSYSVX) or Hermitian (CHESVX) and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

2. If some D(i,i)=0, so that D is exactly singular, then the routine returns with INFO = i. Otherwise, the factored form of A is used to estimate the condition number of the matrix A. If the reciprocal of the condition number is less than machine precision, INFO = N+1 is returned as a warning, but the routine still goes on to solve for X and compute error bounds as described below.

3. The system of equations is solved for X using the factored form of A.

4. Iterative refinement is applied to improve the computed solution matrix and calculate error bounds and backward error estimates for it.

Arguments

FACT (input) CHARACTER*1
Specifies whether or not the factored form of A has been supplied on entry.
= 'F': On entry, AF and IPIV contain the factored form of A. A, AF and IPIV will not be modified.
= 'N': The matrix A will be copied to AF and factored.

UPLO (input) CHARACTER*1
= 'U': Upper triangle of A is stored;
= 'L': Lower triangle of A is stored.

N (input) INTEGER
The number of linear equations, i.e., the order of the matrix A. $N \geq 0$.

NRHS (input) INTEGER
The number of right hand sides, i.e., the number of columns of the matrices B and X. $NRHS \geq 0$.

A (input) REAL/COMPLEX/COMPLEX array, dimension (LDA,N)
The symmetric/symmetric/Hermitian matrix A. If UPLO = 'U', the leading n-by-n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n-by-n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

LDA (input) INTEGER
The leading dimension of the array A. $LDA \geq \max(1,N)$.

AF (input or output)
REAL/COMPLEX/COMPLEX array, dimension (LDAF,N)
If FACT = 'F', then AF is an input argument and on entry contains the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization $A = U*D*U^T$ or $A = L*D*L^T$ as computed by SSYTRF/CSYTRF or the factorization $A = U*D*U^H$ or $A = L*D*L^H$ as computed by CHETRF.
If FACT = 'N', then AF is an output argument and on exit returns the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization.

LDAF (input) INTEGER
The leading dimension of the array AF. $LDAF \geq \max(1,N)$.

IPIV (input or output) INTEGER array, dimension (N)
If FACT = 'F', then IPIV is an input argument and on entry contains details of the interchanges and the block structure of D, as determined by SSYTRF/CSYTRF/CHETRF.
If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.
If UPLO = 'U' and IPIV(k) = IPIV(k−1) < 0, then rows and columns k−1 and −IPIV(k) were interchanged and D(k−1:k,k−1:k) is a 2-by-2 diagonal block. If UPLO = 'L' and IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and −IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.
If FACT = 'N', then IPIV is an output argument and on exit contains details of the interchanges and the block structure of D, as determined by SSYTRF/CSYTRF/CHETRF.

B (input) REAL/COMPLEX/COMPLEX array, dimension (LDB,NRHS)
The n-by-nrhs right hand side matrix B.

LDB (input) INTEGER
The leading dimension of the array B. $LDB \geq \max(1,N)$.

X (output)
REAL/COMPLEX/COMPLEX array, dimension (LDX,NRHS)
If INFO = 0 or INFO = N+1, the N-by-NRHS solution matrix X.

LDX (input) INTEGER
The leading dimension of the array X. $LDX \geq \max(1,N)$.

RCOND (output) REAL
The estimate of the reciprocal condition number of the matrix A. If RCOND is less than the machine precision (in particular, if RCOND = 0), the matrix is singular to working precision. This condition is indicated by a return code of INFO > 0.

FERR (output) REAL array, dimension (NRHS)
The estimated forward error bound for each solution vector X(j) (the jth column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) − XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR (output) REAL array, dimension (NRHS)
The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK (workspace/output) REAL/COMPLEX/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The length of the array WORK.
SSYSVX
$LWORK \geq 3*N$, and for best performance $LWORK \geq N*NB$, where NB is the optimal block size for SSYTRF.
CSYSVX/CHESVX
$LWORK \geq 2*N$, and for best performance $LWORK \geq N*NB$, where

NB is the optimal block size for CSYTRF/CHETRF.

If LWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

IWORK *SSYSVX only* (workspace) INTEGER array, dimension (N)

RWORK *CSYSVX/CHESVX only* (workspace) REAL array, dimension (N)

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, and i is
≤ N: D(i,i) is exactly zero. The factorization has has been completed, but the block diagonal matrix D is exactly singular, so the solution and error bounds could not be computed. RCOND = 0 is returned.
= N+1: D is nonsingular, but RCOND is less than machine precision, meaning that the matrix is singular to working precision. Nevertheless, the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the value of RCOND would suggest.

SSYTRD/CHETRD

```
      SUBROUTINE SSYTRD( UPLO, N, A, LDA, D, E, TAU, WORK, LWORK, INFO )
      CHARACTER          UPLO
      INTEGER            INFO, LDA, LWORK, N
      REAL               A( LDA, * ), D( * ), E( * ), TAU( * ),
     $                   WORK( * )

      SUBROUTINE CHETRD( UPLO, N, A, LDA, D, E, TAU, WORK, LWORK, INFO )
      CHARACTER          UPLO
      INTEGER            INFO, LDA, LWORK, N
      REAL               D( * ), E( * )
      COMPLEX            A( LDA, * ), TAU( * ), WORK( * )
```

Purpose

SSYTRD/CHETRD reduces a real/complex symmetric/Hermitian matrix A to real symmetric tridiagonal form T by an orthogonal/unitary similarity transformation: $Q^H*A*Q = T$.

Arguments

UPLO (input) CHARACTER*1
= 'U': Upper triangle of A is stored;
= 'L': Lower triangle of A is stored.

N (input) INTEGER
The order of the matrix A. N ≥ 0.

A (input/output) REAL/COMPLEX array, dimension (LDA,N)
On entry, the symmetric/Hermitian matrix A. If UPLO = 'U', the leading n-by-n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n-by-n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.
On exit, if UPLO = 'U', the diagonal and first superdiagonal of A are overwritten by the corresponding elements of the tridiagonal matrix T, and the elements above the first superdiagonal, with the array TAU, represent the orthogonal/unitary matrix Q as a product of elementary reflectors; if UPLO = 'L', the diagonal and first subdiagonal of A are over- written by the corresponding elements of the tridiagonal matrix T, and the elements below the first subdiagonal, with the array TAU, represent the orthogonal/unitary matrix Q as a product of elementary reflectors.

LDA (input) INTEGER
The leading dimension of the array A. LDA ≥ max(1,N).

D (output) REAL array, dimension (N)
The diagonal elements of the tridiagonal matrix T: D(i) = A(i,i).

E (output) REAL array, dimension (N−1)
The off-diagonal elements of the tridiagonal matrix T: E(i) = A(i,i+1) if UPLO = 'U', E(i) = A(i+1,i) if UPLO = 'L'.

TAU (output) REAL/COMPLEX array, dimension (N−1)
The scalar factors of the elementary reflectors.

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK. LWORK ≥ 1. For optimum performance LWORK ≥ N*NB, where NB is the optimal blocksize.

If LWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

SSYTRF/CSYTRF/CHETRF

```
SUBROUTINE SSYTRF( UPLO, N, A, LDA, IPIV, WORK, LWORK, INFO )
CHARACTER          UPLO
INTEGER            INFO, LDA, LWORK, N
INTEGER            IPIV( * )
REAL               A( LDA, * ), WORK( LWORK )

SUBROUTINE CSYTRF( UPLO, N, A, LDA, IPIV, WORK, LWORK, INFO )
CHARACTER          UPLO
INTEGER            INFO, LDA, LWORK, N
INTEGER            IPIV( * )
COMPLEX            A( LDA, * ), WORK( LWORK )

SUBROUTINE CHETRF( UPLO, N, A, LDA, IPIV, WORK, LWORK, INFO )
CHARACTER          UPLO
INTEGER            INFO, LDA, LWORK, N
INTEGER            IPIV( * )
COMPLEX            A( LDA, * ), WORK( LWORK )
```

Purpose

SSYTRF/CSYTRF/CHETRF computes the factorization of a real/complex/complex symmetric/symmetric/Hermitian matrix A using the diagonal pivoting method. The form of the factorization is

$A = U*D*U^T$ or $A = L*D*L^T$ (SSYTRF/CSYTRF) or

$A = U*D*U^H$ or $A = L*D*L^H$ (CHETRF),

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and D is symmetric (SSYTRF/CSYTRF) or Hermitian (CHETRF) and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

Arguments

UPLO (input) CHARACTER*1
= 'U': Upper triangle of A is stored;
= 'L': Lower triangle of A is stored.

N (input) INTEGER
The order of the matrix A. $N \geq 0$.

A (input/output)
REAL/COMPLEX/COMPLEX array, dimension (LDA,N)
On entry, the symmetric/symmetric/Hermitian matrix A. If UPLO = 'U', the leading n-by-n upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n-by-n lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.
On exit, the block diagonal matrix D and the multipliers used to obtain the factor U or L.

LDA (input) INTEGER
The leading dimension of the array A. $LDA \geq \max(1,N)$.

IPIV (output) INTEGER array, dimension (N)
Details of the interchanges and the block structure of D.
If IPIV(k) > 0, then rows and columns k and IPIV(k) were interchanged and D(k,k) is a 1-by-1 diagonal block.
If UPLO = 'U' and IPIV(k) = IPIV(k−1) < 0, then rows and columns k−1 and −IPIV(k) were interchanged and D(k−1:k,k−1:k) is a 2-by-2 diagonal block. If UPLO = 'L' and IPIV(k) = IPIV(k+1) < 0, then rows and columns k+1 and −IPIV(k) were interchanged and D(k:k+1,k:k+1) is a 2-by-2 diagonal block.

WORK (workspace/output)
REAL/COMPLEX/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The length of WORK. $LWORK \geq 1$. For best performance $LWORK \geq N*NB$, where NB is the block size returned by ILAENV.

If LWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, D(i,i) is exactly zero. The factorization has been completed, but the block diagonal matrix D is exactly singular, and division by zero will occur if it is used to solve a system of equations.

SSYTRI/CSYTRI/CHETRI

```
SUBROUTINE SSYTRI( UPLO, N, A, LDA, IPIV, WORK, INFO )
CHARACTER          UPLO
INTEGER            INFO, LDA, N
INTEGER            IPIV( * )
REAL               A( LDA, * ), WORK( * )

SUBROUTINE CSYTRI( UPLO, N, A, LDA, IPIV, WORK, INFO )
CHARACTER          UPLO
INTEGER            INFO, LDA, N
INTEGER            IPIV( * )
COMPLEX            A( LDA, * ), WORK( * )
```

```
SUBROUTINE CHETRI( UPLO, N, A, LDA, IPIV, WORK, INFO )
CHARACTER          UPLO
INTEGER            INFO, LDA, N
INTEGER            IPIV( * )
COMPLEX            A( LDA, * ), WORK( * )
```

Purpose

SSYTRI/CSYTRI/CHETRI computes the inverse of a real/complex/complex symmetric/symmetric/Hermitian indefinite matrix A using the factorization $A = U*D*U^T$ or $A = L*D*L^T$ computed by SSYTRF/CSYTRF or the factorization $A = U*D*U^H$ or $A = L*D*L^H$ computed by CHETRF.

Arguments

UPLO (input) CHARACTER*1
Specifies whether the details of the factorization are stored as an upper or lower triangular matrix.
= 'U': Upper triangular, form is $A = U*D*U^T$ (SSYTRI/CSYTRI) or $A = U*D*U^H$ (CHETRI);
= 'L': Lower triangular, form is $A = L*D*L^T$ (SSYTRI/CSYTRI) or $A = L*D*L^H$ (CHETRI)

N (input) INTEGER
The order of the matrix A. $N \geq 0$.

A (input/output)
REAL/COMPLEX/COMPLEX array, dimension (LDA,N)
On entry, the block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by SSYTRF/CSYTRF/CHETRF.
On exit, if INFO = 0, the (symmetric/symmetric/Hermitian) inverse of the original matrix. If UPLO = 'U', the upper triangular part of the inverse is formed and the part of A below the diagonal is not referenced; if UPLO = 'L' the lower triangular part of the inverse is formed and the part of A above the diagonal is not referenced.

LDA (input) INTEGER
The leading dimension of the array A. $LDA \geq \max(1,N)$.

IPIV (input) INTEGER array, dimension (N)
Details of the interchanges and the block structure of D as determined by SSYTRF/CSYTRF/CHETRF.

WORK *SSYTRI* (workspace) REAL array, dimension (N)
CSYTRI/CHETRI (workspace) COMPLEX array, dimension (2*N)

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, D(i,i) = 0; the matrix is singular and its inverse could not be computed.

SSYTRS/CSYTRS/CHETRS

```
SUBROUTINE SSYTRS( UPLO, N, NRHS, A, LDA, IPIV, B, LDB, INFO )
CHARACTER          UPLO
INTEGER            INFO, LDA, LDB, N, NRHS
INTEGER            IPIV( * )
REAL               A( LDA, * ), B( LDB, * )

SUBROUTINE CSYTRS( UPLO, N, NRHS, A, LDA, IPIV, B, LDB, INFO )
CHARACTER          UPLO
INTEGER            INFO, LDA, LDB, N, NRHS
INTEGER            IPIV( * )
COMPLEX            A( LDA, * ), B( LDB, * )

SUBROUTINE CHETRS( UPLO, N, NRHS, A, LDA, IPIV, B, LDB, INFO )
CHARACTER          UPLO
INTEGER            INFO, LDA, LDB, N, NRHS
INTEGER            IPIV( * )
COMPLEX            A( LDA, * ), B( LDB, * )
```

Purpose

SSYTRS/CSYTRS/CHETRS solves a system of linear equations $A*X = B$ with a real/complex/complex symmetric/symmetric/Hermitian matrix A using the factorization $A = U*D*U^T$ or $A = L*D*L^T$ computed by SSYTRF/CSYTRF or the factorization $A = U*D*U^H$ or $A = L*D*L^H$ computed by CHETRF.

Arguments

UPLO (input) CHARACTER*1
Specifies whether the details of the factorization are stored as an upper or lower triangular matrix.
= 'U': Upper triangular, form is $A = U*D*U^T$ (SSYTRS/CSYTRS) or $A = U*D*U^H$ (CHETRS);
= 'L': Lower triangular, form is $A = L*D*L^T$ (SSYTRS/CSYTRS) or $A = L*D*L^H$ (CHETRS).

N (input) INTEGER
The order of the matrix A. $N \geq 0$.

NRHS (input) INTEGER
The number of right hand sides, i.e., the number of columns of the matrix B. $NRHS \geq 0$.

A (input) REAL/COMPLEX/COMPLEX array, dimension (LDA,N)
The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by SSYTRF/CSYTRF/CHETRF.

LDA (input) INTEGER
The leading dimension of the array A. $LDA \geq \max(1,N)$.

IPIV (input) INTEGER array, dimension (N)
Details of the interchanges and the block structure of D as determined by SSYTRF/CSYTRF/CHETRF.

B (input/output)
REAL/COMPLEX/COMPLEX array, dimension (LDB,NRHS)
On entry, the right hand side matrix B.
On exit, the solution matrix X.

LDB (input) INTEGER
The leading dimension of the array B. LDB $\geq$ max(1,N).

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

STBCON/CTBCON

```
      SUBROUTINE STBCON( NORM, UPLO, DIAG, N, KD, AB, LDAB, RCOND, WORK,
     $                   IWORK, INFO )
      CHARACTER          DIAG, NORM, UPLO
      INTEGER            INFO, KD, LDAB, N
      REAL               RCOND
      INTEGER            IWORK( * )
      REAL               AB( LDAB, * ), WORK( * )

      SUBROUTINE CTBCON( NORM, UPLO, DIAG, N, KD, AB, LDAB, RCOND, WORK,
     $                   RWORK, INFO )
      CHARACTER          DIAG, NORM, UPLO
      INTEGER            INFO, KD, LDAB, N
      REAL               RCOND
      REAL               RWORK( * )
      COMPLEX            AB( LDAB, * ), WORK( * )
```

Purpose

STBCON/CTBCON estimates the reciprocal of the condition number of a triangular band matrix A, in either the 1-norm or the infinity-norm.

The norm of A is computed and an estimate is obtained for $\|A^{-1}\|$, then the reciprocal of the condition number is computed as RCOND = $1/(\|A\| * \|A^{-1}\|)$.

Arguments

NORM (input) CHARACTER*1
Specifies whether the 1-norm condition number or the infinity-norm condition number is required:
= '1' or 'O': 1-norm;
= 'I': Infinity-norm.

UPLO (input) CHARACTER*1
= 'U': A is upper triangular;
= 'L': A is lower triangular.

DIAG (input) CHARACTER*1
= 'N': A is non-unit triangular;
= 'U': A is unit triangular.

N (input) INTEGER
The order of the matrix A. N $\geq$ 0.

KD (input) INTEGER
The number of superdiagonals or subdiagonals of the triangular band matrix A. KD $\geq$ 0.

AB (input) REAL/COMPLEX array, dimension (LDAB,N)
The upper or lower triangular band matrix A, stored in the first kd+1 rows of the array. The jth column of A is stored in the jth column of the array AB as follows:
if UPLO = 'U', AB(kd+1+i−j,j) = A(i,j) for max(1,j−kd)$\leq$i$\leq$j;
if UPLO = 'L', AB(1+i−j,j) = A(i,j) for j$\leq$i$\leq$min(n,j+kd).
If DIAG = 'U', the diagonal elements of A are not referenced and are assumed to be 1.

LDAB (input) INTEGER
The leading dimension of the array AB. LDAB $\geq$ KD+1.

RCOND (output) REAL
The reciprocal of the condition number of the matrix A, computed as RCOND = $1/(\|A\| * \|A^{-1}\|)$.

WORK *STBCON* (workspace) REAL array, dimension (3*N)
CTBCON (workspace) COMPLEX array, dimension (2*N)

IWORK *STBCON only* (workspace) INTEGER array, dimension (N)

RWORK *CTBCON only* (workspace) REAL array, dimension (N)

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

STBRFS/CTBRFS

```
      SUBROUTINE STBRFS( UPLO, TRANS, DIAG, N, KD, NRHS, AB, LDAB, B,
     $                   LDB, X, LDX, FERR, BERR, WORK, IWORK, INFO )
      CHARACTER          DIAG, TRANS, UPLO
      INTEGER            INFO, KD, LDAB, LDB, LDX, N, NRHS
      INTEGER            IWORK( * )
      REAL               AB( LDAB, * ), B( LDB, * ), BERR( * ),
     $                   FERR( * ), WORK( * ), X( LDX, * )
```

```
      SUBROUTINE CTBRFS( UPLO, TRANS, DIAG, N, KD, NRHS, AB, LDAB, B,
     $                   LDB, X, LDX, FERR, BERR, WORK, RWORK, INFO )
      CHARACTER          DIAG, TRANS, UPLO
      INTEGER            INFO, KD, LDAB, LDB, LDX, N, NRHS
      REAL               BERR( * ), FERR( * ), RWORK( * )
      COMPLEX            AB( LDAB, * ), B( LDB, * ), WORK( * ),
     $                   X( LDX, * )
```

Purpose

STBRFS/CTBRFS provides error bounds and backward error estimates for the solution to a system of linear equations with a triangular band coefficient matrix.

The solution matrix X must be computed by STBTRS/CTBTRS or some other means before entering this routine. STBRFS/CTBRFS does not do iterative refinement because doing so cannot improve the backward error.

Arguments

UPLO (input) CHARACTER*1
= 'U': A is upper triangular;
= 'L': A is lower triangular.

TRANS (input) CHARACTER*1
Specifies the form of the system of equations:
= 'N': A*X = B (No transpose)
= 'T': A^T*X = B (Transpose)
= 'C': A^H*X = B (Conjugate transpose)

DIAG (input) CHARACTER*1
= 'N': A is non-unit triangular;
= 'U': A is unit triangular.

N (input) INTEGER
The order of the matrix A. N $\geq$ 0.

KD (input) INTEGER
The number of superdiagonals or subdiagonals of the triangular band matrix A. KD $\geq$ 0.

NRHS (input) INTEGER
The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS $\geq$ 0.

AB (input) REAL/COMPLEX array, dimension (LDAB,N)
The upper or lower triangular band matrix A, stored in the first kd+1 rows of the array. The jth column of A is stored in the jth column of the array AB as follows:
if UPLO = 'U', AB(kd+1+i−j,j) = A(i,j) for max(1,j−kd)$\leq$i$\leq$j;
if UPLO = 'L', AB(1+i−j,j) = A(i,j) for j$\leq$i$\leq$min(n,j+kd).
If DIAG = 'U', the diagonal elements of A are not referenced and are assumed to be 1.

LDAB (input) INTEGER
The leading dimension of the array AB. LDAB $\geq$ KD+1.

B (input) REAL/COMPLEX array, dimension (LDB,NRHS)
The right hand side matrix B.

LDB (input) INTEGER
The leading dimension of the array B. LDB $\geq$ max(1,N).

X (input) REAL/COMPLEX array, dimension (LDX,NRHS)
The solution matrix X.

LDX (input) INTEGER
The leading dimension of the array X. LDX $\geq$ max(1,N).

FERR (output) REAL array, dimension (NRHS)
The estimated forward error bound for each solution vector X(j) (the jth column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) − XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR (output) REAL array, dimension (NRHS)
The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK *STBRFS* (workspace) REAL array, dimension (3*N)
CTBRFS (workspace) COMPLEX array, dimension (2*N)

IWORK *STBRFS only* (workspace) INTEGER array, dimension (N)

RWORK *CTBRFS only* (workspace) REAL array, dimension (N)

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

STBTRS/CTBTRS

```
      SUBROUTINE STBTRS( UPLO, TRANS, DIAG, N, KD, NRHS, AB, LDAB, B,
     $                   LDB, INFO )
      CHARACTER          DIAG, TRANS, UPLO
      INTEGER            INFO, KD, LDAB, LDB, N, NRHS
      REAL               AB( LDAB, * ), B( LDB, * )

      SUBROUTINE CTBTRS( UPLO, TRANS, DIAG, N, KD, NRHS, AB, LDAB, B,
     $                   LDB, INFO )
      CHARACTER          DIAG, TRANS, UPLO
      INTEGER            INFO, KD, LDAB, LDB, N, NRHS
      COMPLEX            AB( LDAB, * ), B( LDB, * )
```

Purpose

STBTRS/CTBTRS solves a triangular system of the form A*X = B, A^T*X = B, or A^H*X = B, where A is a triangular band matrix of order n, and B is an n-by-nrhs matrix. A check is made to verify that A is nonsingular.

Arguments

UPLO (input) CHARACTER*1
= 'U': A is upper triangular;
= 'L': A is lower triangular.

TRANS (input) CHARACTER*1
Specifies the form of the system of equations:
= 'N': A*X = B (No transpose)
= 'T': A^T*X = B (Transpose)
= 'C': A^H*X = B (Conjugate transpose)

DIAG (input) CHARACTER*1
= 'N': A is non-unit triangular;
= 'U': A is unit triangular.

N (input) INTEGER
The order of the matrix A. N $\geq$ 0.

KD (input) INTEGER
The number of superdiagonals or subdiagonals of the triangular band matrix A. KD $\geq$ 0.

NRHS (input) INTEGER
The number of right hand sides, i.e., the number of columns of the matrix B. NRHS $\geq$ 0.

AB (input) REAL/COMPLEX array, dimension (LDAB,N)
The upper or lower triangular band matrix A, stored in the first kd+1 rows of AB. The jth column of A is stored in the jth column of the array AB as follows:
if UPLO = 'U', AB(kd+1+i−j,j) = A(i,j) for max(1,j−kd)$\leq$i$\leq$j;
if UPLO = 'L', AB(1+i−j,j) = A(i,j) for j$\leq$i$\leq$min(n,j+kd).
If DIAG = 'U', the diagonal elements of A are not referenced and are assumed to be 1.

LDAB (input) INTEGER
The leading dimension of the array AB. LDAB $\geq$ KD+1.

B (input/output) REAL/COMPLEX array, dimension (LDB,NRHS)
On entry, the right hand side matrix B.
On exit, if INFO = 0, the solution matrix X.

LDB (input) INTEGER
The leading dimension of the array B. LDB $\geq$ max(1,N).

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, the ith diagonal element of A is zero, indicating that the matrix is singular and the solutions X have not been computed.

STGEVC/CTGEVC

```
      SUBROUTINE STGEVC( SIDE, HOWMNY, SELECT, N, A, LDA, B, LDB, VL,
     $                   LDVL, VR, LDVR, MM, M, WORK, INFO )
      CHARACTER          HOWMNY, SIDE
      INTEGER            INFO, LDA, LDB, LDVL, LDVR, M, MM, N
      LOGICAL            SELECT( * )
      REAL               A( LDA, * ), B( LDB, * ), VL( LDVL, * ),
     $                   VR( LDVR, * ), WORK( N, * )

      SUBROUTINE CTGEVC( SIDE, HOWMNY, SELECT, N, A, LDA, B, LDB, VL,
     $                   LDVL, VR, LDVR, MM, M, WORK, RWORK, INFO )
      CHARACTER          HOWMNY, SIDE
      INTEGER            INFO, LDA, LDB, LDVL, LDVR, M, MM, N
      LOGICAL            SELECT( * )
      REAL               RWORK( * )
      COMPLEX            A( LDA, * ), B( LDB, * ), VL( LDVL, * ),
     $                   VR( LDVR, * ), WORK( * )
```

Purpose

STGEVC/CTGEVC computes some or all of the right and/or left generalized eigenvectors of a pair of real/complex upper triangular matrices (A,B).

The right generalized eigenvector x and the left generalized eigenvector y of (A,B) corresponding to a generalized eigenvalue w are defined by:

$$(A - w*B)*x = 0 \quad \text{and} \quad y^H(A - w*B) = 0$$

If an eigenvalue w is determined by zero diagonal elements of both A and B, a unit vector is returned as the corresponding eigenvector.

If all eigenvectors are requested, the routine may either return the matrices X and/or Y of right or left eigenvectors of (A,B), or the products Z*X and/or Q*Y, where Z and Q are input orthogonal/unitary matrices. If (A,B) was obtained from the generalized real-Schur/Schur factorization of an original pair of matrices $(A_0,B_0) = (Q*A*Z^H, Q*B*Z^H)$, then Z*X and Q*Y are the matrices of right or left eigenvectors of A.

STGEVC only
A must be block upper triangular, with 1-by-1 and 2-by-2 diagonal blocks. Corresponding to each 2-by-2 diagonal block is a complex conjugate pair of eigenvalues and eigenvectors; only one eigenvector of the pair is computed, namely the one corresponding to the eigenvalue with positive imaginary part.

Arguments

SIDE (input) CHARACTER*1
= 'R': compute right eigenvectors only;
= 'L': compute left eigenvectors only;
= 'B': compute both right and left eigenvectors.

HOWMNY (input) CHARACTER*1
= 'A': compute all right and/or left eigenvectors;
= 'B': compute all right and/or left eigenvectors, and backtransform them using the input matrices supplied in VR and/or VL;
= 'S': compute selected right and/or left eigenvectors, specified by the logical array SELECT.

SELECT (input) LOGICAL array, dimension (N)
If HOWMNY='S', SELECT specifies the eigenvectors to be computed.
If HOWMNY='A' or 'B', SELECT is not referenced.
STGEVC
To select the real eigenvector corresponding to a real eigenvalue w(j), SELECT(j) must be set to .TRUE.. To select the complex eigenvector corresponding to a complex conjugate pair w(j) and w(j+1), either SELECT(j) or SELECT(j+1) must be set to .TRUE..
CTGEVC
To select the eigenvector corresponding to the jth eigenvalue, SELECT(j) must be set to .TRUE..

N (input) INTEGER
The order of the matrices A and B. N $\geq$ 0.

A (input) REAL/COMPLEX array, dimension (LDA,N)
STGEVC
The upper quasi-triangular matrix A.
CTGEVC
The upper triangular matrix A.

LDA (input) INTEGER
The leading dimension of array A. LDA $\geq$ max(1,N).

B (input) REAL/COMPLEX array, dimension (LDB,N)
The upper triangular matrix B.
STGEVC
If A has a 2-by-2 diagonal block, then the corresponding 2-by-2 block of B must be diagonal with positive elements.
CTGEVC
B must have real diagonal elements.

LDB (input) INTEGER
The leading dimension of array B. LDB $\geq$ max(1,N).

VL (input/output) REAL/COMPLEX array, dimension (LDVL,MM)
On entry, if SIDE = 'L' or 'B' and HOWMNY = 'B', VL must contain an n-by-n matrix Q (usually the orthogonal/unitary matrix Q of left Schur vectors returned by SHGEQZ/CHGEQZ).
On exit, if SIDE = 'L' or 'B', VL contains:
if HOWMNY = 'A', the matrix Y of left eigenvectors of (A,B);
if HOWMNY = 'B', the matrix Q*Y;
if HOWMNY = 'S', the left eigenvectors of (A,B) specified by SELECT, stored consecutively in the columns of VL, in the same order as their eigenvalues.
If SIDE = 'R', VL is not referenced.
STGEVC only
A complex eigenvector corresponding to a complex eigenvalue is stored in two consecutive columns, the first holding the real part, and the second the imaginary part.

LDVL (input) INTEGER
The leading dimension of array VL.
LDVL $\geq$ max(1,N) if SIDE = 'L' or 'B'; LDVL $\geq$ 1 otherwise.

VR (input/output) REAL/COMPLEX array, dimension (LDVR,MM)
On entry, if SIDE = 'R' or 'B' and HOWMNY = 'B', VR must contain an n-by-n matrix Q (usually the orthogonal/unitary matrix Z of right Schur vectors returned by SHGEQZ/CHGEQZ).
On exit, if SIDE = 'R' or 'B', VR contains:
if HOWMNY = 'A', the matrix X of right eigenvectors of (A,B);
if HOWMNY = 'B', the matrix Z*X;
if HOWMNY = 'S', the right eigenvectors of (A,B) specified by SELECT, stored consecutively in the columns of VR, in the same order as their eigenvalues.
If SIDE = 'L', VR is not referenced.
STGEVC only
A complex eigenvector corresponding to a complex eigenvalue is stored in two consecutive columns, the first holding the real part and the second the imaginary part.

LDVR (input) INTEGER
The leading dimension of the array VR.
LDVR $\geq$ max(1,N) if SIDE = 'R' or 'B'; LDVR $\geq$ 1 otherwise.

MM (input) INTEGER
The number of columns in the arrays VL and/or VR. MM $\geq$ M.

M (output) INTEGER
The number of columns in the arrays VL and/or VR actually used to store the eigenvectors. If HOWMNY = 'A' or 'B', M is set to N.
STGEVC
Each selected real eigenvector occupies one column and each selected complex eigenvector occupies two columns.
CTGEVC
Each selected eigenvector occupies one column.

WORK *STGEVC* (workspace) REAL array, dimension (6*N)
CTGEVC (workspace) COMPLEX array, dimension (2*N)

RWORK *CTGEVC only* (workspace) REAL array, dimension (2*N)

INFO (output) INTEGER

= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: (*STGEVC only*) the 2-by-2 block (INFO:INFO+1) does not have a complex eigenvalue.

STGEXC/CTGEXC

```
      SUBROUTINE STGEXC( WANTQ, WANTZ, N, A, LDA, B, LDB, Q, LDQ, Z,
     $                   LDZ, IFST, ILST, WORK, LWORK, INFO )
      LOGICAL            WANTQ, WANTZ
      INTEGER            IFST, ILST, INFO, LDA, LDB, LDQ, LDZ, LWORK, N
      REAL               A( LDA, * ), B( LDB, * ), Q( LDQ, * ),
     $                   WORK( * ), Z( LDZ, * )

      SUBROUTINE CTGEXC( WANTQ, WANTZ, N, A, LDA, B, LDB, Q, LDQ, Z,
     $                   LDZ, IFST, ILST, INFO )
      LOGICAL            WANTQ, WANTZ
      INTEGER            IFST, ILST, INFO, LDA, LDB, LDQ, LDZ, N
      COMPLEX            A( LDA, * ), B( LDB, * ), Q( LDQ, * ),
     $                   Z( LDZ, * )
```

Purpose

STGEXC/CTGEXC reorders the generalized real-Schur/Schur decomposition of a real/complex matrix pair (A,B) using an orthogonal/unitary equivalence transformation

$$(A,B) \quad = \quad Q * (A,B) * Z',$$

so that the diagonal block of (A, B) with row index IFST is moved to row ILST.

(A, B) must be in generalized real-Schur/Schur canonical form (as returned by SGGES/CGGES), i.e. A is block upper triangular with 1-by-1 and 2-by-2 diagonal blocks. B is upper triangular.

Optionally, the matrices Q and Z of generalized Schur vectors are updated.

$$\begin{aligned} Q(in) * A(in) * Z(in)' &= Q(out) * A(out) * Z(out)' \\ Q(in) * B(in) * Z(in)' &= Q(out) * B(out) * Z(out)' \end{aligned}$$

Arguments

WANTQ (input) LOGICAL
= .TRUE.: update the left transformation matrix Q;
= .FALSE.: do not update Q.

WANTZ (input) LOGICAL
= .TRUE.: update the right transformation matrix Z;
= .FALSE.: do not update Z.

N (input) INTEGER
The order of the matrices A and B. N $\geq$ 0.

A (input/output) REAL/COMPLEX array, dimension (LDA,N)
STGEXC
On entry, the matrix A in generalized real Schur canonical form.
On exit, the updated matrix A, again in generalized real Schur canonical form.
CTGEXC
On entry, the upper triangular matrix A in the pair (A, B).
On exit, th : updated matrix A.

LDA (input) INTEGER
The leading dimension of array A. LDA $\geq$ max(1,N).

B (input) REAL/COMPLEX array, dimension (LDB,N)
STGEXC
On entry, the matrix B in generalized real Schur canonical form (A,B).
On exit, the updated matrix B, again in generalized real Schur canonical form (A,B).
CTGEXC
On entry, the upper triangular matrix B in the pair (A, B). On exit, the updated matrix B.

LDB (input) INTEGER
The leading dimension of array B. LDB $\geq$ max(1,N).

Q (input/output) REAL/COMPLEX array, dimension (LDZ,N)
On entry, if WANTQ = .TRUE., the orthogonal/unitary matrix Q.
On exit, the updated matrix Q.
If WANTQ = .FALSE., Q is not referenced.

LDQ (input) INTEGER
The leading dimension of the array Q. LDQ $\geq$ 1.
If WANTQ = .TRUE., LDQ $\geq$ N.

Z (input/output) REAL/COMPLEX array, dimension (LDZ,N)
On entry, if WANTZ = .TRUE., the orthogonal/unitary matrix Z.
On exit, the updated matrix Z.
If WANTZ = .FALSE., Z is not referenced.

LDZ (input) INTEGER
The leading dimension of the array Z. LDZ $\geq$ 1.
If WANTZ = .TRUE., LDZ $\geq$ N.

IFST (input/output) INTEGER

ILST (input/output) INTEGER
Specify the reordering of the diagonal blocks of (A, B). The block with row index IFST is moved to row ILST, by a sequence of swapping between adjacent blocks.
STGEXC

On exit, if IFST pointed on entry to the second row of a 2-by-2 block, it is changed to point to the first row; ILST always points to the first row of the block in its final position (which may differ from its input value by +1 or -1). $1 \leq$ IFST, ILST $\leq$ N.

WORK *STGEXC only* (workspace) REAL array, dimension (LWORK)

LWORK *STGEXC only* (input) INTEGER
The dimension of the array WORK. LWORK $\geq$ 4*N + 16.

If LWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
= 1: The transformed matrix pair (A, B) would be too far from generalized Schur form; the problem is ill- conditioned. (A, B) may have been partially reordered, and ILST points to the first row of the current position of the block being moved.

STGSEN/CTGSEN

```
      SUBROUTINE STGSEN( IJOB, WANTQ, WANTZ, SELECT, N, A, LDA, B, LDB,
     $                   ALPHAR, ALPHAI, BETA, Q, LDQ, Z, LDZ, M, PL,
     $                   PR, DIF, WORK, LWORK, IWORK, LIWORK, INFO )
      LOGICAL            WANTQ, WANTZ
      INTEGER            IJOB, INFO, LDA, LDB, LDQ, LDZ, LIWORK, LWORK,
     $                   M, N
      REAL               PL, PR
      LOGICAL            SELECT( * )
      INTEGER            IWORK( * )
      REAL               A( LDA, * ), ALPHAI( * ), ALPHAR( * ),
     $                   B( LDB, * ), BETA( * ), DIF( * ), Q( LDQ, * ),
     $                   WORK( * ), Z( LDZ, * )
```

```
      SUBROUTINE CTGSEN( IJOB, WANTQ, WANTZ, SELECT, N, A, LDA, B, LDB,
     $                   ALPHA, BETA, Q, LDQ, Z, LDZ, M, PL, PR, DIF,
     $                   WORK, LWORK, IWORK, LIWORK, INFO )
      LOGICAL            WANTQ, WANTZ
      INTEGER            IJOB, INFO, LDA, LDB, LDQ, LDZ, LIWORK, LWORK,
     $                   M, N
      REAL               PL, PR
      LOGICAL            SELECT( * )
      INTEGER            IWORK( * )
      REAL               DIF( * )
      COMPLEX            A( LDA, * ), ALPHA( * ), B( LDB, * ),
     $                   BETA( * ), Q( LDQ, * ), WORK( * ), Z( LDZ, * )
```

Purpose

STGSEN/CTGSEN reorders the generalized real-Schur/Schur decomposition of a real/complex matrix pair (A, B) (in terms of an orthogonal/unitary equivalence transformation Q' * (A, B) * Z), so that a selected cluster of eigenvalues appears in the leading diagonal blocks of the pair (A,B). The leading columns of Q and Z form orthonormal/unitary bases of the corresponding left and right eigenspaces (deflating subspaces). (A, B) must be in generalized real-Schur/Schur canonical form (as returned by SGGES/CGGES), that is, A and B are both upper triangular.

STGSEN/CTGSEN also computes the generalized eigenvalues

$$w(j) = (ALPHAR(j) + i * ALPHAI(j))/BETA(j)(STGSEN)$$
$$w(j) = ALPHA(j)/BETA(j)(CTGSEN)$$

of the reordered matrix pair (A, B).

Optionally, the routine computes estimates of reciprocal condition numbers for eigenvalues and eigenspaces. These are Difu[(A11,B11), (A22,B22)] and Difl[(A11,B11), (A22,B22)], i.e. the separation(s) between the matrix pairs (A11, B11) and (A22,B22) that correspond to the selected cluster and the eigenvalues outside the cluster, resp., and norms of "projections" onto left and right eigenspaces w.r.t. the selected cluster in the (1,1)-block.

Arguments

IJOB (input) INTEGER
Specifies whether condition numbers are required for the cluster of eigenvalues (PL and PR) or the deflating subspaces (Difu and Difl):
= 0: Only reorder w.r.t. SELECT. No extras.
= 1: Reciprocal of norms of "projections" onto left and right eigenspaces w.r.t. the selected cluster (PL and PR).
= 2: Upper bounds on Difu and Difl. F-norm-based estimate (DIF(1:2)).
= 3: Estimate of Difu and Difl. 1-norm-based estimate (DIF(1:2)). About 5 times as expensive as IJOB = 2.
= 4: Compute PL, PR and DIF (i.e. 0, 1 and 2 above): Economic version to get it all.

= 5: Compute PL, PR and DIF (i.e. 0, 1 and 3 above)

WANTQ (input) LOGICAL
= .TRUE.: update the left transformation matrix Q;
= .FALSE.: do not update Q.

WANTZ (input) LOGICAL
= .TRUE.: update the right transformation matrix Z;
= .FALSE.: do not update Z.

SELECT (input) LOGICAL array, dimension (N)
STGSEN
SELECT specifies the eigenvalues in the selected cluster. To select a real eigenvalue w(j), SELECT(j) must be set to .TRUE.. To select a complex conjugate pair of eigenvalues w(j) and w(j+1), corresponding to a 2-by-2 diagonal block, either SELECT(j) or SELECT(j+1) or both must be set to .TRUE.; a complex conjugate pair of eigenvalues must be either both included in the cluster or both excluded.
CTGSEN
SELECT specifies the eigenvalues in the selected cluster. To select an eigenvalue w(j), SELECT(j) must be set to .TRUE..

N (input) INTEGER
The order of the matrices A and B. N $\geq$ 0.

A (input/output) REAL/COMPLEX array, dimension (LDA,N)
STGSEN
On entry, the upper quasi-triangular matrix A, with (A, B) in generalized real Schur canonical form.
On exit, A is overwritten by the reordered matrix A.
CTGSEN
On entry, the upper triangular matrix A, in generalized Schur canonical form.
On exit, A is overwritten by the reordered matrix A.

LDA (input) INTEGER
The leading dimension of the array A. LDA $\geq$ max(1,N).

B (input/output) REAL/COMPLEX array, dimension (LDB,N)
STGSEN
On entry, the upper triangular matrix B, with (A, B) in generalized real Schur canonical form.
On exit, B is overwritten by the reordered matrix B.
CTGSEN
On entry, the upper triangular matrix B, in generalized Schur canonical form.
On exit, B is overwritten by the reordered matrix B.

LDB (input) INTEGER
The leading dimension of the array B. LDB $\geq$ max(1,N).

ALPHAR *STGSEN only* (output) REAL array, dimension (N)

ALPHAI *STGSEN only* (output) REAL array, dimension (N)

ALPHA *CTGSEN only* (output) COMPLEX array, dimension (N)

BETA (output) REAL/COMPLEX array, dimension (N)
STGSEN
On exit, (ALPHAR(j) + ALPHAI(j)*i)/BETA(j), j=1,...,N, will be the generalized eigenvalues. ALPHAR(j) + ALPHAI(j)*i and BETA(j),j=1,...,N are the diagonals of the complex Schur form (S,T) that would result if the 2-by-2 diagonal blocks of the real generalized Schur form of (A,B) were further reduced to triangular form using complex unitary transformations. If ALPHAI(j) is zero, then the jth eigenvalue is real; if positive, then the jth and $(j+1)^{st}$ eigenvalues are a complex conjugate pair, with ALPHAI(j+1) negative.
CTGSEN
The diagonal elements of A and B, respectively, when the pair (A,B) has been reduced to generalized Schur form. ALPHA(i)/BETA(i) i=1,...,N are the generalized eigenvalues.

Q (input/output) REAL/COMPLEX array, dimension (LDQ,N)
On entry, if WANTQ = .TRUE., Q is an N-by-N matrix. On exit, Q has been postmultiplied by the left orthogonal/unitary transformation matrix which reorder (A, B); The leading M columns of Q form orthonormal/unitary bases for the specified pair of left eigenspaces (deflating subspaces).
If WANTQ = .FALSE., Q is not referenced.

LDQ (input) INTEGER
The leading dimension of the array Q. LDQ $\geq$ 1;
and if WANTQ = .TRUE., LDQ $\geq$ N.

Z (input/output) REAL/COMPLEX array, dimension (LDZ,N)
On entry, if WANTZ = .TRUE., Z is an N-by-N matrix.
On exit, Z has been postmultiplied by the left orthogonal/unitary transformation matrix which reorder (A, B); The leading M columns of Z form orthonormal/unitary bases for the specified pair of left eigenspaces (deflating subspaces).
If WANTZ = .FALSE., Z is not referenced.

LDZ (input) INTEGER
The leading dimension of the array Z. LDZ $\geq$ 1;
If WANTZ = .TRUE., LDZ $\geq$ N.

M (output) INTEGER
The dimension of the specified pair of left and right eigen- spaces (deflating subspaces). 0 $\leq$ M $\leq$ N.

PL, PR (output) REAL
If IJOB = 1, 4 or 5, PL, PR are lower bounds on the reciprocal of the norm of "projections" onto left and right eigenspaces with respect to the selected cluster. 0 < PL, PR $\leq$ 1. If M = 0 or M = N, PL = PR = 1. If IJOB = 0, 2 or 3, PL and PR are not referenced.

DIF (output) REAL array, dimension (2)
If IJOB $\geq$ 2, DIF(1:2) store the estimates of Difu and Difl. If IJOB =

2 or 4, DIF(1:2) are F-norm-based upper bounds on Difu and Difl. If IJOB = 3 or 5, DIF(1:2) are 1-norm-based estimates of Difu and Difl. If M = 0 or N, DIF(1:2) = F-norm([A, B]). If IJOB = 0 or 1, DIF is not referenced.

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
If IJOB = 0, WORK is not referenced. Otherwise, on exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
STGSEN
The dimension of the array WORK. LWORK $\geq$ 4*N+16.
If IJOB = 1, 2 or 4, LWORK $\geq$ MAX(4*N+16, 2*M*(N−M)).
If IJOB = 3 or 5, LWORK $\geq$ MAX(4*N+16, 4*M*(N−M)).
CTGSEN
The dimension of the array WORK. LWORK $\geq$ 1.
If IJOB = 1, 2 or 4, LWORK $\geq$ 2*M*(N−M).
If IJOB = 3 or 5, LWORK $\geq$ 4*M*(N−M).

If LWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

IWORK (workspace/output) INTEGER array, dimension (LIWORK)
IF IJOB = 0, IWORK is not referenced. Otherwise, on exit, if INFO = 0, IWORK(1) returns the optimal LIWORK.

LIWORK (input) INTEGER
STGSEN
The dimension of the array IWORK. LIWORK $\geq$ 1. If IJOB = 1, 2 or 4, LIWORK $\geq$ N+6. If IJOB = 3 or 5, LIWORK $\geq$ MAX(2*M*(N−M), N+6).
CTGSEN
The dimension of the array IWORK. LIWORK $\geq$ 1.
If IJOB = 1, 2 or 4, LIWORK $\geq$ N+2.
If IJOB = 3 or 5, LIWORK $\geq$ MAX(N+2, 2*M*(N−M)).

If LIWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the IWORK array, returns this value as the first entry of the IWORK array, and no error message related to LIWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
= 1: Reordering of (A, B) failed because the transformed matrix pair (A, B) would be too far from generalized Schur form; the problem is very ill-conditioned. (A, B) may have been partially reordered. If requested, 0 is returned in DIF(*), PL and PR.

STGSJA/CTGSJA

```
      SUBROUTINE STGSJA( JOBU, JOBV, JOBQ, M, P, N, K, L, A, LDA, B,
     $                   LDB, TOLA, TOLB, ALPHA, BETA, U, LDU, V, LDV,
     $                   Q, LDQ, WORK, NCYCLE, INFO )
      CHARACTER          JOBQ, JOBU, JOBV
      INTEGER            INFO, K, L, LDA, LDB, LDQ, LDU, LDV, M, N,
     $                   NCYCLE, P
      REAL               TOLA, TOLB
      REAL               ALPHA( * ), BETA( * ), A( LDA, * ),
     $                   B( LDB, * ), Q( LDQ, * ), U( LDU, * ),
     $                   V( LDV, * ), WORK( * )

      SUBROUTINE CTGSJA( JOBU, JOBV, JOBQ, M, P, N, K, L, A, LDA, B,
     $                   LDB, TOLA, TOLB, ALPHA, BETA, U, LDU, V, LDV,
     $                   Q, LDQ, WORK, NCYCLE, INFO )
      CHARACTER          JOBQ, JOBU, JOBV
      INTEGER            INFO, K, L, LDA, LDB, LDQ, LDU, LDV, M, N,
     $                   NCYCLE, P
      REAL               TOLA, TOLB
      REAL               ALPHA( * ), BETA( * )
      COMPLEX            A( LDA, * ), B( LDB, * ), Q( LDQ, * ),
     $                   U( LDU, * ), V( LDV, * ), WORK( * )
```

Purpose

STGSJA/CTGSJA computes the generalized singular value decomposition (GSVD) of two real/complex upper triangular (or trapezoidal) matrices A and B.

On entry, it is assumed that matrices A and B have the following forms, which may be obtained by the preprocessing subroutine SGGSVP/CGGSVP from a general m-by-n matrix A and p-by-n matrix B:

$$A = \begin{array}{r} \\ k \\ l \\ m-k-l \end{array} \begin{array}{c} \begin{array}{ccc} n-k-l & k & l \end{array} \\ \begin{pmatrix} 0 & A_{12} & A_{13} \\ 0 & 0 & A_{23} \\ 0 & 0 & 0 \end{pmatrix} \end{array} \text{ if } m-k-l \geq 0;$$

$$= \begin{array}{r} \\ k \\ m-k \end{array} \begin{array}{c} \begin{array}{ccc} n-k-l & k & l \end{array} \\ \begin{pmatrix} 0 & A_{12} & A_{13} \\ 0 & 0 & A_{23} \end{pmatrix} \end{array} \text{ if } m-k-l < 0;$$

$$B = \begin{array}{r} \\ l \\ p-l \end{array} \begin{array}{c} \begin{array}{ccc} n-k-l & k & l \end{array} \\ \begin{pmatrix} 0 & 0 & B_{13} \\ 0 & 0 & 0 \end{pmatrix} \end{array}$$

where the k-by-k matrix A_{12} and l-by-l matrix B_{13} are nonsingular upper triangular; A_{23} is l-by-l upper triangular if m−k−l $\geq$ 0, otherwise A_{23} is (m−k)-by-l upper trapezoidal.

On exit,

$$U^H*A*Q = D_1*\begin{pmatrix} 0 & R \end{pmatrix}, \quad V^H*B*Q = D_2*\begin{pmatrix} 0 & R \end{pmatrix},$$

where U, V and Q are orthogonal/unitary matrices, R is a nonsingular upper triangular matrix, and D_1 and D_2 are "diagonal" matrices, which are of the following structures:

If m−k−l ≥ 0,

$$D_1 = \begin{array}{r} \\ k \\ l \\ m-k-l \end{array}\!\!\begin{array}{c} \begin{array}{cc} k & l \end{array} \\ \begin{pmatrix} I & 0 \\ 0 & C \\ 0 & 0 \end{pmatrix} \end{array}$$

$$D_2 = \begin{array}{r} \\ l \\ p-l \end{array}\!\!\begin{array}{c} \begin{array}{cc} k & l \end{array} \\ \begin{pmatrix} 0 & S \\ 0 & 0 \end{pmatrix} \end{array}$$

$$\begin{pmatrix} 0 & R \end{pmatrix} = \begin{array}{r} \\ k \\ l \end{array}\!\!\begin{array}{c} \begin{array}{ccc} n-k-l & k & l \end{array} \\ \begin{pmatrix} 0 & R_{11} & R_{12} \\ 0 & 0 & R_{22} \end{pmatrix} \end{array}$$

where

C = diag(ALPHA(k+1), ... , ALPHA(k+l)),
S = diag(BETA(k+1), ... , BETA(k+l)),
$C^2 + S^2 = I$;

R is stored in A(1:k+l, n−k−l+1:n) on exit.

If m−k−l < 0,

$$D_1 = \begin{array}{r} \\ k \\ m-k \end{array}\!\!\begin{array}{c} \begin{array}{ccc} k & m-k & k+l-m \end{array} \\ \begin{pmatrix} I & 0 & 0 \\ 0 & C & 0 \end{pmatrix} \end{array}$$

$$D_2 = \begin{array}{r} \\ m-k \\ k+l-m \\ p-l \end{array}\!\!\begin{array}{c} \begin{array}{ccc} k & m-k & k+l-m \end{array} \\ \begin{pmatrix} 0 & S & 0 \\ 0 & 0 & I \\ 0 & 0 & 0 \end{pmatrix} \end{array}$$

$$\begin{pmatrix} 0 & R \end{pmatrix} = \begin{array}{r} \\ k \\ m-k \\ k+l-m \end{array}\!\!\begin{array}{c} \begin{array}{cccc} n-k-l & k & m-k & k+l-m \end{array} \\ \begin{pmatrix} 0 & R_{11} & R_{12} & R_{13} \\ 0 & 0 & R_{22} & R_{23} \\ 0 & 0 & 0 & R_{33} \end{pmatrix} \end{array}$$

where

C = diag(ALPHA(k+1), ... , ALPHA(m)),
S = diag(BETA(k+1), ... , BETA(m)),
$C^2 + S^2 = I$;

$\begin{pmatrix} R_{11} & R_{12} & R_{13} \\ 0 & R_{22} & R_{23} \end{pmatrix}$ is stored in A(1:m, n−k−l+1:n), and R_{33} is stored in B(m−k+1:l, n+m−k−l+1:n) on exit.

The computation of the orthogonal/unitary transformation matrices U, V or Q is optional. The matrices may either be formed explicitly, or they may be postmultiplied into input matrices U_1, V_1 or Q_1.

Arguments

JOBU (input) CHARACTER*1
= 'U': U must contain a orthogonal/unitary matrix U_1 on entry, and the product U_1*U is returned;
= 'I': U is initialized to the unit matrix, and the orthogonal/unitary matrix U is returned;
= 'N': U is not computed.

JOBV (input) CHARACTER*1
= 'V': V must contain a orthogonal/unitary matrix V_1 on entry, and the product V_1*V is returned;
= 'I': V is initialized to the unit matrix, and the orthogonal/unitary matrix V is returned;
= 'N': V is not computed.

JOBQ (input) CHARACTER*1
= 'Q': Q must contain a orthogonal/unitary matrix Q_1 on entry, and the product Q_1*Q is returned;
= 'I': Q is initialized to the unit matrix, and the orthogonal/unitary matrix Q is returned;
= 'N': Q is not computed.

M (input) INTEGER
The number of rows of the matrix A. M ≥ 0.

P (input) INTEGER
The number of rows of the matrix B. P ≥ 0.

N (input) INTEGER
The number of columns of the matrices A and B. N ≥ 0.

K, L (input) INTEGER
K and L specify the subblocks in the input matrices A and B: A_{23} = A(K+1:min(K+L,M),N−L+1:N) and B_{13} = B(1:L,N−L+1:N) of A and B, whose GSVD is going to be computed by STGSJA/CTGSJA.

A (input/output) REAL/COMPLEX array, dimension (LDA,N)
On entry, the m-by-n matrix A.
On exit, A(N−K+1:N,1:min(K+L,M)) contains the triangular matrix R or part of R.

LDA (input) INTEGER
The leading dimension of the array A. LDA ≥ max(1,M).

B (input/output) REAL/COMPLEX array, dimension (LDB,N)
On entry, the p-by-n matrix B.

On exit, if necessary, B(M−K+1:L,N+M−K-L+1:N) contains a part of R.

LDB (input) INTEGER
The leading dimension of the array B. LDB $\geq$ max(1,P).

TOLA (input) REAL

TOLB (input) REAL
TOLA and TOLB are the convergence criteria for the Jacobi-Kogbetliantz iteration procedure. Generally, they are the same as used in the preprocessing step, say
TOLA = max(m,n)*$\|A\|$*MACHEPS,
TOLB = max(p,n)*$\|B\|$*MACHEPS.

ALPHA (output) REAL array, dimension (N)

BETA (output) REAL array, dimension (N)
On exit, ALPHA and BETA contain the generalized singular value pairs of A and B;
ALPHA(1:K)=1,
BETA(1:K)=0,
and if M−K−L $\geq$ 0,
ALPHA(K+1:K+L)=diag(C),
BETA(K+1:K+L)=diag(S),
or if M−K−L < 0,
ALPHA(K+1:M)=C, ALPHA(M+1:K+L)=0,
BETA(K+1:M)=S, BETA(M+1:K+L)=1.
Furthermore, if K+L < N,
ALPHA(K+L+1:N) = 0 and BETA(K+L+1:N) = 0.

U (input/output) REAL/COMPLEX array, dimension (LDU,M)
On entry, if JOBU = 'U', U must contain a matrix U_1 (usually the orthogonal/unitary matrix returned by SGGSVP/CGGSVP).
On exit,
if JOBU = 'I', U contains the orthogonal/unitary matrix U;
if JOBU = 'U', U contains the product U_1*U.
If JOBU = 'N': U is not referenced.

LDU (input) INTEGER
The leading dimension of the array U.
LDU $\geq$ max(1,M) if JOBU = 'U'; LDU $\geq$ 1 otherwise.

V (input/output) REAL/COMPLEX array, dimension (LDV,P)
On entry, if JOBV = 'V', V must contain a matrix V_1 (usually the orthogonal/unitary matrix returned by SGGSVP/CGGSVP).
On exit,
if JOBV = 'I', V contains the orthogonal/unitary matrix V;
if JOBV = 'V', V contains the product V_1*V.
If JOBV = 'N': V is not referenced.

LDV (input) INTEGER
The leading dimension of the array V.
LDV $\geq$ max(1,P) if JOBV = 'V'; LDV $\geq$ 1 otherwise.

Q (input/output) REAL/COMPLEX array, dimension (LDQ,N)
On entry, if JOBQ = 'Q', Q must contain a matrix Q_1 (usually the orthogonal/unitary matrix returned by SGGSVP/CGGSVP).
On exit,
if JOBQ = 'I', Q contains the orthogonal/unitary matrix Q;
if JOBQ = 'Q', Q contains the product Q_1*Q.
If JOBQ = 'N': Q is not referenced.

LDQ (input) INTEGER
The leading dimension of the array Q.
LDQ $\geq$ max(1,N) if JOBQ = 'Q'; LDQ $\geq$ 1 otherwise.

WORK (workspace) REAL/COMPLEX array, dimension (2*N)

NCYCLE (output) INTEGER
The number of cycles required for convergence.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
= 1: the procedure does not converge after MAXIT cycles.

STGSNA/CTGSNA

```
      SUBROUTINE STGSNA( JOB, HOWMNY, SELECT, N, A, LDA, B, LDB, VL,
     $                   LDVL, VR, LDVR, S, DIF, MM, M, WORK, LWORK,
     $                   IWORK, INFO )
      CHARACTER          HOWMNY, JOB
      INTEGER            INFO, LDA, LDB, LDVL, LDVR, LWORK, M, MM, N
      LOGICAL            SELECT( * )
      INTEGER            IWORK( * )
      REAL               A( LDA, * ), B( LDB, * ), DIF( * ), S( * ),
     $                   VL( LDVL, * ), VR( LDVR, * ), WORK( * )

      SUBROUTINE CTGSNA( JOB, HOWMNY, SELECT, N, A, LDA, B, LDB, VL,
     $                   LDVL, VR, LDVR, S, DIF, MM, M, WORK, LWORK,
     $                   IWORK, INFO )
      CHARACTER          HOWMNY, JOB
      INTEGER            INFO, LDA, LDB, LDVL, LDVR, LWORK, M, MM, N
      LOGICAL            SELECT( * )
      INTEGER            IWORK( * )
      REAL               DIF( * ), S( * )
      COMPLEX            A( LDA, * ), B( LDB, * ), VL( LDVL, * ),
     $                   VR( LDVR, * ), WORK( * )
```

Purpose

STGSNA estimates reciprocal condition numbers for specified eigenvalues and/or eigenvectors of a matrix pair (A, B) in generalized real Schur canonical form (or of any matrix pair ($Q*A*Z^T$, $Q*B*Z^T$) with orthogonal matrices Q and Z.

(A,B) must be in generalized real Schur form (as returned by SGGES), i.e. A is block upper triangular with 1-by-1 and 2-by-2 diagonal blocks. B is upper triangular.

CTGSNA estimates reciprocal condition numbers for specified eigenvalues and/or eigenvectors of a matrix pair (A, B).

(A,B) must be in generalized Schur canonical form, that is, A and B are both upper triangular.

Arguments

JOB (input) CHARACTER*1
Specifies whether condition numbers are required for eigenvalues (S) or eigenvectors (DIF):
= 'E': for eigenvalues only (S);
= 'V': for eigenvectors only (DIF);
= 'B': for both eigenvalues and eigenvectors (S and DIF).

HOWMNY (input) CHARACTER*1
= 'A': compute condition numbers for all eigenpairs;
= 'S': compute condition numbers for selected eigenpairs specified by the array SELECT.

SELECT (input) LOGICAL array, dimension (N)
STGSNA
If HOWMNY = 'S', SELECT specifies the eigenpairs for which condition numbers are required. To select condition numbers for the eigenpair corresponding to a real eigenvalue w(j), SELECT(j) must be set to .TRUE.. To select condition numbers corresponding to a complex conjugate pair of eigenvalues w(j) and w(j+1), either SELECT(j) or SELECT(j+1) or both, must be set to .TRUE..
If HOWMNY = 'A', SELECT is not referenced.
CTGSNA
If HOWMNY = 'S', SELECT specifies the eigenpairs for which condition numbers are required. To select condition numbers for the corresponding j-th eigenvalue and/or eigenvector, SELECT(j) must be set to .TRUE..
If HOWMNY = 'A', SELECT is not referenced.

N (input) INTEGER
The order of the square matrix pair (A, B). $N \geq 0$.

A (input) REAL/COMPLEX array, dimension (LDA,N)
The upper quasi-triangular/triangular matrix A in the pair (A,B).

LDA (input) INTEGER
The leading dimension of the array A. $LDA \geq \max(1,N)$.

B (input) REAL/COMPLEX array, dimension (LDB,N)
The upper triangular matrix B in the pair (A,B).

LDB (input) INTEGER
The leading dimension of the array B. $LDB \geq \max(1,N)$.

VL (input) REAL/COMPLEX array, dimension (LDVL,M)
IF JOB = 'E' or 'B', VL must contain left eigenvectors of (A, B), corresponding to the eigenpairs specified by HOWMNY and SELECT. The eigenvectors must be stored in consecutive columns of VL, as returned by STGEVC/CTGEVC. If JOB = 'V', VL is not referenced.

LDVL (input) INTEGER
The leading dimension of the array VL. $LDVL \geq 1$; and If JOB = 'E' or 'B', $LDVL \geq N$.

VR (input) REAL/COMPLEX array, dimension (LDVR,M)
IF JOB = 'E' or 'B', VR must contain right eigenvectors of (A, B), corresponding to the eigenpairs specified by HOWMNY and SELECT. The eigenvectors must be stored in consecutive columns of VR, as returned by STGEVC/CTGEVC. If JOB = 'V', VR is not referenced.

LDVR (input) INTEGER
The leading dimension of the array VR. $LDVR \geq 1$; If JOB = 'E' or 'B', $LDVR \geq N$.

S (output) REAL array, dimension (MM)
STGSNA
If JOB = 'E' or 'B', the reciprocal condition numbers of the selected eigenvalues, stored in consecutive elements of the array. For a complex conjugate pair of eigenvalues two consecutive elements of S are set to the same value. Thus S(j), DIF(j), and the jth columns of VL and VR all correspond to the same eigenpair (but not in general the jth eigenpair, unless all eigenpairs are selected). If JOB = 'V', S is not referenced.
CTGSNA
If JOB = 'E' or 'B', the reciprocal condition numbers of the selected eigenvalues, stored in consecutive elements of the array.
If JOB = 'V', S is not referenced.

DIF (output) REAL array, dimension (MM)
STGSNA
If JOB = 'V' or 'B', the estimated reciprocal condition numbers of the selected eigenvectors, stored in consecutive elements of the array. For a complex eigenvector two consecutive elements of DIF are set to the same value. If the eigenvalues cannot be reordered to compute DIF(j), DIF(j) is set to 0; this can only occur when the true value would be very small anyway.
If JOB = 'E', DIF is not referenced.
CTGSNA
If JOB = 'V' or 'B', the estimated reciprocal condition numbers of the selected eigenvectors, stored in consecutive elements of the array. If the eigenvalues cannot be reordered to compute DIF(j), DIF(j) is set to 0; this can only occur when the true value would be very small anyway. For each eigenvalue/vector specified by SELECT, DIF stores a Frobenius norm-based estimate of Difl.
If JOB = 'E', DIF is not referenced.

MM (input) INTEGER
The number of elements in the arrays S and DIF. $MM \geq M$.

M (output) INTEGER
The number of elements of the arrays S and DIF used to store the specified condition numbers; for each selected eigenvalue one element is used. If HOWMNY = 'A', M is set to N.

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
If JOB = 'E', WORK is not referenced. Otherwise, on exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
STGSNA
The dimension of the array WORK. LWORK $\geq$ N.
If JOB = 'V' or 'B' LWORK $\geq$ 2*N*(N+2)+16.
CTGSNA
The dimension of the array WORK. LWORK $\geq$ 1.
If JOB = 'V' or 'B', LWORK $\geq$ 2*N*N.

IWORK *STGSNA* (workspace) INTEGER array, dimension (N+6)
CTGSNA (workspace) INTEGER array, dimension (N+2)
IF JOB = 'E', IWORK is not referenced.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = $-$i, the ith argument had an illegal value.

STGSYL/CTGSYL

```
      SUBROUTINE STGSYL( TRANS, IJOB, M, N, A, LDA, B, LDB, C, LDC, D,
     $                   LDD, E, LDE, F, LDF, SCALE, DIF, WORK, LWORK,
     $                   IWORK, INFO )
      CHARACTER          TRANS
      INTEGER            IJOB, INFO, LDA, LDB, LDC, LDD, LDE, LDF,
     $                   LWORK, M, N
      REAL               DIF, SCALE
      INTEGER            IWORK( * )
      REAL               A( LDA, * ), B( LDB, * ), C( LDC, * ),
     $                   D( LDD, * ), E( LDE, * ), F( LDF, * ),
     $                   WORK( * )
```

```
      SUBROUTINE CTGSYL( TRANS, IJOB, M, N, A, LDA, B, LDB, C, LDC, D,
     $                   LDD, E, LDE, F, LDF, SCALE, DIF, WORK, LWORK,
     $                   IWORK, INFO )
      CHARACTER          TRANS
      INTEGER            IJOB, INFO, LDA, LDB, LDC, LDD, LDE, LDF,
     $                   LWORK, M, N
      REAL               DIF, SCALE
      INTEGER            IWORK( * )
      COMPLEX            A( LDA, * ), B( LDB, * ), C( LDC, * ),
     $                   D( LDD, * ), E( LDE, * ), F( LDF, * ),
     $                   WORK( * )
```

Purpose

STGSYL/CTGSYL solves the generalized Sylvester equation:

$$\begin{aligned} A*R-L*B &= scale*C \\ D*R-L*E &= scale*F \end{aligned} \qquad (1)$$

where R and L are unknown m-by-n matrices, (A, D), (B, E) and (C, F) are given matrix pairs of size m-by-m, n-by-n and m-by-n, respectively, with real/complex entries. (A, D) and (B, E) must be in generalized real-Schur/Schur canonical form, i.e. A, B are upper quasi-triangular/triangular and D, E are upper triangular.

The solution (R, L) overwrites (C, F). 0 $\leq$ SCALE $\leq$ 1 is an output scaling factor chosen to avoid overflow.

In matrix notation (1) is equivalent to solve Z * x = scale * b, where Z is defined as

$$Z = \begin{pmatrix} kron(In, A) & -kron(B', Im) \\ kron(In, D) & -kron(E', Im) \end{pmatrix} \qquad (2)$$

Here Ik is the identity matrix of size k and X' is the transpose/conjugate-transpose of X. kron(X, Y) is the Kronecker product between the matrices X and Y.

If TRANS = 'T' (*STGSYL*) or TRANS = 'C' (*CTGSYL*), STGSYL/CTGSYL solves the transposed/conjugate-transposed system Z' * y = scale * b, which is equivalent to solve for R and L in

$$\begin{aligned} A'*R+D'*L &= scale*C \\ R*B'+L*E' &= scale*(-F) \end{aligned} \qquad (3)$$

This case (TRANS = 'T' for STGSYL or TRANS = 'C' for CTGSYL) is used to compute an one-norm-based estimate of Dif[(A,D), (B,E)], the separation between the matrix pairs (A,D) and (B,E), using SLACON/CLACON.

If IJOB $\geq$ 1, STGSYL/CTGSYL computes a Frobenius norm-based estimate of Dif[(A,D),(B,E)]. That is, the reciprocal of a lower bound on the reciprocal of the smallest singular value of Z. See [1-2] for more information.

This is a level 3 BLAS algorithm.

Arguments

TRANS (input) CHARACTER*1
= 'N': solve the generalized Sylvester equation (1).
= 'T': solve the 'transposed' system (3). (*STGSYL only*)
= 'C': solve the "conjugate transposed" system (3). (*CTGSYL only*)

IJOB (input) INTEGER
Specifies what kind of functionality to be performed.
= 0: solve (1) only.
= 1: The functionality of 0 and 3.
= 2: The functionality of 0 and 4.
= 3: Only an estimate of Dif[(A,D), (B,E)] is computed. (look ahead strategy is used).
= 4: Only an estimate of Dif[(A,D), (B,E)] is computed. (SGECON/CGECON on sub-systems is used).
Not referenced if TRANS = 'T' (STGSYL) or TRANS = 'C' (CTGSYL).

M (input) INTEGER
The order of the matrices A and D, and the row dimension of the matrices C, F, R and L.

N (input) INTEGER
The order of the matrices B and E, and the column dimension of the matrices C, F, R and L.

A (input) REAL/COMPLEX array, dimension (LDA,M)
The upper quasi-triangular/triangular matrix A.

LDA (input) INTEGER
The leading dimension of the array A. LDA $\geq$ max(1,M).

B (input) REAL/COMPLEX array, dimension (LDB,N)
The upper quasi-triangular/triangular matrix B.

LDB (input) INTEGER
The leading dimension of the array B. LDB $\geq$ max(1,N).

C (input/output) REAL/COMPLEX array, dimension (LDC,N)
On entry, C contains the right-hand-side of the first matrix equation in (1) or (3).
On exit, if IJOB = 0, 1 or 2, C has been overwritten by the solution R. If IJOB = 3 or 4 and TRANS = 'N', C holds R, the solution achieved during the computation of the Dif-estimate.

LDC (input) INTEGER
The leading dimension of the array C. LDC $\geq$ max(1,M).

D (input) REAL/COMPLEX array, dimension (LDD,M)
The upper triangular matrix D.

LDD (input) INTEGER
The leading dimension of the array D. LDD $\geq$ max(1, M).

E (input) REAL/COMPLEX array, dimension (LDE,N)
The upper triangular matrix E.

LDE (input) INTEGER
The leading dimension of the array E. LDE $\geq$ max(1, N).

F (input/output) REAL/COMPLEX array, dimension (LDF,N)
On entry, F contains the right-hand-side of the second matrix equation in (1) or (3).
On exit, if IJOB = 0, 1 or 2, F has been overwritten by the solution L. If IJOB = 3 or 4 and TRANS = 'N', F holds L, the solution achieved during the computation of the Dif-estimate.

LDF (input) INTEGER
The leading dimension of the array F. LDF $\geq$ max(1, M).

DIF (output) REAL
On exit DIF is the reciprocal of a lower bound of the reciprocal of the Dif-function, i.e. DIF is an upper bound of Dif[(A,D), (B,E)] = sigma_min(Z), where Z as in (2).
IF IJOB = 0 or TRANS = 'T' (STGSYL) or TRANS = 'C' (CTGSYL), DIF is not touched.

SCALE (output) REAL
On exit SCALE is the scaling factor in (1) or (3). If 0 < SCALE < 1, C and F hold the solutions R and L, resp., to a slightly perturbed system but the input matrices A, B, D and E have not been changed. If SCALE = 0, C and F hold the solutions R and L, respectively, to the homogeneous system with C = F = 0. Normally, SCALE = 1.

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
If IJOB = 0, WORK is not referenced. Otherwise, on exit, if INFO = 0, WORK(1) returns the optimal LWORK.

LWORK (input) INTEGER
The dimension of the array WORK. LWORK $\geq$ 1.
If IJOB = 1 or 2 and TRANS = 'N', LWORK $\geq$ 2*M*N.

IWORK *STGSYL* (workspace) INTEGER array, dimension (M+N+6)
CTGSYL (workspace) INTEGER array, dimension (M+N+2)
If IJOB = 0, IWORK is not referenced.

If LWORK = −1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: (A, D) and (B, E) have common or close eigenvalues.

STPCON/CTPCON

```
      SUBROUTINE STPCON( NORM, UPLO, DIAG, N, AP, RCOND, WORK, IWORK,
     $                   INFO )
      CHARACTER          DIAG, NORM, UPLO
      INTEGER            INFO, N
      REAL               RCOND
      INTEGER            IWORK( * )
      REAL               AP( * ), WORK( * )

      SUBROUTINE CTPCON( NORM, UPLO, DIAG, N, AP, RCOND, WORK, RWORK,
     $                   INFO )
      CHARACTER          DIAG, NORM, UPLO
      INTEGER            INFO, N
      REAL               RCOND
      REAL               RWORK( * )
      COMPLEX            AP( * ), WORK( * )
```

Purpose

STPCON/CTPCON estimates the reciprocal of the condition number of a packed triangular matrix A, in either the 1-norm or the infinity-norm.

The norm of A is computed and an estimate is obtained for $\|A^{-1}\|$, then the reciprocal of the condition number is computed as RCOND $= 1/(\|A\| * \|A^{-1}\|)$.

Arguments

NORM (input) CHARACTER*1
Specifies whether the 1-norm condition number or the infinity-norm condition number is required:
= '1' or 'O': 1-norm;
= 'I': Infinity-norm.

UPLO (input) CHARACTER*1
= 'U': A is upper triangular;
= 'L': A is lower triangular.

DIAG (input) CHARACTER*1
= 'N': A is non-unit triangular;
= 'U': A is unit triangular.

N (input) INTEGER
The order of the matrix A. N $\geq$ 0.

AP (input) REAL/COMPLEX array, dimension (N*(N+1)/2)
The upper or lower triangular matrix A, packed columnwise in a linear array. The jth column of A is stored in the array AP as follows:
if UPLO = 'U', AP(i + (j−1)*j/2) = A(i,j) for 1≤i≤j;
if UPLO = 'L', AP(i + (j−1)*(2*n−j)/2) = A(i,j) for j≤i≤n.
If DIAG = 'U', the diagonal elements of A are not referenced and are assumed to be 1.

RCOND (output) REAL
The reciprocal of the condition number of the matrix A, computed as RCOND $= 1/(\|A\| * \|A^{-1}\|)$.

WORK *STPCON* (workspace) REAL array, dimension (3*N)
CTPCON (workspace) COMPLEX array, dimension (2*N)

IWORK *STPCON only* (workspace) INTEGER array, dimension (N)

RWORK *CTPCON only* (workspace) REAL array, dimension (N)

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

STPRFS/CTPRFS

```
      SUBROUTINE STPRFS( UPLO, TRANS, DIAG, N, NRHS, AP, B, LDB, X, LDX,
     $                   FERR, BERR, WORK, IWORK, INFO )
      CHARACTER          DIAG, TRANS, UPLO
      INTEGER            INFO, LDB, LDX, N, NRHS
      INTEGER            IWORK( * )
      REAL               AP( * ), B( LDB, * ), BERR( * ), FERR( * ),
     $                   WORK( * ), X( LDX, * )

      SUBROUTINE CTPRFS( UPLO, TRANS, DIAG, N, NRHS, AP, B, LDB, X, LDX,
     $                   FERR, BERR, WORK, RWORK, INFO )
      CHARACTER          DIAG, TRANS, UPLO
      INTEGER            INFO, LDB, LDX, N, NRHS
      REAL               BERR( * ), FERR( * ), RWORK( * )
      COMPLEX            AP( * ), B( LDB, * ), WORK( * ), X( LDX, * )
```

Purpose

STPRFS/CTPRFS provides error bounds and backward error estimates for the solution to a system of linear equations with a triangular packed coefficient matrix.

The solution matrix X must be computed by STPTRS/CTPTRS or some other means before entering this routine. STPRFS/CTPRFS does not do iterative refinement because doing so cannot improve the backward error.

Arguments

UPLO (input) CHARACTER*1
= 'U': A is upper triangular;
= 'L': A is lower triangular.

TRANS (input) CHARACTER*1
Specifies the form of the system of equations:
= 'N': A*X = B (No transpose)
= 'T': A^T*X = B (Transpose)

= 'C': $A^H*X = B$ (Conjugate transpose)

DIAG (input) CHARACTER*1
= 'N': A is non-unit triangular;
= 'U': A is unit triangular.

N (input) INTEGER
The order of the matrix A. $N \geq 0$.

NRHS (input) INTEGER
The number of right hand sides, i.e., the number of columns of the matrices B and X. $NRHS \geq 0$.

AP (input) REAL/COMPLEX array, dimension (N*(N+1)/2)
The upper or lower triangular matrix A, packed columnwise in a linear array. The jth column of A is stored in the array AP as follows:
if UPLO = 'U', $AP(i + (j-1)*j/2) = A(i,j)$ for $1 \leq i \leq j$;
if UPLO = 'L', $AP(i + (j-1)*(2*n-j)/2) = A(i,j)$ for $j \leq i \leq n$.
If DIAG = 'U', the diagonal elements of A are not referenced and are assumed to be 1.

B (input) REAL/COMPLEX array, dimension (LDB,NRHS)
The right hand side matrix B.

LDB (input) INTEGER
The leading dimension of the array B. $LDB \geq \max(1,N)$.

X (input) REAL/COMPLEX array, dimension (LDX,NRHS)
The solution matrix X.

LDX (input) INTEGER
The leading dimension of the array X. $LDX \geq \max(1,N)$.

FERR (output) REAL array, dimension (NRHS)
The estimated forward error bound for each solution vector X(j) (the jth column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) − XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR (output) REAL array, dimension (NRHS)
The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK *STPRFS* (workspace) REAL array, dimension (3*N)
CTPRFS (workspace) COMPLEX array, dimension (2*N)

IWORK *STPRFS only* (workspace) INTEGER array, dimension (N)

RWORK *CTPRFS only* (workspace) REAL array, dimension (N)

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

STPTRI/CTPTRI

```
SUBROUTINE STPTRI( UPLO, DIAG, N, AP, INFO )
CHARACTER          DIAG, UPLO
INTEGER            INFO, N
REAL               AP( * )

SUBROUTINE CTPTRI( UPLO, DIAG, N, AP, INFO )
CHARACTER          DIAG, UPLO
INTEGER            INFO, N
COMPLEX            AP( * )
```

Purpose

STPTRI/CTPTRI computes the inverse of a real/complex upper or lower triangular matrix A stored in packed format.

Arguments

UPLO (input) CHARACTER*1
= 'U': A is upper triangular;
= 'L': A is lower triangular.

DIAG (input) CHARACTER*1
= 'N': A is non-unit triangular;
= 'U': A is unit triangular.

N (input) INTEGER
The order of the matrix A. $N \geq 0$.

AP (input/output) REAL/COMPLEX array, dimension (N*(N+1)/2)
On entry, the upper or lower triangular matrix A, stored columnwise in a linear array. The jth column of A is stored in the array AP as follows:
if UPLO = 'U', $AP(i + (j-1)*j/2) = A(i,j)$ for $1 \leq i \leq j$;
if UPLO = 'L', $AP(i + (j-1)*(2*n-j)/2) = A(i,j)$ for $j \leq i \leq n$.
On exit, the (triangular) inverse of the original matrix, in the same packed storage format.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, A(i,i) is exactly zero. The triangular matrix is singular and its inverse can not be computed.

STPTRS/CTPTRS

```
SUBROUTINE STPTRS( UPLO, TRANS, DIAG, N, NRHS, AP, B, LDB, INFO )
CHARACTER          DIAG, TRANS, UPLO
INTEGER            INFO, LDB, N, NRHS
REAL               AP( * ), B( LDB, * )

SUBROUTINE CTPTRS( UPLO, TRANS, DIAG, N, NRHS, AP, B, LDB, INFO )
CHARACTER          DIAG, TRANS, UPLO
INTEGER            INFO, LDB, N, NRHS
COMPLEX            AP( * ), B( LDB, * )
```

Purpose

STPTRS/CTPTRS solves a triangular system of the form A*X = B, A^T*X = B, or A^H*X = B, where A is a triangular matrix of order n stored in packed format, and B is an n-by-nrhs matrix. A check is made to verify that A is nonsingular.

Arguments

UPLO (input) CHARACTER*1
= 'U': A is upper triangular;
= 'L': A is lower triangular.

TRANS (input) CHARACTER*1
Specifies the form of the system of equations:
= 'N': A*X = B (No transpose)
= 'T': A^T*X = B (Transpose)
= 'C': A^H*X = B (Conjugate transpose)

DIAG (input) CHARACTER*1
= 'N': A is non-unit triangular;
= 'U': A is unit triangular.

N (input) INTEGER
The order of the matrix A. N $\geq$ 0.

NRHS (input) INTEGER
The number of right hand sides, i.e., the number of columns of the matrix B. NRHS $\geq$ 0.

AP (input) REAL/COMPLEX array, dimension (N*(N+1)/2)
The upper or lower triangular matrix A, packed columnwise in a linear array. The jth column of A is stored in the array AP as follows:
if UPLO = 'U', AP(i + (j−1)*j/2) = A(i,j) for 1$\leq$i$\leq$j;
if UPLO = 'L', AP(i + (j−1)*(2*n−j)/2) = A(i,j) for j$\leq$i$\leq$n.

B (input/output) REAL/COMPLEX array, dimension (LDB,NRHS)
On entry, the right hand side matrix B.
On exit, if INFO = 0, the solution matrix X.

LDB (input) INTEGER
The leading dimension of the array B. LDB $\geq$ max(1,N).

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.
> 0: if INFO = i, the ith diagonal element of A is zero, indicating that the matrix is singular and the solutions X have not been computed.

STRCON/CTRCON

```
SUBROUTINE STRCON( NORM, UPLO, DIAG, N, A, LDA, RCOND, WORK,
$                  IWORK, INFO )
CHARACTER          DIAG, NORM, UPLO
INTEGER            INFO, LDA, N
REAL               RCOND
INTEGER            IWORK( * )
REAL               A( LDA, * ), WORK( * )

SUBROUTINE CTRCON( NORM, UPLO, DIAG, N, A, LDA, RCOND, WORK,
$                  RWORK, INFO )
CHARACTER          DIAG, NORM, UPLO
INTEGER            INFO, LDA, N
REAL               RCOND
REAL               RWORK( * )
COMPLEX            A( LDA, * ), WORK( * )
```

Purpose

STRCON/CTRCON estimates the reciprocal of the condition number of a triangular matrix A, in either the 1-norm or the infinity-norm.

The norm of A is computed and an estimate is obtained for $\|A^{-1}\|$, then the reciprocal of the condition number is computed as RCOND = $1/(\|A\| * \|A^{-1}\|)$.

Arguments

NORM (input) CHARACTER*1
Specifies whether the 1-norm condition number or the infinity-norm condition number is required:
= '1' or 'O': 1-norm;
= 'I': Infinity-norm.

UPLO (input) CHARACTER*1
= 'U': A is upper triangular;
= 'L': A is lower triangular.

DIAG (input) CHARACTER*1
= 'N': A is non-unit triangular;
= 'U': A is unit triangular.

N (input) INTEGER
The order of the matrix A. N $\geq$ 0.

A (input) REAL/COMPLEX array, dimension (LDA,N)
The triangular matrix A. If UPLO = 'U', the leading n-by-n upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n-by-n lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If DIAG = 'U', the diagonal elements of A are also not referenced and are assumed to be 1.

LDA (input) INTEGER
The leading dimension of the array A. LDA $\geq$ max(1,N).

RCOND (output) REAL
The reciprocal of the condition number of the matrix A, computed as RCOND = $1/(\|A\| * \|A^{-1}\|)$.

WORK *STRCON* (workspace) REAL array, dimension (3*N)
CTRCON (workspace) COMPLEX array, dimension (2*N)

IWORK *STRCON only* (workspace) INTEGER array, dimension (N)

RWORK *CTRCON only* (workspace) REAL array, dimension (N)

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = $-i$, the ith argument had an illegal value.

STREVC/CTREVC

```
      SUBROUTINE STREVC( SIDE, HOWMNY, SELECT, N, T, LDT, VL, LDVL, VR,
     $                   LDVR, MM, M, WORK, INFO )
      CHARACTER          HOWMNY, SIDE
      INTEGER            INFO, LDT, LDVL, LDVR, M, MM, N
      LOGICAL            SELECT( * )
      REAL               T( LDT, * ), VL( LDVL, * ), VR( LDVR, * ),
     $                   WORK( * )

      SUBROUTINE CTREVC( SIDE, HOWMNY, SELECT, N, T, LDT, VL, LDVL, VR,
     $                   LDVR, MM, M, WORK, RWORK, INFO )
      CHARACTER          HOWMNY, SIDE
      INTEGER            INFO, LDT, LDVL, LDVR, M, MM, N
      LOGICAL            SELECT( * )
      REAL               RWORK( * )
      COMPLEX            T( LDT, * ), VL( LDVL, * ), VR( LDVR, * ),
     $                   WORK( * )
```

Purpose

STREVC/CTREVC computes some or all of the right and/or left eigenvectors of a real/complex upper quasi-triangular/triangular matrix T.

The right eigenvector x and the left eigenvector y of T corresponding to an eigenvalue w are defined by:

$$T*x = w*x \quad \text{and} \quad y^H*T = w*y^H.$$

If all eigenvectors are requested, the routine may either return the matrices X and/or Y of right or left eigenvectors of T, or the products Q*X and/or Q*Y, where Q is an input orthogonal/unitary matrix. If T was obtained from the real-Schur/Schur factorization of an original matrix $A = Q*T*Q^H$, then Q*X and Q*Y are the matrices of right or left eigenvectors of A.

STREVC only
T must be in Schur canonical form (as returned by SHSEQR), that is, block upper triangular with 1-by-1 and 2-by-2 diagonal blocks; each 2-by-2 diagonal block has its diagonal elements equal and its off-diagonal elements of opposite sign. Corresponding to each 2-by-2 diagonal block is a complex conjugate pair of eigenvalues and eigenvectors; only one eigenvector of the pair is computed, namely the one corresponding to the eigenvalue with positive imaginary part.

Arguments

SIDE (input) CHARACTER*1
= 'R': compute right eigenvectors only;
= 'L': compute left eigenvectors only;
= 'B': compute both right and left eigenvectors.

HOWMNY (input) CHARACTER*1
= 'A': compute all right and/or left eigenvectors;
= 'B': compute all right and/or left eigenvectors, and backtransform them using the input matrices supplied in VR and/or VL;
= 'S': compute selected right and/or left eigenvectors, specified by the logical array SELECT.

SELECT *STREVC* (input/output) LOGICAL array, dimension (N)
CTREVC (input) LOGICAL array, dimension (N)
If HOWMNY = 'S', SELECT specifies the eigenvectors to be computed.
If HOWMNY = 'A' or 'B', SELECT is not referenced.
STREVC
To select the real eigenvector corresponding to a real eigenvalue w(j), SELECT(j) must be set to .TRUE.. To select the complex eigenvector corresponding to a complex conjugate pair w(j) and w(j+1), either SELECT(j) or SELECT(j+1) must be set to .TRUE.; then on exit SELECT(j) is .TRUE. and SELECT(j+1) is .FALSE..
CTREVC
To select the eigenvector corresponding to the jth eigenvalue, SELECT(j) must be set to .TRUE..

N (input) INTEGER
The order of the matrix T. N $\geq$ 0.

T (input) REAL array, dimension (LDT,N)
STREVC

The upper quasi-triangular matrix T in Schur canonical form.
CTREVC
The upper triangular matrix T. T is modified by the routine, but restored on exit.

LDT (input) INTEGER
The leading dimension of the array T. LDT $\geq$ max(1,N).

VL (input/output) REAL/COMPLEX array, dimension (LDVL,MM)
On entry, if SIDE = 'L' or 'B' and HOWMNY = 'B', VL must contain an n-by-n matrix Q (usually the orthogonal/unitary matrix Q of Schur vectors returned by SHSEQR/CHSEQR).
On exit, if SIDE = 'L' or 'B', VL contains:
if HOWMNY = 'A', the matrix Y of left eigenvectors of T;
STREVC
VL has the same quasi-lower triangular form as T'. If T(i,i) is a real eigenvalue, then the ith column VL(i) of VL is its corresponding eigenvector. If T(i:i+1,i:i+1) is a 2-by-2 block whose eigenvalues are complex-conjugate eigenvalues of T, then VL(i)+$\sqrt{-1}$*VL(i+1) is the complex eigenvector corresponding to the eigenvalue with positive real part.
CTREVC
VL is lower triangular. The ith column VL(i) of VL is the eigenvector corresponding to T(i,i).
if HOWMNY = 'B', the matrix Q*Y;
if HOWMNY = 'S', the left eigenvectors of T specified by SELECT, stored consecutively in the columns of VL, in the same order as their eigenvalues.
If SIDE = 'R', VL is not referenced.
STREVC only
A complex eigenvector corresponding to a complex eigenvalue is stored in two consecutive columns, the first holding the real part, and the second the imaginary part.

LDVL (input) INTEGER
The leading dimension of the array VL.
LDVL $\geq$ max(1,N) if SIDE = 'L' or 'B'; LDVL $\geq$ 1 otherwise.

VR (input/output) REAL/COMPLEX array, dimension (LDVR,MM)
On entry, if SIDE = 'R' or 'B' and HOWMNY = 'B', VR must contain an n-by-n matrix Q (usually the orthogonal/unitary matrix Q of Schur vectors returned by SHSEQR/CHSEQR).
On exit, if SIDE = 'R' or 'B', VR contains:
if HOWMNY = 'A', the matrix X of right eigenvectors of T;
STREVC
VR has the same quasi-upper triangular form as T. If T(i,i) is a real eigenvalue, then the ith column VR(i) of VR is its corresponding eigenvector. If T(i:i+1,i:i+1) is a 2-by-2 block whose eigenvalues are complex-conjugate eigenvalues of T, then VR(i)+$\sqrt{-1}$*VR(i+1) is the complex eigenvector corresponding to the eigenvalue with positive real part.
CTREVC
VR is upper triangular. The ith column VR(i) of VR is the eigenvector corresponding to T(i,i).
if HOWMNY = 'B', the matrix Q*X;
if HOWMNY = 'S', the right eigenvectors of T specified by SELECT, stored consecutively in the columns of VR, in the same order as their eigenvalues.
If SIDE = 'L', VR is not referenced.
STREVC only
A complex eigenvector corresponding to a complex eigenvalue is stored in two consecutive columns, the first holding the real part and the second the imaginary part.

LDVR (input) INTEGER
The leading dimension of the array VR.
LDVR $\geq$ max(1,N) if SIDE = 'R' or 'B'; LDVR $\geq$ 1 otherwise.

MM (input) INTEGER
The number of columns in the arrays VL and/or VR. MM $\geq$ M.

M (output) INTEGER
The number of columns in the arrays VL and/or VR actually used to store the eigenvectors. If HOWMNY = 'A' or 'B', M is set to N.
STREVC
Each selected real eigenvector occupies one column and each selected complex eigenvector occupies two columns.
CTREVC
Each selected eigenvector occupies one column.

WORK *STREVC* (workspace) REAL array, dimension (3*N)
CTREVC (workspace) COMPLEX array, dimension (2*N)

RWORK *CTREVC only* (workspace) REAL array, dimension (N)

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

STREXC/CTREXC

```
      SUBROUTINE STREXC( COMPQ, N, T, LDT, Q, LDQ, IFST, ILST, WORK,
     $                   INFO )
      CHARACTER          COMPQ
      INTEGER            IFST, ILST, INFO, LDQ, LDT, N
      REAL               Q( LDQ, * ), T( LDT, * ), WORK( * )

      SUBROUTINE CTREXC( COMPQ, N, T, LDT, Q, LDQ, IFST, ILST, INFO )
      CHARACTER          COMPQ
      INTEGER            IFST, ILST, INFO, LDQ, LDT, N
      COMPLEX            Q( LDQ, * ), T( LDT, * )
```

Purpose

STREXC/CTREXC reorders the real-Schur/Schur factorization of a real/complex matrix $A = Q*T*Q^H$, so that the diagonal block/element of T with row index IFST is moved to row ILST.

The real-Schur/Schur form T is reordered by an orthogonal/unitary similarity transformation Z^H*T*Z, and optionally the matrix Q of Schur vectors is updated by postmultiplying it with Z.

STREXC only
T must be in Schur canonical form (as returned by SHSEQR), that is, block upper triangular with 1-by-1 and 2-by-2 diagonal blocks; each 2-by-2 diagonal block has its diagonal elements equal and its off-diagonal elements of opposite sign.

Arguments

COMPQ (input) CHARACTER*1
= 'V': update the matrix Q of Schur vectors;
= 'N': do not update Q.

N (input) INTEGER
The order of the matrix T. $N \geq 0$.

T (input/output) REAL/COMPLEX array, dimension (LDT,N)
STREXC
On entry, the upper quasi-triangular matrix T, in Schur canonical form.
On exit, the reordered upper quasi-triangular matrix, again in Schur canonical form.
CTREXC
On entry, the upper triangular matrix T.
On exit, the reordered upper triangular matrix.

LDT (input) INTEGER
The leading dimension of the array T. $LDT \geq \max(1,N)$.

Q (input/output) REAL/COMPLEX array, dimension (LDQ,N)
On entry, if COMPQ = 'V', the matrix Q of Schur vectors.
On exit, if COMPQ = 'V', Q has been postmultiplied by the orthogonal/unitary transformation matrix Z which reorders T.
If COMPQ = 'N', Q is not referenced.

LDQ (input) INTEGER
The leading dimension of the array Q. $LDQ \geq \max(1,N)$.

IFST *STREXC* (input/output) INTEGER
CTREXC (input) INTEGER

ILST *STREXC* (input/output) INTEGER
CTREXC (input) INTEGER
Specify the reordering of the diagonal blocks/elements of T. The block/element with row index IFST is moved to row ILST by a sequence of transpositions between adjacent blocks/elements.
$1 \leq IFST \leq N$; $1 \leq ILST \leq N$.

STREXC only
On exit, if IFST pointed on entry to the second row of a 2-by-2 block, it is changed to point to the first row; ILST always points to the first row of the block in its final position (which may differ from its input value by ± 1).

WORK *STREXC only* (workspace) REAL array, dimension (N)

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

STRRFS/CTRRFS

```
      SUBROUTINE STRRFS( UPLO, TRANS, DIAG, N, NRHS, A, LDA, B, LDB, X,
     $                   LDX, FERR, BERR, WORK, IWORK, INFO )
      CHARACTER          DIAG, TRANS, UPLO
      INTEGER            INFO, LDA, LDB, LDX, N, NRHS
      INTEGER            IWORK( * )
      REAL               A( LDA, * ), B( LDB, * ), BERR( * ), FERR( * ),
     $                   WORK( * ), X( LDX, * )

      SUBROUTINE CTRRFS( UPLO, TRANS, DIAG, N, NRHS, A, LDA, B, LDB, X,
     $                   LDX, FERR, BERR, WORK, RWORK, INFO )
      CHARACTER          DIAG, TRANS, UPLO
      INTEGER            INFO, LDA, LDB, LDX, N, NRHS
      REAL               BERR( * ), FERR( * ), RWORK( * )
      COMPLEX            A( LDA, * ), B( LDB, * ), WORK( * ),
     $                   X( LDX, * )
```

Purpose

STRRFS/CTRRFS provides error bounds and backward error estimates for the solution to a system of linear equations with a triangular coefficient matrix.

The solution matrix X must be computed by STRTRS/CTRTRS or some other means before entering this routine. STRRFS/CTRRFS does not do iterative refinement because doing so cannot improve the backward error.

Arguments

UPLO (input) CHARACTER*1
= 'U': A is upper triangular;
= 'L': A is lower triangular.

TRANS (input) CHARACTER*1
Specifies the form of the system of equations:
= 'N': $A*X = B$ (No transpose)
= 'T': $A^T*X = B$ (Transpose)
= 'C': $A^H*X = B$ (Conjugate transpose)

DIAG (input) CHARACTER*1
= 'N': A is non-unit triangular;
= 'U': A is unit triangular.

N (input) INTEGER
The order of the matrix A. N $\geq$ 0.

NRHS (input) INTEGER
The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS $\geq$ 0.

A (input) REAL/COMPLEX array, dimension (LDA,N)
The triangular matrix A. If UPLO = 'U', the leading n-by-n upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n-by-n lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If DIAG = 'U', the diagonal elements of A are also not referenced and are assumed to be 1.

LDA (input) INTEGER
The leading dimension of the array A. LDA $\geq$ max(1,N).

B (input) REAL/COMPLEX array, dimension (LDB,NRHS)
The right hand side matrix B.

LDB (input) INTEGER
The leading dimension of the array B. LDB $\geq$ max(1,N).

X (input) REAL/COMPLEX array, dimension (LDX,NRHS)
The solution matrix X.

LDX (input) INTEGER
The leading dimension of the array X. LDX $\geq$ max(1,N).

FERR (output) REAL array, dimension (NRHS)
The estimated forward error bound for each solution vector X(j) (the jth column of the solution matrix X). If XTRUE is the true solution corresponding to X(j), FERR(j) is an estimated upper bound for the magnitude of the largest element in (X(j) − XTRUE) divided by the magnitude of the largest element in X(j). The estimate is as reliable as the estimate for RCOND, and is almost always a slight overestimate of the true error.

BERR (output) REAL array, dimension (NRHS)
The componentwise relative backward error of each solution vector X(j) (i.e., the smallest relative change in any element of A or B that makes X(j) an exact solution).

WORK *STRRFS* (workspace) REAL array, dimension (3*N)
CTRRFS (workspace) COMPLEX array, dimension (2*N)

IWORK *STRRFS only* (workspace) INTEGER array, dimension (N)

RWORK *CTRRFS only* (workspace) REAL array, dimension (N)

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

STRSEN/CTRSEN

```
      SUBROUTINE STRSEN( JOB, COMPQ, SELECT, N, T, LDT, Q, LDQ, WR, WI,
     $                   M, S, SEP, WORK, LWORK, IWORK, LIWORK, INFO )
      CHARACTER          COMPQ, JOB
      INTEGER            INFO, LDQ, LDT, LIWORK, LWORK, M, N
      REAL               S, SEP
      LOGICAL            SELECT( * )
      INTEGER            IWORK( * )
      REAL               Q( LDQ, * ), T( LDT, * ), WI( * ), WORK( * ),
     $                   WR( * )

      SUBROUTINE CTRSEN( JOB, COMPQ, SELECT, N, T, LDT, Q, LDQ, W, M, S,
     $                   SEP, WORK, LWORK, INFO )
      CHARACTER          COMPQ, JOB
      INTEGER            INFO, LDQ, LDT, LWORK, M, N
      REAL               S, SEP
      LOGICAL            SELECT( * )
      COMPLEX            Q( LDQ, * ), T( LDT, * ), W( * ), WORK( * )
```

Purpose

STRSEN/CTRSEN reorders the real-Schur/Schur factorization of a real/complex matrix $A = Q*T*Q^H$, so that a selected cluster of eigenvalues appears in the leading blocks/elements on the diagonal of the upper quasi-triangular/triangular matrix T, and the leading columns of Q form a(n) orthonormal/unitary basis of the corresponding right invariant subspace.

Optionally, the routine computes the reciprocal condition numbers of the cluster of eigenvalues and/or the invariant subspace.

STRSEN only
T must be in Schur canonical form (as returned by SHSEQR), that is, block upper triangular with 1-by-1 and 2-by-2 diagonal blocks; each 2-by-2 diagonal block has its diagonal elements equal and its off-diagonal elements of opposite sign.

Arguments

JOB (input) CHARACTER*1
Specifies whether condition numbers are required for the cluster of eigenvalues (S) or the invariant subspace (SEP):
= 'N': none;
= 'E': for eigenvalues only (S);
= 'V': for invariant subspace only (SEP);
= 'B': for both eigenvalues and invariant subspace (S and SEP).

COMPQ (input) CHARACTER*1
= 'V': update the matrix Q of Schur vectors;
= 'N': do not update Q.

SELECT (input) LOGICAL array, dimension (N)
SELECT specifies the eigenvalues in the selected cluster.
STRSEN
To select a real eigenvalue w(j), SELECT(j) must be set to .TRUE.. To select a complex conjugate pair of eigenvalues w(j) and w(j+1), corresponding to a 2-by-2 diagonal block, either SELECT(j) or SELECT(j+1) or both must be set to .TRUE.; a complex conjugate pair of eigenvalues must be either both included in the cluster or both excluded.
CTRSEN
To select the jth eigenvalue, SELECT(j) must be set to .TRUE..

N (input) INTEGER
The order of the matrix T. $N \geq 0$.

T (input/output) REAL/COMPLEX array, dimension (LDT,N)
STRSEN
On entry, the upper quasi-triangular matrix T, in Schur canonical form. On exit, T is overwritten by the reordered matrix T, again in Schur canonical form, with the selected eigenvalues in the leading diagonal blocks.
CTRSEN
On entry, the upper triangular matrix T.
On exit, T is overwritten by the reordered matrix T, with the selected eigenvalues as the leading diagonal elements.

LDT (input) INTEGER
The leading dimension of the array T. $LDT \geq \max(1,N)$.

Q (input/output) REAL/COMPLEX array, dimension (LDQ,N)
On entry, if COMPQ = 'V', the matrix Q of Schur vectors.
On exit, if COMPQ = 'V', Q has been postmultiplied by the orthogonal/unitary transformation matrix which reorders T; the leading M columns of Q form a(n) orthonormal/unitary basis for the specified invariant subspace.
If COMPQ = 'N', Q is not referenced.

LDQ (input) INTEGER
The leading dimension of the array Q.
$LDQ \geq 1$; and if COMPQ = 'V', $LDQ \geq N$.

WR, WI *STRSEN only* (output) REAL array, dimension (N)
The real and imaginary parts, respectively, of the reordered eigenvalues of T. The eigenvalues are stored in the same order as on the diagonal of T, with WR(i) = T(i,i) and, if T(i:i+1,i:i+1) is a 2-by-2 diagonal block, WI(i) > 0 and WI(i+1) = −WI(i). Note that if a complex eigenvalue is sufficiently ill-conditioned, then its value may differ significantly from its value before reordering.

W *CTRSEN only* (output) COMPLEX array, dimension (N)
The reordered eigenvalues of T, in the same order as they appear on the diagonal of T.

M (output) INTEGER
The dimension of the specified invariant subspace ($0 \leq M \leq N$).

S (output) REAL
If JOB = 'E' or 'B', S is a lower bound on the reciprocal condition number of the selected cluster of eigenvalues. S cannot underestimate the true reciprocal condition number by more than a factor of sqrt(N).
If M = 0 or N, S = 1.
If JOB = 'N' or 'V', S is not referenced.

SEP (output) REAL
If JOB = 'V' or 'B', SEP is the estimated reciprocal condition number of the specified invariant subspace. If M = 0 or N, SEP = $\|T\|$.
If JOB = 'N' or 'E', SEP is not referenced.

WORK (workspace) REAL/COMPLEX array, dimension (LWORK)
CTRSEN only
If JOB = 'N', WORK is not referenced.

LWORK (input) INTEGER
The dimension of the array WORK.
If JOB = 'N', LWORK ≥ max(1,N); (*STRSEN only*)
if JOB = 'N', LWORK ≥ 1; (*CTRSEN only*)
if JOB = 'E', LWORK ≥ max(1,M*(N−M));
if JOB = 'V' or 'B', LWORK ≥ max(1,2*M*(N−M)).

IWORK *STRSEN only* (workspace) INTEGER array, dimension (LIWORK)
If JOB = 'N' or 'E', IWORK is not referenced.

LIWORK *STRSEN only* (input) INTEGER
The dimension of the array IWORK.
If JOB = 'N' or 'E', LIWORK ≥ 1;
if JOB = 'V' or 'B', LIWORK ≥ max(1,M*(N−M)).

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

STRSNA/CTRSNA

```
      SUBROUTINE STRSNA( JOB, HOWMNY, SELECT, N, T, LDT, VL, LDVL, VR,
     $                   LDVR, S, SEP, MM, M, WORK, LDWORK, IWORK,
     $                   INFO )
      CHARACTER          HOWMNY, JOB
      INTEGER            INFO, LDT, LDVL, LDVR, LDWORK, M, MM, N
      LOGICAL            SELECT( * )
      INTEGER            IWORK( * )
      REAL               S( * ), SEP( * ), T( LDT, * ), VL( LDVL, * ),
     $                   VR( LDVR, * ), WORK( LDWORK, * )

      SUBROUTINE CTRSNA( JOB, HOWMNY, SELECT, N, T, LDT, VL, LDVL, VR,
     $                   LDVR, S, SEP, MM, M, WORK, LDWORK, RWORK,
     $                   INFO )
      CHARACTER          HOWMNY, JOB
      INTEGER            INFO, LDT, LDVL, LDVR, LDWORK, M, MM, N
      LOGICAL            SELECT( * )
      REAL               RWORK( * ), S( * ), SEP( * )
      COMPLEX            T( LDT, * ), VL( LDVL, * ), VR( LDVR, * ),
     $                   WORK( LDWORK, * )
```

Purpose

STRSNA/CTRSNA estimates reciprocal condition numbers for specified eigenvalues and/or right eigenvectors of a real/complex upper quasi-triangular/triangular matrix T (or of any matrix $A = Q*T*Q^H$ with Q orthogonal/unitary).

STRSNA only
T must be in Schur canonical form (as returned by SHSEQR), that is, block upper triangular with 1-by-1 and 2-by-2 diagonal blocks; each 2-by-2 diagonal block has its diagonal elements equal and its off-diagonal elements of opposite sign.

Arguments

JOB (input) CHARACTER*1
Specifies whether condition numbers are required for eigenvalues (S) or eigenvectors (SEP):
= 'E': for eigenvalues only (S);
= 'V': for eigenvectors only (SEP);
= 'B': for both eigenvalues and eigenvectors (S and SEP).

HOWMNY (input) CHARACTER*1
= 'A': compute condition numbers for all eigenpairs;
= 'S': compute condition numbers for selected eigenpairs specified by the array SELECT.

SELECT (input) LOGICAL array, dimension (N)
If HOWMNY = 'S', SELECT specifies the eigenpairs for which condition numbers are required.
If HOWMNY = 'A', SELECT is not referenced.
STRSNA
To select condition numbers for the eigenpair corresponding to a real eigenvalue w(j), SELECT(j) must be set to .TRUE.. To select condition numbers corresponding to a complex conjugate pair of eigenvalues w(j) and w(j+1), either SELECT(j) or SELECT(j+1) or both must be set to .TRUE..
CTRSNA
To select condition numbers for the jth eigenpair, SELECT(j) must be set to .TRUE..

N (input) INTEGER
The order of the matrix T. $N \geq 0$.

T (input) REAL/COMPLEX array, dimension (LDT,N)
STRSNA
The upper quasi-triangular matrix T, in Schur canonical form.
CTRSNA
The upper triangular matrix T.

LDT (input) INTEGER
The leading dimension of the array T. $LDT \geq \max(1,N)$.

VL (input) REAL/COMPLEX array, dimension (LDVL,M)
If JOB = 'E' or 'B', VL must contain left eigenvectors of T (or of any matrix $Q*T*Q^H$ with Q orthogonal/unitary), corresponding to the eigenpairs specified by HOWMNY and SELECT. The eigenvectors must be stored in consecutive columns of VL, as returned by STREVC/CTREVC or SHSEIN/CHSEIN.
If JOB = 'V', VL is not referenced.

LDVL (input) INTEGER
The leading dimension of the array VL.
$LDVL \geq 1$; and if JOB = 'E' or 'B', $LDVL \geq N$.

VR (input) REAL/COMPLEX array, dimension (LDVR,M)
If JOB = 'E' or 'B', VR must contain right eigenvectors of T (or of any matrix $Q*T*Q^H$ with Q orthogonal/unitary), corresponding to the eigenpairs specified by HOWMNY and SELECT. The eigenvectors must be stored in consecutive columns of VR, as returned by STREVC/CTREVC or SHSEIN/CHSEIN.
If JOB = 'V', VR is not referenced.

LDVR (input) INTEGER
The leading dimension of the array VR.
$LDVR \geq 1$; and if JOB = 'E' or 'B', $LDVR \geq N$.

S (output) REAL array, dimension (MM)
If JOB = 'E' or 'B', the reciprocal condition numbers of the selected eigenvalues, stored in consecutive elements of the array. Thus S(j), SEP(j), and the jth columns of VL and VR all correspond to the same eigenpair (but not in general the jth eigenpair unless all eigenpairs have been selected).
If JOB = 'V', S is not referenced.
STRSNA only

For a complex conjugate pair of eigenvalues, two consecutive elements of S are set to the same value.

SEP (output) REAL array, dimension (MM)
If JOB = 'V' or 'B', the estimated reciprocal condition numbers of the selected right eigenvectors, stored in consecutive elements of the array.
If JOB = 'E', SEP is not referenced.
STRSNA only
For a complex eigenvector, two consecutive elements of SEP are set to the same value. If the eigenvalues cannot be reordered to compute SEP(j), SEP(j) is set to zero; this can only occur when the true value would be very small anyway.

MM (input) INTEGER
The number of elements in the arrays S (if JOB = 'E' or 'B') and/or SEP (if JOB = 'V' or 'B'). MM $\geq$ M.

M (output) INTEGER
The number of elements of the arrays S and/or SEP actually used to store the estimated condition numbers.
If HOWMNY = 'A', M is set to N.

WORK (workspace) REAL/COMPLEX array, dimension (LDWORK,N+1)
If JOB = 'E', WORK is not referenced.

LDWORK (input) INTEGER
The leading dimension of the array WORK. LDWORK $\geq$ 1; and if JOB = 'V' or 'B', LDWORK $\geq$ N.

IWORK *STRSNA only* (workspace) INTEGER array, dimension (N)
If JOB = 'E', IWORK is not referenced.

RWORK *CTRSNA only* (workspace) REAL array, dimension (N)
If JOB = 'E', RWORK is not referenced.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

STRSYL/CTRSYL

```
      SUBROUTINE STRSYL( TRANA, TRANB, ISGN, M, N, A, LDA, B, LDB, C,
     $                   LDC, SCALE, INFO )
      CHARACTER          TRANA, TRANB
      INTEGER            INFO, ISGN, LDA, LDB, LDC, M, N
      REAL               SCALE
      REAL               A( LDA, * ), B( LDB, * ), C( LDC, * )
```

```
      SUBROUTINE CTRSYL( TRANA, TRANB, ISGN, M, N, A, LDA, B, LDB, C,
     $                   LDC, SCALE, INFO )
      CHARACTER          TRANA, TRANB
      INTEGER            INFO, ISGN, LDA, LDB, LDC, M, N
      REAL               SCALE
      COMPLEX            A( LDA, * ), B( LDB, * ), C( LDC, * )
```

Purpose

STRSYL/CTRSYL solves the real/complex Sylvester matrix equation:

$$op(A)*X \pm X*op(B) = scale*C,$$

where op(A) = A or A^H, and A and B are both upper quasi-triangular/triangular. A is m-by-m and B is n-by-n; the right hand side C and the solution X are m-by-n; and scale is an output scale factor, set $\leq$ 1 to avoid overflow in X.

STRSYL only
A and B must be in Schur canonical form (as returned by SHSEQR), that is, block upper triangular with 1-by-1 and 2-by-2 diagonal blocks; each 2-by-2 diagonal block has its diagonal elements equal and its off-diagonal elements of opposite sign.

Arguments

TRANA (input) CHARACTER*1
Specifies the option op(A):
= 'N': op(A) = A (No transpose)
= 'T': op(A) = A^T (Transpose) (*STRSYL only*)
= 'C': op(A) = A^H (Conjugate transpose)

TRANB (input) CHARACTER*1
Specifies the option op(B):
= 'N': op(B) = B (No transpose)
= 'T': op(B) = B^T (Transpose) (*STRSYL only*)
= 'C': op(B) = B^H (Conjugate transpose)

ISGN (input) INTEGER
Specifies the sign in the equation:
= +1: solve op(A)*X + X*op(B) = scale*C
= −1: solve op(A)*X − X*op(B) = scale*C

M (input) INTEGER
The order of the matrix A, and the number of rows in the matrices X and C. M $\geq$ 0.

N (input) INTEGER
The order of the matrix B, and the number of columns in the matrices X and C. N $\geq$ 0.

A (input) REAL/COMPLEX array, dimension (LDA,M)
STRSYL
The upper quasi-triangular matrix A, in Schur canonical form.
CTRSYL
The upper triangular matrix A.

LDA (input) INTEGER
The leading dimension of the array A. LDA $\geq$ max(1,M).

B (input) REAL/COMPLEX array, dimension (LDB,N)
STRSYL
The upper quasi-triangular matrix B, in Schur canonical form.
CTRSYL
The upper triangular matrix B.

LDB (input) INTEGER
The leading dimension of the array B. LDB $\geq$ max(1,N).

C (input/output) REAL/COMPLEX array, dimension (LDC,N)
On entry, the m-by-n right hand side matrix C.
On exit, C is overwritten by the solution matrix X.

LDC (input) INTEGER
The leading dimension of the array C. LDC $\geq$ max(1,M)

SCALE (output) REAL
The scale factor, scale, set ≤ 1 to avoid overflow in X.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = $-$i, the ith argument had an illegal value.
= 1: A and B have common or very close eigenvalues; perturbed values were used to solve the equation (but the matrices A and B are unchanged).

STRTRI/CTRTRI

```
SUBROUTINE STRTRI( UPLO, DIAG, N, A, LDA, INFO )
CHARACTER          DIAG, UPLO
INTEGER            INFO, LDA, N
REAL               A( LDA, * )

SUBROUTINE CTRTRI( UPLO, DIAG, N, A, LDA, INFO )
CHARACTER          DIAG, UPLO
INTEGER            INFO, LDA, N
COMPLEX            A( LDA, * )
```

Purpose

STRTRI/CTRTRI computes the inverse of a real/complex upper or lower triangular matrix A.

Arguments

UPLO (input) CHARACTER*1
= 'U': A is upper triangular;
= 'L': A is lower triangular.

DIAG (input) CHARACTER*1
= 'N': A is non-unit triangular;
= 'U': A is unit triangular.

N (input) INTEGER
The order of the matrix A. N $\geq$ 0.

A (input/output) REAL/COMPLEX array, dimension (LDA,N)
On entry, the triangular matrix A. If UPLO = 'U', the leading n-by-n upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n-by-n lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If DIAG = 'U', the diagonal elements of A are also not referenced and are assumed to be 1.
On exit, the (triangular) inverse of the original matrix, in the same storage format.

LDA (input) INTEGER
The leading dimension of the array A. LDA $\geq$ max(1,N).

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = $-$i, the ith argument had an illegal value.
> 0: if INFO = i, A(i,i) is exactly zero. The triangular matrix is singular and its inverse can not be computed.

STRTRS/CTRTRS

```
SUBROUTINE STRTRS( UPLO, TRANS, DIAG, N, NRHS, A, LDA, B, LDB,
$                  INFO )
CHARACTER          DIAG, TRANS, UPLO
INTEGER            INFO, LDA, LDB, N, NRHS
REAL               A( LDA, * ), B( LDB, * )

SUBROUTINE CTRTRS( UPLO, TRANS, DIAG, N, NRHS, A, LDA, B, LDB,
$                  INFO )
CHARACTER          DIAG, TRANS, UPLO
INTEGER            INFO, LDA, LDB, N, NRHS
COMPLEX            A( LDA, * ), B( LDB, * )
```

Purpose

STRTRS/CTRTRS solves a triangular system of the form A*X = B, A^T*X = B, or A^H*X = B, where A is a triangular matrix of order n, and B is an n-by-nrhs matrix. A check is made to verify that A is nonsingular.

Arguments

UPLO (input) CHARACTER*1
= 'U': A is upper triangular;
= 'L': A is lower triangular.

TRANS (input) CHARACTER*1
Specifies the form of the system of equations:
= 'N': A*X = B (No transpose)
= 'T': A^T*X = B (Transpose)
= 'C': A^H*X = B (Conjugate transpose)

DIAG (input) CHARACTER*1
= 'N': A is non-unit triangular;
= 'U': A is unit triangular.

N (input) INTEGER
The order of the matrix A. N $\geq$ 0.

NRHS (input) INTEGER
The number of right hand sides, i.e., the number of columns of the matrix B. NRHS $\geq$ 0.

A (input) REAL/COMPLEX array, dimension (LDA,N)
The triangular matrix A. If UPLO = 'U', the leading n-by-n upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading n-by-n lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If DIAG = 'U', the diagonal elements of A are also not referenced and are assumed to be 1.

LDA (input) INTEGER
The leading dimension of the array A. LDA $\geq$ max(1,N).

B (input/output) REAL/COMPLEX array, dimension (LDB,NRHS)
On entry, the right hand side matrix B.
On exit, if INFO = 0, the solution matrix X.

LDB (input) INTEGER
The leading dimension of the array B. LDB $\geq$ max(1,N).

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = $-$i, the ith argument had an illegal value.
> 0: if INFO = i, the ith diagonal element of A is zero, indicating that the matrix is singular and the solutions X have not been computed.

STZRZF/CTZRZF

```
SUBROUTINE STZRZF( M, N, A, LDA, TAU, WORK, LWORK, INFO )
INTEGER            INFO, LDA, LWORK, M, N
REAL               A( LDA, * ), TAU( * ), WORK( LWORK )

SUBROUTINE CTZRZF( M, N, A, LDA, TAU, WORK, LWORK, INFO )
INTEGER            INFO, LDA, LWORK, M, N
COMPLEX            A( LDA, * ), TAU( * ), WORK( LWORK )
```

Purpose

STZRZF/CTZRZF reduces the m-by-n (m $\leq$ n) real/complex upper trapezoidal matrix A to upper triangular form by means of orthogonal/unitary transformations.

The upper trapezoidal matrix A is factorized as

$$A = \begin{pmatrix} R & 0 \end{pmatrix} * Z,$$

where Z is an n-by-n orthogonal/unitary matrix and R is an m-by-m upper triangular matrix.

Arguments

M (input) INTEGER
The number of rows of the matrix A. M $\geq$ 0.

N (input) INTEGER
The number of columns of the matrix A. N $\geq$ M.

A (input/output) REAL/COMPLEX array, dimension (LDA,N)
On entry, the leading m-by-n upper trapezoidal part of the array A must contain the matrix to be factorized.
On exit, the leading m-by-m upper triangular part of A contains the upper triangular matrix R, and elements m+1 to n of the first m rows of A, with the array TAU, represent the orthogonal/unitary matrix Z as a product of m elementary reflectors.

LDA (input) INTEGER
The leading dimension of the array A. LDA $\geq$ max(1,M).

TAU (output) REAL/COMPLEX array, dimension (M)
The scalar factors of the elementary reflectors.

WORK (workspace/output) REAL/COMPLEX array, dimension (LWORK)
On exit, if INFO = 0, WORK(1) returns the optimal WORK.

LWORK (input) INTEGER
The dimension of the array WORK. LWORK $\geq$ max(1,M).
For optimum performance LWORK $\geq$ M*NB, where NB is the optimal blocksize.

If LWORK = $-$1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the

first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

INFO (output) INTEGER
= 0: successful exit
< 0: if INFO = −i, the ith argument had an illegal value.

Bibliography

[1] E. ANDERSON, Z. BAI, C. BISCHOF, J. DEMMEL, J. DONGARRA, J. DU CROZ, A. GREENBAUM, S. HAMMARLING, A. MCKENNEY, AND D. SORENSEN, *LAPACK: A portable linear algebra library for high-performance computers*, Computer Science Dept. Technical Report CS-90-105, University of Tennessee, Knoxville, TN, May 1990. (Also LAPACK Working Note #20).

[2] E. ANDERSON, Z. BAI, AND J. DONGARRA, *Generalized QR factorization and its applications*, Linear Algebra and Its Applications, 162-164 (1992), pp. 243–271. (Also LAPACK Working Note #31).

[3] E. ANDERSON, J. DONGARRA, AND S. OSTROUCHOV, *Installation guide for LAPACK*, Computer Science Dept. Technical Report CS-92-151, University of Tennessee, Knoxville, TN, March 1992. (Also LAPACK Working Note #41).

[4] ANSI/IEEE, *IEEE Standard for Binary Floating Point Arithmetic*, New York, Std 754-1985 ed., 1985.

[5] ——, *IEEE Standard for Radix Independent Floating Point Arithmetic*, New York, Std 854-1987 ed., 1987.

[6] M. ARIOLI, J. W. DEMMEL, AND I. S. DUFF, *Solving sparse linear systems with sparse backward error*, SIAM J. Matrix Anal. Appl., 10 (1989), pp. 165–190.

[7] M. ARIOLI, I. S. DUFF, AND P. P. M. DE RIJK, *On the augmented system approach to sparse least squares problems*, Num. Math., 55 (1989), pp. 667–684.

[8] Z. BAI, , AND H. ZHA, *A new preprocessing algorithm for the computation of the generalized singular value decomposition*, SIAM J. Sci. Comp., 14 (1993), pp. 1007–1012.

[9] Z. BAI AND J. W. DEMMEL, *On a block implementation of Hessenberg multishift QR iteration*, International Journal of High Speed Computing, 1 (1989), pp. 97–112. (Also LAPACK Working Note #8).

[10] ——, *Computing the generalized singular value decomposition*, SIAM J. Sci. Comp., 14 (1993), pp. 1464–1486. (Also LAPACK Working Note #46).

[11] ——, *Design of a parallel nonsymmetric eigenroutine toolbox, Part I*, in Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing, R. F. *et al.* Sincovec,

ed., Philadelphia, PA, 1993, Society for Industrial and Applied Mathematics, pp. 391–398. Long version available as Computer Science Report CSD-92-718, University of California, Berkeley, 1992.

[12] Z. Bai, J. W. Demmel, and A. McKenney, *On computing condition numbers for the nonsymmetric eigenproblem*, ACM Trans. Math. Softw., 19 (1993), pp. 202–223. (LAPACK Working Note #13).

[13] Z. Bai and M. Fahey, *Computation of error bounds in linear least squares problems with equality constraints and generalized linear model problems.* to appear, 1997.

[14] J. Barlow and J. Demmel, *Computing accurate eigensystems of scaled diagonally dominant matrices*, SIAM J. Num. Anal., 27 (1990), pp. 762–791. (Also LAPACK Working Note #7).

[15] J. Bilmes, K. Asanovic, J. Demmel, D. Lam, and C. Chin, *Optimizing matrix multiply using PHiPAC: A portable, high-performance, ANSI C coding methodology*, Computer Science Dept. Technical Report CS-96-326, University of Tennessee, Knoxville, TN, 1996. (Also LAPACK Working Note #111).

[16] Å. Björck, *Numerical Methods for Least Squares Problem*, SIAM, 1996.

[17] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley, *ScaLAPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1997.

[18] A. J. Cox and N. J. Higham, *Backward error bounds for constrained least squares problems*, BIT, 39 (1999), pp. 210–227.

[19] C. R. Crawford, *Reduction of a band-symmetric generalized eigenvalue problem*, Comm. ACM, 16 (1973), pp. 41–44.

[20] J. J. M. Cuppen, *A divide and conquer method for the symmetric tridiagonal eigenproblem*, Numerische Math., 36 (1981), pp. 177–195.

[21] M. Dayde, I. Duff, and A. Petitet, *A Parallel Block Implementation of Level 3 BLAS for MIMD Vector Processors*, ACM Trans. Math. Softw., 20 (1994), pp. 178–193.

[22] B. De Moor and P. Van Dooren, *Generalization of the singular value and QR decompositions*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 993–1014.

[23] P. Deift, J. W. Demmel, L.-C. Li, and C. Tomei, *The bidiagonal singular values decomposition and Hamiltonian mechanics*, SIAM J. Numer. Anal., 28 (1991), pp. 1463–1516. (LAPACK Working Note #11).

[24] J. Demmel, *Underflow and the reliability of numerical software*, SIAM J. Sci. Stat. Comput., 5 (1984), pp. 887–919.

[25] ——, *Applied Numerical Linear Algebra*, SIAM, Philadelphia, PA, 1997.

[26] J. W. DEMMEL, *The condition number of equivalence transformations that block diagonalize matrix pencils*, SIAM J. Numer. Anal., 20 (1983), pp. 599–610.

[27] J. W. DEMMEL AND N. J. HIGHAM, *Stability of block algorithms with fast level 3 BLAS*, ACM Trans. Math. Softw., 18 (1992), pp. 274–291. (Also LAPACK Working Note #22).

[28] ——, *Improved error bounds for underdetermined systems solvers*, SIAM J. Matrix Anal. Appl., 14 (1993), pp. 1–14. (Also LAPACK Working Note #23).

[29] J. W. DEMMEL AND B. KÅGSTRÖM, *Computing stable eigendecompositions of matrix pencils*, Lin. Alg. Appl., 88/89 (1987), pp. 139–186.

[30] J. W. 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. Softw., 19 (1993), pp. 160–174.

[31] ——, *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. Softw., 19 (1993), pp. 175–201.

[32] J. W. DEMMEL AND W. KAHAN, *Accurate singular values of bidiagonal matrices*, SIAM J. Sci. Stat. Comput., 11 (1990), pp. 873–912. (Also LAPACK Working Note #3).

[33] J. W. DEMMEL AND X. LI, *Faster numerical algorithms via exception handling*, IEEE Trans. Comp., 43 (1994), pp. 983–992. (Also LAPACK Working Note #59).

[34] J. W. DEMMEL AND K. VESELIĆ, *Jacobi's method is more accurate than QR*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 1204–1246. (Also LAPACK Working Note #15).

[35] I. DHILLON, *A new $O(n^2)$ algorithm for the symmetric tridiagonal eigenvalue/eigenvector problem*, Computer Science Division Technical Report no. UCB/CSD-97-971, University of California, Berkeley, CA, May 1997.

[36] I. S. DHILLON AND B. N. PARLETT, *Orthogonal eigenvectors and relative gaps*, June 1999. to appear.

[37] J. DONGARRA AND S. OSTROUCHOV, *Quick installation guide for LAPACK on unix systems*, Computer Science Dept. Technical Report CS-94-249, University of Tennessee, Knoxville, TN, September 1994. (LAPACK Working Note #81).

[38] J. J. DONGARRA, J. R. BUNCH, C. B. MOLER, AND G. W. STEWART, *LINPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1979.

[39] J. J. DONGARRA, J. DU CROZ, I. S. DUFF, AND S. HAMMARLING, *Algorithm 679: A set of Level 3 Basic Linear Algebra Subprograms*, ACM Trans. Math. Soft., 16 (1990), pp. 18–28.

[40] ——, *A set of Level 3 Basic Linear Algebra Subprograms*, ACM Trans. Math. Soft., 16 (1990), pp. 1–17.

[41] J. J. DONGARRA, J. DU CROZ, S. HAMMARLING, AND R. J. HANSON, *Algorithm 656: An extended set of FORTRAN Basic Linear Algebra Subroutines*, ACM Trans. Math. Soft., 14 (1988), pp. 18–32.

[42] ——, *An extended set of FORTRAN basic linear algebra subroutines*, ACM Trans. Math. Soft., 14 (1988), pp. 1–17.

[43] J. J. DONGARRA, I. S. DUFF, D. C. SORENSEN, AND H. A. VAN DER VORST, *Numerical Linear Algebra for High-Performance Computers*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1998.

[44] J. J. DONGARRA AND E. GROSSE, *Distribution of mathematical software via electronic mail*, Communications of the ACM, 30 (1987), pp. 403–407.

[45] J. J. DONGARRA, F. G. GUSTAFSON, AND A. KARP, *Implementing linear algebra algorithms for dense matrices on a vector pipeline machine*, SIAM Review, 26 (1984), pp. 91–112.

[46] J. J. DONGARRA, S. HAMMARLING, AND D. C. SORENSEN, *Block reduction of matrices to condensed forms for eigenvalue computations*, JCAM, 27 (1989), pp. 215–227. (LAPACK Working Note #2).

[47] J. DU CROZ AND N. J. HIGHAM, *Stability of methods for matrix inversion*, IMA J. Numer. Anal., 12 (1992), pp. 1–19. (Also LAPACK Working Note #27).

[48] J. DU CROZ, P. J. D. MAYES, AND G. RADICATI DI BROZOLO, *Factorizations of band matrices using Level 3 BLAS*, Computer Science Dept. Technical Report CS-90-109, University of Tennessee, Knoxville, TN, 1990. (LAPACK Working Note #21).

[49] A. DUBRULLE, *The multishift QR algorithm: is it worth the trouble?*, Palo Alto Scientific Center Report G320-3558x, IBM Corp., 1530 Page Mill Road, Palo Alto, CA 94304, 1991.

[50] L. ELDÉN, *Perturbation theory for the least squares problem with linear equality constraints*, SIAM J. Numer. Anal., 17 (1980), pp. 338–350.

[51] V. FERNANDO AND B. PARLETT, *Accurate singular values and differential qd algorithms*, Numerisch Math., 67 (1994), pp. 191–229.

[52] K. A. GALLIVAN, R. J. PLEMMONS, AND A. H. SAMEH, *Parallel algorithms for dense linear algebra computations*, SIAM Review, 32 (1990), pp. 54–135.

[53] F. GANTMACHER, *The Theory of Matrices, vol. II (transl.)*, Chelsea, New York, 1959.

[54] B. S. GARBOW, J. M. BOYLE, J. J. DONGARRA, AND C. B. MOLER, *Matrix Eigensystem Routines – EISPACK Guide Extension*, vol. 51 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1977.

[55] G. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, Johns Hopkins University Press, Baltimore, MD, third ed., 1996.

[56] A. GREENBAUM AND J. J. DONGARRA, *Experiments with QL/QR methods for the symmetric tridiagonal eigenproblem*, Computer Science Dept. Technical Report CS-89-92, University of Tennessee, Knoxville,TN, 1989. (LAPACK Working Note #17).

[57] M. GU AND S. EISENSTAT, *A stable algorithm for the rank-1 modification of the symmetric eigenproblem*, Computer Science Department Report YALEU/DCS/RR-916, Yale University, New Haven, CT, 1992.

[58] ——, *A divide-and-conquer algorithm for the bidiagonal SVD*, SIAM J. Mat. Anal. Appl., 16 (1995), pp. 79–92.

[59] W. W. HAGER, *Condition estimators*, SIAM J. Sci. Stat. Comput., 5 (1984), pp. 311–316.

[60] S. HAMMARLING, *The numerical solution of the general Gauss-Markov linear model*, in Mathematics in Signal Processing, T. S. *et al.*. Durani, ed., Clarendon Press, Oxford, UK, 1986.

[61] N. J. HIGHAM, *Efficient algorithms for computing the condition number of a tridiagonal matrix*, SIAM J. Sci. Stat. Comput., 7 (1986), pp. 150–165.

[62] ——, *A survey of condition number estimation for triangular matrices*, SIAM Review, 29 (1987), pp. 575–596.

[63] ——, *FORTRAN codes for estimating the one-norm of a real or complex matrix, with applications to condition estimation*, ACM Trans. Math. Softw., 14 (1988), pp. 381–396.

[64] ——, *Algorithm 674: FORTRAN codes for estimating the one-norm of a real or complex matrix, with applications to condition estimation*, ACM Trans. Math. Softw., 15 (1989), p. 168.

[65] ——, *Experience with a matrix norm estimator*, SIAM J. Sci. Stat. Comput., 11 (1990), pp. 804–809.

[66] ——, *Perturbation theory and backward error for $AX - XB = C$*, BIT, 33 (1993), pp. 124–136.

[67] ——, *Accuracy and Stability of Numerical Algorithms*, SIAM, Philadelphia, PA, 1996.

[68] S. HUSS-LEDERMAN, A. TSAO, AND G. ZHANG, *A parallel implementation of the invariant subspace decomposition algorithm for dense symmetric matrices*, in Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing, Society for Industrial and Applied Mathematics, 1993, pp. 367–374.

[69] E. JESSUP AND D. SORENSEN, *A parallel algorithm for computing the singular value decomposition of a matrix*, Mathematics and Computer Science Division Report ANL/MCS-TM-102, Argonne National Laboratory, Argonne, IL, December 1987.

[70] B. KÅGSTRÖM, *A direct method for reordering eigenvalues in the generalized real Schur form of a regular matrix pair (a,b)*, in Linear Algebra for Large Scale and Real-Time Applications, Kluwer Academic Publishers, 1993, pp. 195–218.

[71] ——, *A perturbation analysis of the generalized sylvester equation*, SIAM J. Matrix Anal. Appl., 15 (1994), pp. 1045–1060.

[72] B. KÅGSTRÖM, P. LING, AND C. V. LOAN, *GEMM-based level 3 BLAS: High-performance model implementations and performance evaluation benchmark*, Tech. Rep. UMINF 95-18, Department of Computing Science, Umeå University, 1995. Submitted to ACM Trans. Math. Softw.

[73] B. KÅGSTRÖM AND P. POROMAA, *Computing eigenspaces with specified eigenvalues of a regular matrix pair (A, B) and condition estimation: Theory, algorithms and software*, Tech. Rep. UMINF 94.04, Department of Computing Science, Umeå University, 1994.

[74] B. KÅGSTRÖM AND P. POROMAA, *LAPACK-style algorithms and software for solving the generalized Sylvester equation and estimating the separation between regular matrix pairs*, ACM Trans. Math. Softw., 22 (1996), pp. 78–103.

[75] B. KÅGSTRÖM AND L. WESTIN, *Generalized schur methods with condition estimators for solving the generalized Sylvester equation*, IEEE Trans. Autom. Contr., 34 (1989), pp. 745–751.

[76] T. KATO, *Perturbation Theory for Linear Operators*, Springer-Verlag, Berlin, 2 ed., 1980.

[77] L. KAUFMAN, *Banded eigenvalue solvers on vector machines*, ACM Trans. Math. Softw., 10 (1984), pp. 73–86.

[78] C. L. LAWSON, R. J. HANSON, D. KINCAID, AND F. T. KROGH, *Basic linear algebra subprograms for Fortran usage*, ACM Trans. Math. Soft., 5 (1979), pp. 308–323.

[79] R. LEHOUCQ, *The computation of elementary unitary matrices*, Computer Science Dept. Technical Report CS-94-233, University of Tennessee, Knoxville, TN, 1994. (Also LAPACK Working Note 72).

[80] C. PAIGE, *Computer solution and perturbation analysis of generalized linear least squares problems*, Math. of Comput., 33 (1979), pp. 171–183.

[81] ——, *Fast numerically stable computations for generalized linear least squares problems controllability*, SIAM J. Num. Anal., 16 (1979), pp. 165–179.

[82] ——, *A note on a result of sun ji-guang: sensitivity of the cs and gsv decomposition*, SIAM J. Num. Anal., 21 (1984), pp. 186–191.

[83] ——, *Computing the generalized singular value decomposition*, SIAM J. Sci. Stat., 7 (1986), pp. 1126–1146.

[84] ——, *Some aspects of generalized QR factorization*, in Reliable Numerical Computations, M. Cox and S. Hammarling, eds., Clarendon Press, 1990.

[85] B. PARLETT, *The Symmetric Eigenvalue Problem*, Prentice-Hall, Englewood Cliffs, NJ, 1980.

[86] B. N. PARLETT AND I. S. DHILLON, *Relatively robust representation of symmetric tridiagonals*, June 1999. to appear.

[87] B. N. PARLETT AND O. A. MARQUES, *An implementation of the dqds algorithm (positive case)*, June 1999. to appear.

[88] E. POLLICINI, A. A., *Using Toolpack Software Tools*, 1989.

[89] J. RUTTER, *A serial implementation of cuppen's divide and conquer algorithm for the symmetric tridiagonal eigenproblem*, Computer Science Division Report UCB/CSD 94/799, University of California, Berkeley, Berkeley, CA, 1994. (Also LAPACK Working Note 69).

[90] R. SCHREIBER AND C. F. VAN LOAN, *A storage efficient WY representation for products of Householder transformations*, SIAM J. Sci. Stat. Comput., 10 (1989), pp. 53–57.

[91] I. SLAPNIČAR, *Accurate symmetric eigenreduction by a Jacobi method*, PhD thesis, Fernuniversität - Hagen, Hagen, Germany, 1992.

[92] B. T. SMITH, J. M. BOYLE, J. J. DONGARRA, B. S. GARBOW, Y. IKEBE, V. C. KLEMA, AND C. B. MOLER, *Matrix Eigensystem Routines – EISPACK Guide*, vol. 6 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1976.

[93] G. W. STEWART, *On the sensitivity of the eigenvalue problem $Ax = \lambda Bx$*, SIAM J. Num. Anal., 9 (1972), pp. 669–686.

[94] ——, *Error and perturbation bounds for subspaces associated with certain eigenvalue problems*, SIAM Review, 15 (1973), pp. 727–764.

[95] G. W. STEWART AND J.-G. SUN, *Matrix Perturbation Theory*, Academic Press, New York, 1990.

[96] J. G. SUN, *Perturbation analysis for the generalized singular value problem*, SIAM J. Num. Anal., 20 (1983), pp. 611–625.

[97] P. VAN DOOREN, *The computation of Kronecker's canonical form of a singular pencil*, Lin. Alg. Appl., 27 (1979), pp. 103–141.

[98] J. VARAH, *On the separation of two matrices*, SIAM J. Numer. Anal., 16 (1979), pp. 216–222.

[99] K. VESELIĆ AND I. SLAPNIČAR, *Floating-point perturbations of Hermitian matrices*, Linear Algebra and Appl., 195 (1993), pp. 81–116.

[100] R. C. WARD, *Balancing the generalized eigenvalue problem*, SIAM J. Sci. Stat. Comput., 2 (1981), pp. 141–152.

[101] D. WATKINS AND L. ELSNER, *Convergence of algorithms of decomposition type for the eigenvalue problem*, Linear Algebra Appl., 143 (1991), pp. 19–47.

[102] R. C. WHALEY AND J. DONGARRA, *Automatically Tuned Linear Algebra Software*. http://www.supercomp.org/sc98/TechPapers/sc98_FullAbstracts/Whaley814/INDEX.HTM, 1998. Winner, best paper in the systems category, SC98: High Performance Networking and Computing.

[103] J. H. WILKINSON, *The Algebraic Eigenvalue Problem*, Oxford University Press, Oxford, UK, 1965.

[104] ——, *Some recent advances in numerical linear algebra*, in The State of the Art in Numerical Analysis, D. A. H. Jacobs, ed., Academic Press, New York, 1977.

[105] ——, *Kronecker's canonical form and the QZ algorithm*, Lin. Alg. Appl., 28 (1979), pp. 285–303.

[106] J. H. WILKINSON AND C. REINSCH, eds., *Handbook for Automatic Computation, vol 2.: Linear Algebra*, Springer-Verlag, Heidelberg, 1971.

Index by Keyword

absolute error, 80, 81
absolute gap, 104, 113, 114, 117
accuracy, 77, 90, 114
    high, 87, 104, 105, 113, 114
angle between vectors and subspaces, 82–84, 94, 103, 106, 111, 115, 118, 119, 130
arguments
    ABNRM, 109, 124
    arrays, 136
    BALANC, 109, 124
    BBNRM, 124
    BERR, 91
    description conventions, 135
    DIAG, 142
    dimensions, 136
    FERR, 89, 91
    ILO and IHI, 43, 109, 124
    INFO, 137, 152
    LDA, 136, 152
    LSCALE, 124
    LWORK, 137
    options, 136
    order of, 134
    RANK, 94
    RCOND, 82
    RCONDE, 107, 110, 121, 126
    RCONDV, 107, 111, 122, 127
    RSCALE, 124
    SCALE, 109
    UPLO, 136
    work space, 137
ARPACK, xvi, xx
auxiliary routines, 11
Auxiliary Routines, index of: see Appendix B, 169
avoiding poor performance, 154

backward error, 85, 86, 89–91, 93, 105
backward stability, 85, 107, 112, 118, 121, 131
    componentwise, 87, 114
    normwise, 86
balancing and conditioning, eigenproblems, 50
balancing of eigenproblems, 109, 124
basis, orthonormal, 19
bidiagonal form, 45, 64, 144
BLAS, 4, 55, 154
    Level 1, 56
    Level 2, 56, 58
    Level 3, 56, 58
    Level 3, fast, 77, 132
    quick reference guide: see Appendix C, 180
block algorithm, 56
block size, 148
    determination of, 138
    from ILAENV, 138
    tuning ILAENV, 138
block width, 58
blocked algorithms, performance, 59
bug reports, 7
    checklist, 150
    mailing alias, 150

cache, 54, 55
CAPSS, xvi, xx
Cholesky factorization
    blocked form, 58
    split, 47
chordal distance, 118, 119
CLAPACK, 8
cluster
    eigenvalues, 105
      error bound, 107
    generalized eigenvalues

error bound, 122
singular values, 113
commercial use, 6
complete orthogonal factorization, 34
computational routines, 11, 25
Computational Routines, index of: see Appendix A, 156
condensed form
reduction to, 64
condition number, 43, 44, 51, 82, 86, 88, 90, 92, 93, 107–109, 116–118, 121, 124, 130
estimate, 87, 90, 110, 127
Cosine-Sine decomposition, 24
Crawford's algorithm, 47
crossover point, 147
Cuppen's divide and conquer algorithm, 40
cycle time, 54

data movement, 55
debugging
release_notes, 151
debugging hints
installation, 151
deflating subspaces, 22, 49, 51
error bound, 122
diagonal pivoting method, 165, 166, 177, 178
distributed memory, xvi, xx
divide and conquer, 21
least squares, 16
SVD, 20
documentation
installation guide, 150
documentation, structure, 134
driver routine
generalized least squares, 16
generalized nonsymmetric eigenvalue problem, 21, 22
generalized SVD, 23
generalized symmetric definite eigenvalue problem, 20
linear equations, 14
linear least squares, 16
nonsymmetric eigenvalue problem, 19
symmetric tridiagonal eigenvalue problem, 12
driver routines, 11, 14
divide and conquer, 16, 20, 21
expert, 14, 21
simple, 14, 21
Driver Routines, index of: see Appendix A, 156

effective rank, 33
efficiency, 56
eigendecomposition
blocked form, 63
multishift QR iteration, 67
symmetric, 103
eigenvalue, 17, 18, 39, 42, 103
error bound, 103, 104, 106, 107, 109, 115–117, 119, 125
generalized, 48
ordering of, 23
GNEP, 21
GSEP, 21
infinite, 48
NEP, 18
nontrivial, 25
ordering of, 19, 43, 51
sensitivity of, 43, 44
generalized, 51
SEP, 39
trivial, 25
eigenvalue problem
ill-conditioned, 22
singular, 22, 128
eigenvector, 17, 39, 103
error bound, 103, 104, 106, 107, 109, 115–117, 119, 125
GNEP, 21
left, 22
right, 22
GSEP, 21
left, 18, 42, 50
generalized, 48
NEP, 18
right, 18, 42, 50
generalized, 48

SEP, 39
EISPACK, 4, 139, 142, 149
converting from: see Appendix D, 185
elementary Householder matrix, see Householder matrix, 31, 32, 34, 45, 143
elementary reflector, see Householder matrix, 31, 34, 45, 143
equality-constrained least squares, 17
equilibration, 14, 27
errata, 8
error
absolute, 80, 81
analysis, 85
backward, 85, 86, 89–91, 93, 105
measurement of, 80
matrix, 80, 81
scalar, 80
subspace, 80, 82
vector, 80
relative, 78, 80, 81, 83, 86, 90
error bounds, 77
clustered eigenvalues, 107, 121
generalized nonsymmetric eigenproblem, 119, 121
generalized singular value decomposition, 129–131
generalized symmetric definite eigenproblem, 114, 117
linear equations, 88, 89
linear least squares, 91, 93
nonsymmetric eigenproblem, 106, 107
required for fast Level 3 BLAS, 132
singular value decomposition, 111, 112
symmetric eigenproblem, 103, 104
error handler, XERBLA, 138, 152

failures, 152
common causes, 151
error handling, 137
INFO, 137
FAQ
LAPACK, 3
floating-point arithmetic, 78
guard digit, 80
IEEE standard, 79
infinity, 80
machine precision, 78
NaN, 80
Not-a-Number, 80
overflow, 78–80, 82, 90, 94, 104, 113
roundoff error, 78
underflow, 78–80
forward error, 28
forward stability, 87
componentwise relative, 87

gap, 104, 105, 113, 114, 117
general linear model problem, 36
generalized eigenproblem
nonsymmetric, 21
symmetric definite, 46
symmetric definite banded, 47
generalized eigenvalue
error bound, 122
generalized eigenvector
error bound, 122
generalized Hessenberg form
reduction to, 49
generalized least squares, 16
generalized orthogonal factorization, 35
generalized Schur vectors, 22
generalized singular value, 23
generalized singular value decomposition, 23, 51
special cases, 24
Givens rotation, 39, 45, 47
GLM, 17, 36, 159
GNEP, 21
GQR, 17, 34–36
GRQ, 17, 37
GSEP, 20
GSVD, 23, 25, 51
guard digit, 80
GUPTRI, 129

Hessenberg form, 42, 64
reduction to, 64
upper, 42
Householder matrix, 63, 143
complex, 143

Householder transformation - blocked form, 63
Householder vector, 143

ill-conditioned, 86
ill-posed, 86
infinity, 80
INFO, 137, 152
input error, 77
installation, 6
  ILAENV, 146
  LAPACK, 145
  xLAMCH, 78, 146
    cost of, 155
installation guide, 145
invariant subspaces, 22, 43, 108
  error bound, 105, 107
inverse iteration, 40, 42
iterative refinement, 28, 132

JLAPACK, 8

LAPACK
  commercial use of, 6
  prebuilt libraries, availability of, 5
LAPACK++, xx
LAPACK90, 8
LDL^T factorization, 26
linear equations, 25
linear least squares problem, 15, 31–33
  generalized, 16
    equality-constrained (LSE), 17, 38
    regression model (GLM), 17
  overdetermined, 91
  overdetermined system, 94
  rank-deficient, 32, 34, 46
  regularization, 94
  underdetermined, 94
  weighted, 17
linear systems, solution of, 14, 25
LINPACK, 4, 56, 139
  converting from: see Appendix D, 185
LLS, 15, 16
local memory, 55
LQ factorization, 32
LSE, 17, 38, 159
LU factorization, 26
  blocked form, 60
  matrix types, 26
LWORK, 137
LWORK query, 137

machine parameters, 146
machine precision, 78
matrix inversion, 14
minimum norm least squares solution, 16
minimum norm solution, 15, 32, 34, 46
multishift QR algorithm, tuning, 149

naming scheme, 12
  auxiliary, 13
  driver and computational, 12
NaN, 80
NEP, 18
netlib, 5
  mirror repositories, 5
nonsymmetric eigenproblem, 42
  generalized, 21, 48
norm
  Frobenius norm, 81, 84
  infinity norm, 81
  matrix, 81
  one norm, 81
  two norm, 81, 84
  vector, 80, 81
normalization, 48
Not-a-Number, 80
numerical error, sources of, 77
  input error, 77
  roundoff error, 77, 78

orthogonal (unitary) factorizations, 31
orthogonal (unitary) transformation, 34, 64
orthogonal factorization
  generalized, 35
overdetermined system, 15, 16, 93, 94
overflow, 78–80, 82, 90, 94, 104, 113

parallelism
  compiler directives, 55
  loop-based, 55
ParPre, xvi

pencil, 22
performance, 4, 54
    block size, 148
    crossover point, 148
    LWORK, 154
    recommendations, 154
    sensitivity, 154
permutation, 50
portability, 54, 56, 78
prebuilt libraries
    lapack, 5

QL factorization, 34
    implicit, 40
QR decomposition
    with pivoting, 53
QR factorization, 31
    blocked form, 63
    generalized (GQR), 17, 34, 35
    implicit, 40
    with pivoting, 32
query, 137
quotient singular value decomposition, 23, 51
QZ method, 49

rank
    numerical determination of, 33, 53, 92
reduction
    bidiagonal, 45
    tridiagonal, 39
    upper Hessenberg, 42
regression, generalized linear, 17
regularization, 94
relative error, 78, 80, 81, 83, 86, 90
relative gap, 105, 114
reliability, see test suites, 7, 150
reporting bugs, see bug reports, 150
roundoff error, 77, 78
RQ factorization, 34
    generalized (GRQ), 17, 37

s and dif, 125
s and sep, 110
ScaLAPACK, 8
scaling, 27, 109, 124
Schur decomposition
    generalized, 48
Schur factorization, 18, 42
    generalized, 22
Schur form, 42, 43
    generalized, 49, 51
Schur vectors, 19, 42
    generalized, 22, 49
SEP, 17, 39
separation of matrices, 110
separation of two matrix pairs, 125
shared memory, 54
similarity transformation, 43
singular value, 45
    error bound, 111, 113, 114, 130, 132
    generalized, 23
singular value decomposition
    generalized, 23, 25, 51
      special cases, 24
singular value decomposition (SVD), 19, 44
singular vector
    error bound, 111, 113, 114
singular vectors
    left, 19, 45
    right, 19, 45
source code, 5
spectral factorization, 18
spectral projector, 110
split Cholesky factorization, 47
stability, 31, 77
    backward, 85–87, 107, 112, 114, 118, 121, 131
    forward, 87
storage scheme, 139
    band, 141
    band LU, 141
    bidiagonal, 142
    conventional, 139
    diagonal of Hermitian matrix, 143
    Hermitian, 140
    orthogonal or unitary matrices, 143
    packed, 140
    symmetric, 140
    symmetric tridiagonal, 142
    triangular, 140
    unsymmetric tridiagonal, 142

Strassen's method, 132, 133
subspaces, 84
    angle between, 82–84, 94, 103, 106, 111, 115, 118, 119, 130
    deflating, 22, 49, 51
    invariant, 22, 43, 158
support, 7
SVD, 19
Sylvester equation, 43, 86, 110, 111
    generalized, 51, 126, 127
symmetric eigenproblem, 39
symmetric indefinite factorization, 26
    blocked form, 60

test suites, 7, 150
tridiagonal form, 39, 64, 105, 144
troubleshooting, 150
tuning
    block multishift QR: NS, MAXB, 147
    block size: NB, NBMIN, and NX, 147
    divide and conquer, 147
    SVD: NXSVD, 147

underdetermined system, 15, 16, 32, 94
underflow, 78–80
upper Hessenberg form, 42

vector registers, 55
vectorization, 55

workspace query, 137
workstation, super-scalar, 54
wrong results, 153

Index by Routine Name

BAKVEC (EISPACK), 190
BALANC (EISPACK), 190
BALBAK (EISPACK), 190
BANDR (EISPACK), 186, 190
BANDV (EISPACK), 186, 190
BISECT (EISPACK), 190
BQR (EISPACK), 186, 190

CBABK2 (EISPACK), 190
CBAL (EISPACK), 190
CBDSDC, 147
CBDSQR, 45, 46, 66, 113, 157, 209
CG (EISPACK), 190
CGBBRD, 45, 46, 157, 210
CGBCON, 27, 29, 157, 211
CGBEQU, 27, 29, 157, 174, 212
CGBRFS, 27, 29, 157, 212
CGBSV, 15, 157, 213
CGBSVX, 15, 85, 157, 214
CGBTRF, 27, 29, 147, 157, 216
CGBTRS, 27, 29, 157, 217
CGEBAK, 43, 44, 157, 190, 218
CGEBAL, 42–44, 109, 157, 190, 218
CGEBRD, 45, 46, 64, 66, 147, 157, 160, 161, 219
CGECON, 27, 29, 157, 220
CGEEQU, 27, 29, 157, 174, 220
CGEES, 19, 20, 154, 157, 221
CGEESX, 19, 20, 83, 107, 154, 158, 222
CGEEV, 19, 20, 154, 158, 190, 224
CGEEVX, 19, 20, 106, 107, 154, 158, 225
CGEHRD, 42–44, 64, 147, 158, 160, 161, 190, 227
CGELQF, 32, 35, 46, 147, 158, 160, 161, 228
CGELS, 16, 91, 93, 158, 228
CGELSD, 16, 66, 91, 93, 147, 158, 229
CGELSS, 16, 91, 93, 147, 158, 231
CGELSX, 16, 91, 93
CGELSY, 16, 91, 93, 158, 232
CGEQLF, 34, 35, 147, 158, 160, 161, 233
CGEQP3, 33, 35, 158, 233
CGEQPF, 33
CGEQRF, 31, 33, 35, 45, 147, 158, 160, 161, 234
CGERFS, 27, 29, 158, 234
CGERQF, 34, 35, 147, 158, 160, 161, 235
CGESDD, 20, 46, 66, 94, 111, 147, 158, 236
CGESV, 15, 158, 237
CGESVD, 20, 46, 94, 111, 113, 147, 159, 238
CGESVX, 15, 85, 159, 239
CGETRF, 27, 29, 147, 157, 159, 241
CGETRI, 27, 29, 147, 159, 242
CGETRS, 27, 29, 159, 242
CGGBAK, 50, 52, 159, 243
CGGBAL, 50, 52, 124, 159, 244
CGGES, 23, 25, 159, 245
CGGESX, 23, 25, 121, 159, 247
CGGEV, 23, 25, 159, 249
CGGEVX, 23, 25, 119, 121, 159, 250
CGGGLM, 17, 159, 253
CGGHRD, 49, 52, 159, 254
CGGLSE, 17, 159, 255
CGGQRF, 36, 159, 255
CGGRQF, 37, 159, 256
CGGSVD, 25, 130, 159, 258
CGGSVP, 52, 53, 159, 167, 260
CGTCON, 27, 29, 160, 261
CGTRFS, 27, 29, 160, 262
CGTSV, 15, 160, 263
CGTSVX, 15, 160, 263
CGTTRF, 27, 29, 160, 265
CGTTRS, 27, 29, 160, 265
CH (EISPACK), 190
CHBEV, 20, 103, 163, 307

CHBEVD, 20, 80, 103, 163, 308
CHBEVX, 20, 103, 163, 309
CHBGST, 47, 48, 163, 310
CHBGV, 25, 114, 163, 311
CHBGVD, 25, 80, 114, 163, 312
CHBGVX, 25, 114, 163, 313
CHBTRD, 39, 41, 163, 315
CHECON, 27, 30, 165, 189, 339
CHEEV, 20, 103, 165, 190, 340
CHEEVD, 20, 80, 103, 165, 341
CHEEVR, 20, 80, 103, 147, 166, 342
CHEEVX, 20, 103, 166, 344
CHEGST, 48, 147, 166, 345
CHEGV, 25, 86, 114, 166, 346
CHEGVD, 25, 80, 114, 166, 347
CHEGVX, 25, 114, 166, 348
CHERFS, 27, 30, 166, 350
CHESV, 15, 166, 351
CHESVX, 15, 166, 352
CHETRD, 39, 41, 147, 160, 161, 166, 191, 354
CHETRF, 27, 30, 147, 165, 166, 189, 354
CHETRI, 27, 30, 166, 189, 355
CHETRS, 27, 30, 166, 189, 356
CHGEQZ, 49, 52, 160, 266
CHICO (LINPACK), 189
CHIDI (LINPACK), 189
CHIFA (LINPACK), 189
CHISL (LINPACK), 189
CHPCO (LINPACK), 189
CHPCON, 27, 30, 164, 189, 316
CHPDI (LINPACK), 189
CHPEV, 20, 103, 164, 316
CHPEVD, 20, 80, 103, 164, 317
CHPEVX, 20, 103, 164, 318
CHPFA (LINPACK), 189
CHPGST, 48, 164, 319
CHPGV, 25, 114, 164, 320
CHPGVD, 25, 80, 114, 164, 321
CHPGVX, 25, 114, 164, 322
CHPRFS, 27, 30, 164, 324
CHPSL (LINPACK), 189
CHPSV, 15, 164, 325
CHPSVX, 15, 164, 326
CHPTRD, 39, 41, 160, 164, 191, 327
CHPTRF, 27, 30, 164–166, 189, 328
CHPTRI, 27, 30, 165, 189, 328
CHPTRS, 27, 30, 165, 189, 329
CHSEIN, 42, 44, 160, 190, 268
CHSEQR, 42–44, 147, 149, 154, 160, 190, 270
CINVIT (EISPACK), 190
COMBAK (EISPACK), 190
COMHES (EISPACK), 190
COMLR (EISPACK), 190
COMLR2 (EISPACK), 190
COMQR (EISPACK), 67, 190
COMQR2 (EISPACK), 190
CORTB (EISPACK), 190
CORTH (EISPACK), 190
CPBCON, 27, 29, 161, 284
CPBEQU, 27, 29, 161, 174, 285
CPBRFS, 27, 29, 161, 285
CPBSTF, 47, 48, 161, 286
CPBSV, 15, 161, 287
CPBSVX, 15, 161, 288
CPBTRF, 27, 29, 147, 161, 289
CPBTRS, 27, 29, 161, 290
CPOCON, 27, 29, 161, 291
CPOEQU, 27, 29, 162, 174, 291
CPORFS, 27, 29, 162, 292
CPOSV, 15, 162, 292
CPOSVX, 15, 162, 293
CPOTRF, 27, 29, 147, 161, 162, 166, 295
CPOTRI, 27, 29, 147, 162, 295
CPOTRS, 27, 29, 162, 296
CPPCON, 27, 29, 162, 296
CPPEQU, 27, 29, 162, 174, 297
CPPRFS, 27, 29, 162, 297
CPPSV, 15, 162, 298
CPPSVX, 15, 162, 299
CPPTRF, 27, 29, 162, 164, 301
CPPTRI, 27, 29, 162, 301
CPPTRS, 27, 29, 162, 301
CPTCON, 27, 29, 105, 163, 302
CPTEQR, 40, 41, 105, 163, 302
CPTRFS, 27, 29, 163, 303
CPTSV, 15, 163, 304
CPTSVX, 15, 163, 305
CPTTRF, 27, 29, 163, 306
CPTTRS, 27, 29, 163, 306
CSPCON, 27, 30, 164, 316

CSPRFS, 27, 30, 164, 324
CSPSV, 15, 164, 325
CSPSVX, 15, 164, 326
CSPTRF, 27, 30, 164–166, 328
CSPTRI, 27, 30, 165, 328
CSPTRS, 27, 30, 165, 329
CSTEDC, 40, 41, 65, 66, 80, 147, 165, 331
CSTEGR, 40, 41, 64, 66, 68, 80, 105, 147, 165, 332
CSTEIN, 39–41, 66, 104, 165, 334
CSTEQR, 39–41, 65, 165, 335
CSYCON, 27, 30, 165, 339
CSYEVR, 80
CSYRFS, 27, 30, 166, 350
CSYSV, 15, 166, 351
CSYSVX, 15, 166, 352
CSYTRD, 64
CSYTRF, 27, 30, 147, 165, 166, 354
CSYTRI, 27, 30, 166, 355
CSYTRS, 27, 30, 166, 356
CTBCON, 27, 30, 166, 356
CTBRFS, 27, 30, 167, 357
CTBTRS, 27, 30, 167, 358
CTGEVC, 50, 52, 167, 359
CTGEXC, 51, 52, 125, 167, 360
CTGSEN, 51, 52, 121, 167, 362
CTGSJA, 52, 53, 119, 167, 364
CTGSNA, 51, 52, 121, 167, 366
CTGSYL, 51, 52, 126, 167, 368
CTPCON, 27, 30, 167, 369
CTPRFS, 27, 30, 167, 370
CTPTRI, 27, 30, 167, 371
CTPTRS, 27, 30, 167, 371
CTRCON, 27, 30, 94, 167, 372
CTREVC, 42, 44, 167, 190, 373
CTREXC, 43, 44, 110, 167, 374
CTRRFS, 27, 30, 167, 375
CTRSEN, 44, 107, 168, 376
CTRSNA, 43, 44, 107, 168, 377
CTRSYL, 43, 44, 110, 168, 379
CTRTRI, 27, 30, 147, 168, 380
CTRTRS, 27, 30, 32, 168, 380
CTZRQF, 34
CTZRZF, 34, 35, 161, 168, 381
CUNGBR, 45, 46, 160, 272
CUNGHR, 42, 44, 160, 190, 273
CUNGLQ, 32, 35, 147, 160, 273
CUNGQL, 35, 147, 160, 274
CUNGQR, 31, 33, 35, 147, 160, 275
CUNGRQ, 35, 147, 160, 275
CUNGTR, 41, 160, 276
CUNMBR, 45, 46, 161, 276
CUNMHR, 42, 44, 161, 190, 278
CUNMLQ, 32, 35, 147, 161, 279
CUNMQL, 35, 147, 161, 279
CUNMQR, 31–33, 35, 36, 38, 147, 161, 280
CUNMRQ, 35, 37, 38, 147, 161, 281
CUNMRZ, 34, 35, 161, 282
CUNMTR, 39, 41, 161, 191, 283
CUPGTR, 39, 41, 160, 271
CUPMTR, 41, 160, 191, 271

DBDSDC, 45, 46, 80, 114, 147, 208
DBDSQR, 45, 46, 66, 113, 209
DCHDC (LINPACK), 187
DCHDD (LINPACK), 187
DCHEX (LINPACK), 187
DCHUD (LINPACK), 187
DDISNA, 104, 113, 209
DGBBRD, 45, 46, 210
DGBCO (LINPACK), 187
DGBCON, 27, 29, 187, 211
DGBDI (LINPACK), 187
DGBEQU, 27, 29, 212
DGBFA (LINPACK), 187
DGBRFS, 27, 29, 212
DGBSL (LINPACK), 187
DGBSV, 15, 213
DGBSVX, 15, 85, 214
DGBTRF, 27, 29, 187, 216
DGBTRS, 27, 29, 187, 217
DGEBAK, 43, 44, 218
DGEBAL, 42–44, 109, 218
DGEBRD, 45, 46, 64–66, 219
DGECO (LINPACK), 187
DGECON, 27, 29, 187, 220
DGEDI (LINPACK), 187
DGEEQU, 27, 29, 220
DGEES, 19, 20, 154, 221
DGEESX, 19, 20, 83, 107, 154, 222

DGEEV, 19, 20, 67, 154, 224
DGEEVX, 19, 20, 106, 107, 154, 225
DGEFA (LINPACK), 187
DGEHRD, 42–44, 64, 65, 227
DGELQF, 32, 35, 46, 228
DGELS, 16, 69, 91, 93, 228
DGELSD, 16, 66, 69, 80, 91, 93, 147, 229
DGELSS, 16, 69, 91, 93, 147, 231
DGELSX, 16, 69, 91, 93
DGELSY, 16, 69, 91, 93, 232
DGEQLF, 34, 35, 233
DGEQP3, 33, 35, 233
DGEQPF, 33, 188
DGEQRF, 31, 33, 35, 45, 63, 188, 234
DGERFS, 27, 29, 234
DGERQF, 34, 35, 235
DGESDD, 20, 46, 66, 67, 80, 94, 111, 147, 236
DGESL (LINPACK), 187
DGESV, 15, 67, 237
DGESVD, 20, 46, 66, 67, 94, 111, 113, 147, 188, 238
DGESVX, 15, 85, 239
DGETRF, 27, 29, 60, 187, 241
DGETRI, 27, 29, 187, 242
DGETRS, 27, 29, 187, 242
DGGBAK, 50, 52, 243
DGGBAL, 50, 52, 124, 244
DGGES, 23, 25, 245
DGGESX, 23, 25, 121, 247
DGGEV, 23, 25, 249
DGGEVX, 23, 25, 119, 121, 250
DGGGLM, 17, 253
DGGHRD, 49, 52, 254
DGGLSE, 17, 255
DGGQRF, 36, 255
DGGRQF, 37, 256
DGGSVD, 25, 130, 258
DGGSVP, 52, 53, 260
DGTCON, 27, 29, 261
DGTRFS, 27, 29, 262
DGTSL (LINPACK), 187
DGTSV, 15, 187, 263
DGTSVX, 15, 263
DGTTRF, 27, 29, 265
DGTTRS, 27, 29, 265
DHEEVR, 80
DHGEQZ, 49, 52, 266
DHSEIN, 42, 44, 268
DHSEQR, 42–44, 147, 149, 154, 270
DLALSD, 80
DLAMCH, 78, 146, 173
DLASV2, 79
DOPGTR, 39, 41, 271
DOPMTR, 39, 41, 271
DORGBR, 45, 46, 272
DORGHR, 42, 44, 273
DORGLQ, 32, 35, 273
DORGQL, 35, 274
DORGQR, 31, 33, 35, 275
DORGRQ, 35, 275
DORGTR, 39, 41, 276
DORMBR, 45, 46, 276
DORMHR, 42, 44, 278
DORMLQ, 32, 35, 279
DORMQL, 35, 279
DORMQR, 31–33, 35, 36, 38, 188, 280
DORMRQ, 35, 37, 38, 281
DORMRZ, 34, 35, 282
DORMTR, 39, 41, 283
DPBCO (LINPACK), 187
DPBCON, 27, 29, 187, 284
DPBDI (LINPACK), 187
DPBEQU, 27, 29, 285
DPBFA (LINPACK), 187
DPBRFS, 27, 29, 285
DPBSL (LINPACK), 187
DPBSTF, 47, 48, 286
DPBSV, 15, 287
DPBSVX, 15, 288
DPBTRF, 27, 29, 187, 289
DPBTRS, 27, 29, 187, 290
DPOCO (LINPACK), 187
DPOCON, 27, 29, 187, 291
DPODI (LINPACK), 187
DPOEQU, 27, 29, 291
DPOFA (LINPACK), 187
DPORFS, 27, 29, 292
DPOSL (LINPACK), 187
DPOSV, 15, 292
DPOSVX, 15, 293

DPOTRF, 27, 29, 60, 187, 295
DPOTRI, 27, 29, 187, 295
DPOTRS, 27, 29, 187, 296
DPPCO (LINPACK), 187, 188
DPPCON, 27, 29, 187, 296
DPPDI (LINPACK), 188
DPPEQU, 27, 29, 297
DPPFA (LINPACK), 188
DPPRFS, 27, 29, 297
DPPSL (LINPACK), 188
DPPSV, 15, 298
DPPSVX, 15, 299
DPPTRF, 27, 29, 187, 188, 301
DPPTRI, 27, 29, 188, 301
DPPTRS, 27, 29, 188, 301
DPTCON, 27, 29, 105, 302
DPTEQR, 40, 41, 105, 302
DPTRFS, 27, 29, 303
DPTSL (LINPACK), 188
DPTSV, 15, 188, 304
DPTSVX, 15, 305
DPTTRF, 27, 29, 306
DPTTRS, 27, 29, 306
DQRDC (LINPACK), 188
DQRSL (LINPACK), 188
DSBEV, 20, 103, 307
DSBEVD, 20, 80, 103, 308
DSBEVX, 20, 103, 309
DSBGST, 47, 48, 310
DSBGV, 25, 114, 311
DSBGVD, 25, 80, 114, 312
DSBGVX, 25, 114, 313
DSBTRD, 39, 41, 315
DSICO (LINPACK), 188
DSIDI (LINPACK), 188
DSIFA (LINPACK), 188
DSISL (LINPACK), 188
DSPCO (LINPACK), 188
DSPCON, 27, 30, 188, 316
DSPDI (LINPACK), 188
DSPEV, 20, 103, 316
DSPEVD, 20, 80, 103, 317
DSPEVX, 20, 103, 318
DSPFA (LINPACK), 188
DSPGST, 48, 319
DSPGV, 25, 114, 320
DSPGVD, 25, 80, 114, 321
DSPGVX, 25, 114, 322
DSPRFS, 27, 30, 324
DSPSL (LINPACK), 188
DSPSV, 15, 325
DSPSVX, 15, 326
DSPTRD, 39, 41, 327
DSPTRF, 27, 30, 188, 328
DSPTRI, 27, 30, 188, 328
DSPTRS, 27, 30, 188, 329
DSTEBZ, 40, 41, 105, 330
DSTEDC, 40, 41, 65, 66, 69, 80, 147, 331
DSTEGR, 40, 41, 64, 66, 68, 69, 80, 105, 147, 332
DSTEIN, 39–41, 66, 104, 334
DSTEQR, 39–41, 65, 69, 335
DSTERF, 40, 41, 64, 335
DSTEV, 20, 69, 103, 336
DSTEVD, 20, 69, 80, 103, 336
DSTEVR, 20, 69, 80, 103, 147, 337
DSTEVX, 20, 103, 105, 338
DSVDC (LINPACK), 188
DSYCON, 27, 30, 188, 339
DSYEV, 20, 69, 103, 340
DSYEVD, 20, 69, 80, 103, 341
DSYEVR, 20, 69, 80, 103, 147, 342
DSYEVX, 20, 69, 103, 344
DSYGST, 48, 345
DSYGV, 25, 86, 114, 346
DSYGVD, 25, 80, 114, 347
DSYGVX, 25, 114, 348
DSYRFS, 27, 30, 350
DSYSV, 15, 351
DSYSVX, 15, 352
DSYTRD, 39, 41, 64, 65, 354
DSYTRF, 27, 30, 60, 61, 188, 354
DSYTRI, 27, 30, 188, 355
DSYTRS, 27, 30, 188, 356
DTBCON, 27, 30, 356
DTBRFS, 27, 30, 357
DTBTRS, 27, 30, 358
DTGEVC, 50, 52, 359
DTGEXC, 51, 52, 125, 360
DTGSEN, 51, 52, 121, 362

DTGSJA, 52, 53, 119, 364
DTGSNA, 51, 52, 121, 366
DTGSYL, 51, 52, 126, 368
DTPCON, 27, 30, 369
DTPRFS, 27, 30, 370
DTPTRI, 27, 30, 371
DTPTRS, 27, 30, 371
DTRCO (LINPACK), 188
DTRCON, 27, 30, 94, 188, 372
DTRDI (LINPACK), 188
DTREVC, 42, 44, 373
DTREXC, 43, 44, 110, 374
DTRRFS, 27, 30, 375
DTRSEN, 44, 107, 376
DTRSL (LINPACK), 188
DTRSNA, 43, 44, 107, 377
DTRSYL, 43, 44, 110, 379
DTRTRI, 27, 30, 188, 380
DTRTRS, 27, 30, 32, 188, 380
DTZRQF, 34
DTZRZF, 34, 35, 381

ELMBAK (EISPACK), 190
ELMHES (EISPACK), 190
ELTRAN (EISPACK), 190

FIGI (EISPACK), 190, 191
FIGI2 (EISPACK), 191

HQR (EISPACK), 67, 191
HQR2 (EISPACK), 191
HTRIB3 (EISPACK), 191
HTRIBK (EISPACK), 191
HTRID3 (EISPACK), 186, 191
HTRIDI (EISPACK), 186, 191

ILAENV, 138, 146–148, 154, 170
IMTQL1 (EISPACK), 186, 191
IMTQL2 (EISPACK), 186, 191
IMTQLV (EISPACK), 191
INVIT (EISPACK), 191

LSAME, 170
LSAMEN, 170

MINFIT (EISPACK), 191

ORTBAK (EISPACK), 191
ORTHES (EISPACK), 191
ORTRAN (EISPACK), 191

QZHES (EISPACK), 191
QZIT (EISPACK), 191
QZVAL (EISPACK), 191
QZVEC (EISPACK), 191

RATQR (EISPACK), 186, 191
REBAK (EISPACK), 191
REBAKB (EISPACK), 191
REDUC (EISPACK), 191, 192
REDUC2 (EISPACK), 191, 192
RG (EISPACK), 192
RGG (EISPACK), 192
RS (EISPACK), 192
RSB (EISPACK), 186, 192
RSG (EISPACK), 192
RSGAB (EISPACK), 192
RSGBA (EISPACK), 192
RSM (EISPACK), 192
RSP (EISPACK), 192
RST (EISPACK), 186, 192
RT (EISPACK), 192

SBDSDC, 45, 46, 80, 114, 147, 157, 208
SBDSQR, 45, 46, 66, 113, 157, 209
SCHDC (LINPACK), 187
SCHDD (LINPACK), 187
SCHEX (LINPACK), 187
SCHUD (LINPACK), 187
SDISNA, 104, 113, 209
SGBBRD, 45, 46, 157, 210
SGBCO (LINPACK), 187
SGBCON, 27, 29, 157, 187, 211
SGBDI (LINPACK), 187
SGBEQU, 27, 29, 157, 174, 212
SGBFA (LINPACK), 187
SGBRFS, 27, 29, 157, 212
SGBSL (LINPACK), 187
SGBSV, 15, 157, 190, 213
SGBSVX, 15, 85, 157, 214
SGBTRF, 27, 29, 147, 157, 187, 216
SGBTRS, 27, 29, 157, 187, 217
SGEBAK, 43, 44, 157, 190, 218

SGEBAL, 42–44, 109, 157, 190, 218
SGEBRD, 45, 46, 64, 66, 147, 148, 157, 160, 161, 219
SGECO (LINPACK), 187
SGECON, 27, 29, 157, 187, 220
SGEDI (LINPACK), 187
SGEEQU, 27, 29, 157, 174, 220
SGEES, 19, 20, 154, 157, 221
SGEESX, 19, 20, 83, 107, 154, 158, 222
SGEEV, 19, 20, 154, 158, 192, 224
SGEEVX, 19, 20, 106, 107, 154, 158, 225
SGEFA (LINPACK), 187
SGEGV, 192
SGEHRD, 42–44, 64, 147, 158, 160, 161, 190, 191, 227
SGELQF, 32, 35, 46, 147, 158, 160, 161, 177, 178, 228
SGELS, 16, 91, 93, 158, 228
SGELSD, 16, 66, 80, 91, 93, 147, 158, 229
SGELSS, 16, 91, 93, 147, 158, 191, 231
SGELSX, 16, 91–93
SGELSY, 16, 91–93, 158, 232
SGEQLF, 34, 35, 147, 158, 160, 161, 177, 178, 233
SGEQP3, 33, 35, 158, 233
SGEQPF, 33, 188
SGEQRF, 31, 33, 35, 45, 147, 148, 158, 160, 161, 177, 178, 188, 234
SGERFS, 27, 29, 158, 234
SGERQF, 34, 35, 147, 158, 160, 161, 178, 235
SGESDD, 20, 46, 66, 80, 94, 111, 147, 158, 236
SGESL (LINPACK), 187
SGESV, 15, 88, 152, 158, 237
SGESVD, 20, 46, 94, 111, 113, 147, 159, 188, 192, 238
SGESVX, 15, 85, 89, 153, 159, 239
SGETRF, 27, 29, 147, 157, 159, 187, 241
SGETRI, 27, 29, 147, 159, 187, 242
SGETRS, 27, 29, 159, 187, 242
SGGBAK, 50, 52, 159, 243
SGGBAL, 50, 52, 124, 159, 244
SGGES, 23, 25, 159, 245
SGGESX, 23, 25, 121, 159, 247
SGGEV, 23, 25, 159, 249
SGGEVX, 23, 25, 119, 121, 159, 250
SGGGLM, 17, 159, 253
SGGHRD, 49, 52, 159, 191, 254
SGGLSE, 17, 159, 255
SGGQRF, 36, 159, 255
SGGRQF, 37, 159, 256
SGGSVD, 25, 130, 159, 258
SGGSVP, 52, 53, 159, 167, 260
SGTCON, 27, 29, 160, 261
SGTRFS, 27, 29, 160, 262
SGTSL (LINPACK), 187
SGTSV, 15, 160, 187, 263
SGTSVX, 15, 160, 263
SGTTRF, 27, 29, 160, 265
SGTTRS, 27, 29, 160, 265
SHEEVR, 80
SHGEQZ, 49, 52, 160, 191, 266
SHSEIN, 42, 44, 160, 191, 268
SHSEQR, 42–44, 147, 149, 154, 160, 191, 270
SLALSD, 80
SLAMCH, 78, 146, 155, 173
SLASV2, 79
SOPGTR, 39, 41, 160, 271
SOPMTR, 39, 41, 160, 193, 271
SORGBR, 45, 46, 160, 272
SORGHR, 42, 44, 160, 190, 191, 273
SORGLQ, 32, 35, 147, 160, 273
SORGQL, 35, 147, 160, 274
SORGQR, 31, 33, 35, 147, 160, 275
SORGRQ, 35, 147, 160, 275
SORGTR, 39, 41, 160, 193, 276
SORMBR, 45, 46, 161, 276
SORMHR, 42, 44, 161, 190, 191, 278
SORMLQ, 32, 35, 147, 161, 279
SORMQL, 35, 147, 161, 279
SORMQR, 31–33, 35, 36, 38, 147, 161, 188, 280
SORMRQ, 35, 37, 38, 147, 161, 281
SORMRZ, 34, 35, 161, 282
SORMTR, 39, 41, 161, 193, 283
SPBCO (LINPACK), 187
SPBCON, 27, 29, 161, 187, 284
SPBDI (LINPACK), 187
SPBEQU, 27, 29, 161, 174, 285
SPBFA (LINPACK), 187

SPBRFS, 27, 29, 161, 285
SPBSL (LINPACK), 187
SPBSTF, 47, 48, 161, 286
SPBSV, 15, 161, 287
SPBSVX, 15, 161, 288
SPBTRF, 27, 29, 147, 161, 187, 289
SPBTRS, 27, 29, 161, 187, 290
SPOCO (LINPACK), 187
SPOCON, 27, 29, 161, 187, 291
SPODI (LINPACK), 187
SPOEQU, 27, 29, 162, 174, 291
SPOFA (LINPACK), 56, 57, 187
SPORFS, 27, 29, 162, 292
SPOSL (LINPACK), 187
SPOSV, 15, 162, 292
SPOSVX, 15, 162, 293
SPOTRF, 27, 29, 59, 147, 161, 162, 166, 178, 187, 295
SPOTRI, 27, 29, 147, 162, 187, 295
SPOTRS, 27, 29, 162, 187, 296
SPPCO (LINPACK), 187, 188
SPPCON, 27, 29, 162, 187, 296
SPPDI (LINPACK), 188
SPPEQU, 27, 29, 162, 174, 297
SPPFA (LINPACK), 188
SPPRFS, 27, 29, 162, 297
SPPSL (LINPACK), 188
SPPSV, 15, 162, 298
SPPSVX, 15, 162, 299
SPPTRF, 27, 29, 162, 164, 187, 188, 301
SPPTRI, 27, 29, 162, 188, 301
SPPTRS, 27, 29, 162, 188, 301
SPTCON, 27, 29, 105, 163, 302
SPTEQR, 40, 41, 105, 163, 302
SPTRFS, 27, 29, 163, 303
SPTSL (LINPACK), 188
SPTSV, 15, 163, 188, 304
SPTSVX, 15, 163, 305
SPTTRF, 27, 29, 163, 306
SPTTRS, 27, 29, 163, 306
SQRDC (LINPACK), 188
SQRSL (LINPACK), 188
SSBEV, 20, 103, 163, 192, 307
SSBEVD, 20, 80, 103, 163, 192, 308
SSBEVX, 20, 103, 163, 190, 309
SSBGST, 47, 48, 163, 310
SSBGV, 25, 114, 163, 311
SSBGVD, 25, 80, 114, 163, 312
SSBGVX, 25, 114, 163, 313
SSBTRD, 39, 41, 163, 190, 315
SSICO (LINPACK), 188
SSIDI (LINPACK), 188
SSIFA (LINPACK), 188
SSISL (LINPACK), 188
SSPCO (LINPACK), 188
SSPCON, 27, 30, 164, 188, 316
SSPDI (LINPACK), 188
SSPEV, 20, 103, 164, 192, 316
SSPEVD, 20, 80, 103, 164, 192, 317
SSPEVX, 20, 103, 164, 318
SSPFA (LINPACK), 188
SSPGST, 48, 164, 319
SSPGV, 25, 114, 164, 320
SSPGVD, 25, 80, 114, 164, 321
SSPGVX, 25, 114, 164, 322
SSPRFS, 27, 30, 164, 324
SSPSL (LINPACK), 188
SSPSV, 15, 164, 325
SSPSVX, 15, 164, 326
SSPTRD, 39, 41, 160, 164, 193, 327
SSPTRF, 27, 30, 164–166, 188, 328
SSPTRI, 27, 30, 165, 188, 328
SSPTRS, 27, 30, 165, 188, 329
SSTEBZ, 40, 41, 105, 147, 165, 171, 190, 191, 193, 330
SSTEDC, 40, 41, 65, 66, 80, 147, 165, 191, 192, 331
SSTEGR, 40, 41, 64, 66, 68, 80, 105, 147, 165, 332
SSTEIN, 39–41, 66, 104, 165, 192, 193, 334
SSTEQR, 39–41, 65, 165, 191, 192, 335
SSTERF, 40, 41, 64, 165, 191, 192, 335
SSTEV, 20, 103, 165, 192, 336
SSTEVD, 20, 80, 103, 165, 192, 336
SSTEVR, 20, 80, 103, 147, 165, 337
SSTEVX, 20, 103, 105, 165, 338
SSVDC (LINPACK), 188
SSYCON, 27, 30, 165, 188, 339
SSYEV, 20, 103, 165, 192, 340
SSYEVD, 20, 80, 103, 165, 192, 341

SSYEVR, 20, 80, 103, 147, 166, 342
SSYEVX, 20, 103, 166, 192, 344
SSYGST, 48, 147, 166, 192, 345
SSYGV, 25, 86, 114, 166, 192, 346
SSYGVD, 25, 80, 114, 166, 347
SSYGVX, 25, 114, 166, 348
SSYRFS, 27, 30, 166, 350
SSYSV, 15, 166, 351
SSYSVX, 15, 166, 352
SSYTRD, 39, 41, 64, 147, 160, 161, 166, 193, 354
SSYTRF, 27, 30, 60, 147, 165, 166, 188, 354
SSYTRI, 27, 30, 166, 188, 355
SSYTRS, 27, 30, 166, 188, 356
STBCON, 27, 30, 166, 356
STBRFS, 27, 30, 167, 357
STBTRS, 27, 30, 167, 358
STGEVC, 50, 52, 167, 191, 359
STGEXC, 51, 52, 125, 167, 360
STGSEN, 51, 52, 121, 167, 362
STGSJA, 52, 53, 119, 167, 364
STGSNA, 51, 52, 121, 167, 366
STGSYL, 51, 52, 126, 167, 368
STPCON, 27, 30, 167, 369
STPRFS, 27, 30, 167, 370
STPTRI, 27, 30, 167, 371
STPTRS, 27, 30, 167, 371
STRCO (LINPACK), 188
STRCON, 27, 30, 94, 167, 188, 372
STRDI (LINPACK), 188
STREVC, 42, 44, 167, 191, 373
STREXC, 43, 44, 110, 167, 374
STRRFS, 27, 30, 167, 375
STRSEN, 44, 107, 168, 376
STRSL (LINPACK), 188
STRSNA, 43, 44, 107, 168, 377
STRSYL, 43, 44, 110, 168, 379
STRTRI, 27, 30, 147, 168, 188, 380
STRTRS, 27, 30, 32, 168, 188, 380
STZRQF, 34
STZRZF, 34, 35, 161, 168, 178, 381
SVD (EISPACK), 192

TINVIT (EISPACK), 192
TQL1 (EISPACK), 186, 192
TQL2 (EISPACK), 186, 192
TQLRAT (EISPACK), 186, 192
TRBAK1 (EISPACK), 193
TRBAK3 (EISPACK), 193
TRED1 (EISPACK), 186, 193
TRED2 (EISPACK), 186, 193
TRED3 (EISPACK), 186, 193
TRIDIB (EISPACK), 193
TSTURM (EISPACK), 193

XERBLA, 138, 152, 178

ZBDSDC, 147
ZBDSQR, 45, 46, 66, 113, 209
ZGBBRD, 45, 46, 210
ZGBCON, 27, 29, 211
ZGBEQU, 27, 29, 212
ZGBRFS, 27, 29, 212
ZGBSV, 15, 213
ZGBSVX, 15, 85, 214
ZGBTRF, 27, 29, 216
ZGBTRS, 27, 29, 217
ZGEBAK, 43, 44, 218
ZGEBAL, 42–44, 109, 218
ZGEBRD, 45, 46, 64, 66, 219
ZGECON, 27, 29, 220
ZGEEQU, 27, 29, 220
ZGEES, 19, 20, 154, 221
ZGEESX, 19, 20, 83, 107, 154, 222
ZGEEV, 19, 20, 154, 224
ZGEEVX, 19, 20, 106, 107, 154, 225
ZGEHRD, 42–44, 64, 227
ZGELQF, 32, 35, 46, 228
ZGELS, 16, 91, 93, 228
ZGELSD, 16, 66, 91, 93, 147, 229
ZGELSS, 16, 91, 93, 147, 231
ZGELSX, 16, 91, 93
ZGELSY, 16, 91, 93, 232
ZGEQLF, 34, 35, 233
ZGEQP3, 33, 35, 233
ZGEQPF, 33
ZGEQRF, 31, 33, 35, 45, 234
ZGERFS, 27, 29, 234
ZGERQF, 34, 35, 235
ZGESDD, 20, 46, 66, 94, 111, 147, 236
ZGESV, 15, 237

ZGESVD, 20, 46, 94, 111, 113, 147, 238
ZGESVX, 15, 85, 239
ZGETRF, 27, 29, 241
ZGETRI, 27, 29, 242
ZGETRS, 27, 29, 242
ZGGBAK, 50, 52, 243
ZGGBAL, 50, 52, 124, 244
ZGGES, 23, 25, 245
ZGGESX, 23, 121, 247
ZGGEV, 23, 25, 249
ZGGEVX, 23, 25, 119, 121, 250
ZGGGLM, 17, 253
ZGGHRD, 49, 52, 254
ZGGLSE, 17, 255
ZGGQRF, 36, 255
ZGGRQF, 37, 256
ZGGSVD, 25, 130, 258
ZGGSVP, 52, 53, 260
ZGTCON, 27, 29, 261
ZGTRFS, 27, 29, 262
ZGTSV, 15, 263
ZGTSVX, 15, 263
ZGTTRF, 27, 29, 265
ZGTTRS, 27, 29, 265
ZHBEV, 20, 103, 307
ZHBEVD, 20, 80, 103, 308
ZHBEVX, 20, 103, 309
ZHBGST, 47, 48, 310
ZHBGV, 25, 114, 311
ZHBGVD, 25, 80, 114, 312
ZHBGVX, 25, 114, 313
ZHBTRD, 39, 41, 315
ZHECON, 27, 30, 189, 339
ZHEEV, 20, 103, 340
ZHEEVD, 20, 80, 103, 341
ZHEEVR, 20, 80, 103, 147, 342
ZHEEVX, 20, 103, 344
ZHEGST, 48, 345
ZHEGV, 25, 86, 114, 346
ZHEGVD, 25, 80, 114, 347
ZHEGVX, 25, 114, 348
ZHERFS, 27, 30, 350
ZHESV, 15, 351
ZHESVX, 15, 352
ZHETRD, 39, 41, 354
ZHETRF, 27, 30, 189, 354
ZHETRI, 27, 30, 189, 355
ZHETRS, 27, 30, 189, 356
ZHGEQZ, 49, 52, 266
ZHICO (LINPACK), 189
ZHIDI (LINPACK), 189
ZHIFA (LINPACK), 189
ZHISL (LINPACK), 189
ZHPCO (LINPACK), 189
ZHPCON, 27, 30, 189, 316
ZHPDI (LINPACK), 189
ZHPEV, 20, 103, 316
ZHPEVD, 20, 80, 103, 317
ZHPEVX, 20, 103, 318
ZHPFA (LINPACK), 189
ZHPGST, 48, 319
ZHPGV, 25, 114, 320
ZHPGVD, 25, 80, 114, 321
ZHPGVX, 25, 114, 322
ZHPRFS, 27, 30, 324
ZHPSL (LINPACK), 189
ZHPSV, 15, 325
ZHPSVX, 15, 326
ZHPTRD, 39, 41, 327
ZHPTRF, 27, 30, 189, 328
ZHPTRI, 27, 30, 189, 328
ZHPTRS, 27, 30, 189, 329
ZHSEIN, 42, 44, 268
ZHSEQR, 42–44, 147, 149, 154, 270
ZPBCON, 27, 29, 284
ZPBEQU, 27, 29, 285
ZPBRFS, 27, 29, 285
ZPBSTF, 47, 48, 286
ZPBSV, 15, 287
ZPBSVX, 15, 288
ZPBTRF, 27, 29, 289
ZPBTRS, 27, 29, 290
ZPOCON, 27, 29, 291
ZPOEQU, 27, 29, 291
ZPORFS, 27, 29, 292
ZPOSV, 15, 292
ZPOSVX, 15, 293
ZPOTRF, 27, 29, 295
ZPOTRI, 27, 29, 295
ZPOTRS, 27, 29, 296

ZPPCON, 27, 29, 296
ZPPEQU, 27, 29, 297
ZPPRFS, 27, 29, 297
ZPPSV, 15, 298
ZPPSVX, 15, 299
ZPPTRF, 27, 29, 301
ZPPTRI, 27, 29, 301
ZPPTRS, 27, 29, 301
ZPTCON, 27, 29, 105, 302
ZPTEQR, 40, 41, 105, 302
ZPTRFS, 27, 29, 303
ZPTSV, 15, 304
ZPTSVX, 15, 305
ZPTTRF, 27, 29, 306
ZPTTRS, 27, 29, 306
ZSPCON, 27, 30, 316
ZSPRFS, 27, 30, 324
ZSPSV, 15, 325
ZSPSVX, 15, 326
ZSPTRF, 27, 30, 328
ZSPTRI, 27, 30, 328
ZSPTRS, 27, 30, 329
ZSTEDC, 40, 41, 65, 66, 80, 147, 331
ZSTEGR, 40, 41, 64, 66, 68, 80, 105, 147, 332
ZSTEIN, 39–41, 66, 104, 334
ZSTEQR, 39–41, 65, 335
ZSYCON, 27, 30, 339
ZSYEVR, 80
ZSYRFS, 27, 30, 350
ZSYSV, 15, 351
ZSYSVX, 15, 352
ZSYTRD, 64
ZSYTRF, 27, 30, 354
ZSYTRI, 27, 30, 355
ZSYTRS, 27, 30, 356
ZTBCON, 27, 30, 356
ZTBRFS, 27, 30, 357
ZTBTRS, 27, 30, 358
ZTGEVC, 50, 52, 359
ZTGEXC, 51, 52, 125, 360
ZTGSEN, 51, 52, 121, 362
ZTGSJA, 52, 53, 119, 364
ZTGSNA, 51, 52, 121, 366
ZTGSYL, 51, 52, 126, 368
ZTPCON, 27, 30, 369
ZTPRFS, 27, 30, 370
ZTPTRI, 27, 30, 371
ZTPTRS, 27, 30, 371
ZTRCON, 27, 30, 94, 372
ZTREVC, 42, 44, 373
ZTREXC, 43, 44, 110, 374
ZTRRFS, 27, 30, 375
ZTRSEN, 44, 107, 376
ZTRSNA, 43, 44, 107, 377
ZTRSYL, 43, 44, 110, 379
ZTRTRI, 27, 30, 380
ZTRTRS, 27, 30, 32, 380
ZTZRQF, 34
ZTZRZF, 34, 35, 381
ZUNGBR, 45, 46, 272
ZUNGHR, 42, 44, 273
ZUNGLQ, 32, 35, 273
ZUNGQL, 35, 274
ZUNGQR, 31, 33, 35, 275
ZUNGRQ, 35, 275
ZUNGTR, 41, 276
ZUNMBR, 45, 46, 276
ZUNMHR, 42, 44, 278
ZUNMLQ, 32, 35, 279
ZUNMQL, 35, 279
ZUNMQR, 31–33, 35, 36, 38, 280
ZUNMRQ, 35, 37, 38, 281
ZUNMRZ, 34, 35, 282
ZUNMTR, 39, 41, 283
ZUPGTR, 39, 41, 271
ZUPMTR, 41, 271

LAPACK Quick Reference Guide to the Driver Routines

Release 3.0

Simple Drivers

Simple Driver Routines for Linear Equations

Matrix Type	Routine
General	SGESV(N, NRHS, A, LDA, IPIV, B, LDB, INFO)
	CGESV(N, NRHS, A, LDA, IPIV, B, LDB, INFO)
General Band	SGBSV(N, KL, KU, NRHS, AB, LDAB, IPIV, B, LDB, INFO)
	CGBSV(N, KL, KU, NRHS, AB, LDAB, IPIV, B, LDB, INFO)
General Tridiagonal	SGTSV(N, NRHS, DL, D, DU, B, LDB, INFO)
	CGTSV(N, NRHS, DL, D, DU, B, LDB, INFO)
Symmetric/Hermitian Positive Definite	SPOSV(UPLO, N, NRHS, A, LDA, B, LDB, INFO)
	CPOSV(UPLO, N, NRHS, A, LDA, B, LDB, INFO)
Symmetric/Hermitian Positive Definite (Packed Storage)	SPPSV(UPLO, N, NRHS, AP, B, LDB, INFO)
	CPPSV(UPLO, N, NRHS, AP, B, LDB, INFO)
Symmetric/Hermitian Positive Definite Band	SPBSV(UPLO, N, KD, NRHS, AB, LDAB, B, LDB, INFO)
	CPBSV(UPLO, N, KD, NRHS, AB, LDAB, B, LDB, INFO)
Symmetric/Hermitian Positive Definite Tridiagonal	SPTSV(N, NRHS, D, E, B, LDB, INFO)
	CPTSV(N, NRHS, D, E, B, LDB, INFO)
Symmetric/Hermitian Indefinite	SSYSV(UPLO, N, NRHS, A, LDA, IPIV, B, LDB, WORK, LWORK, INFO)
	CSYSV(UPLO, N, NRHS, A, LDA, IPIV, B, LDB, WORK, LWORK, INFO)
	CHESV(UPLO, N, NRHS, A, LDA, IPIV, B, LDB, WORK, LWORK, INFO)
Symmetric/Hermitian Indefinite (Packed Storage)	SSPSV(UPLO, N, NRHS, AP, IPIV, B, LDB, INFO)
	CSPSV(UPLO, N, NRHS, AP, IPIV, B, LDB, INFO)
	CHPSV(UPLO, N, NRHS, AP, IPIV, B, LDB, INFO)

Simple Driver Routines for Standard and Generalized Linear Least Squares Problems

Problem Type	Routine
Solve Using Orthogonal Factor, Assuming Full Rank	SGELS(TRANS, M, N, NRHS, A, LDA, B, LDB, WORK, LWORK, INFO)
	CGELS(TRANS, M, N, NRHS, A, LDA, B, LDB, WORK, LWORK, INFO)
Solve LSE Problem Using GRQ	SGGLSE(M, N, P, A, LDA, B, LDB, C, D, X, WORK, LWORK, INFO)
	CGGLSE(M, N, P, A, LDA, B, LDB, C, D, X, WORK, LWORK, INFO)
Solve GLM Problem Using GQR	SGGGLM(N, M, P, A, LDA, B, LDB, D, X, Y, WORK, LWORK, INFO)
	CGGGLM(N, M, P, A, LDA, B, LDB, D, X, Y, WORK, LWORK, INFO)

Simple and Divide and Conquer Driver Routines for Standard Eigenvalue and Singular Value Problems

Matrix/Problem Type	Routine
Symmetric/Hermitian	SSYEV(JOBZ, UPLO, N, A, LDA, W, WORK, LWORK, INFO)
Eigenvalues/vectors	CHEEV(JOBZ, UPLO, N, A, LDA, W, WORK, LWORK, RWORK, INFO)
Divide and Conquer	SSYEVD(JOBZ, UPLO, N, A, LDA, W, WORK, LWORK, IWORK, LIWORK, INFO)
	CHEEVD(JOBZ, UPLO, N, A, LDA, W, WORK, LWORK, RWORK, LRWORK, IWORK, LIWORK, INFO)
Symmetric/Hermitian	SSPEV(JOBZ, UPLO, N, AP, W, Z, LDZ, WORK, INFO)
(Packed Storage)	CHPEV(JOBZ, UPLO, N, AP, W, Z, LDZ, WORK, RWORK, INFO)
Eigenvalues/vectors	
Divide and Conquer	SSPEVD(JOBZ, UPLO, N, AP, W, Z, LDZ, WORK, LWORK, IWORK, LIWORK, INFO)
	CHPEVD(JOBZ, UPLO, N, AP, W, Z, LDZ, WORK, LWORK, RWORK, LRWORK, IWORK, LIWORK, INFO)
Symmetric/Hermitian Band	SSBEV(JOBZ, UPLO, N, KD, AB, LDAB, W, Z, LDZ, WORK, INFO)
Eigenvalues/vectors	CHBEV(JOBZ, UPLO, N, KD, AB, LDAB, W, Z, LDZ, WORK, RWORK, INFO)
Divide and Conquer	SSBEVD(JOBZ, UPLO, N, KD, AB, LDAB, W, Z, LDZ, WORK, LWORK, IWORK, LIWORK, INFO)
	CHBEVD(JOBZ, UPLO, N, KD, AB, LDAB, W, Z, LDZ, WORK, LWORK, RWORK, LRWORK, IWORK, LIWORK, INFO)
Symmetric Tridiagonal	SSTEV(JOBZ, N, D, E, Z, LDZ, WORK, INFO)
Eigenvalues/vectors	
Divide and Conquer	SSTEVD(JOBZ, N, D, E, Z, LDZ, WORK, LWORK, IWORK, LIWORK, INFO)
General	SGEES(JOBVS, SORT, SELECT, N, A, LDA, SDIM, WR, WI, VS, LDVS, WORK, LWORK, BWORK, INFO)
Schur Factorization	CGEES(JOBVS, SORT, SELECT, N, A, LDA, SDIM, W, VS, LDVS, WORK, LWORK, RWORK, BWORK, INFO)
General	SGEEV(JOBVL, JOBVR, N, A, LDA, WR, WI, VL, LDVL, VR, LDVR, WORK, LWORK, INFO)
Eigenvalues/vectors	CGEEV(JOBVL, JOBVR, N, A, LDA, W, VL, LDVL, VR, LDVR, WORK, LWORK, RWORK, INFO)
General	SGESVD(JOBU, JOBVT, M, N, A, LDA, S, U, LDU, VT, LDVT, WORK, LWORK, INFO)
Singular Values/Vectors	CGESVD(JOBU, JOBVT, M, N, A, LDA, S, U, LDU, VT, LDVT, WORK, LWORK, RWORK, INFO)
Divide and Conquer	SGESDD(JOBZ, M, N, A, LDA, S, U, LDU, VT, LDVT, WORK, LWORK, IWORK, INFO)
	CGESDD(JOBZ, M, N, A, LDA, S, U, LDU, VT, LDVT, WORK, LWORK, RWORK, IWORK, INFO)

Simple and Divide and Conquer Driver Routines for Generalized Eigenvalue and Singular Value Problems

Matrix/Problem Type	Routine
Symmetric-definite	SSYGV(ITYPE, JOBZ, UPLO, N, A, LDA, B, LDB, W, WORK, LWORK, INFO)
Eigenvalues/vectors	CHEGV(ITYPE, JOBZ, UPLO, N, A, LDA, B, LDB, W, WORK, LWORK, RWORK, INFO)
Divide and Conquer	SSYGVD(ITYPE, JOBZ, UPLO, N, A, LDA, B, LDB, W, WORK, LWORK, IWORK, LIWORK, INFO)
	CHEGVD(ITYPE, JOBZ, UPLO, N, A, LDA, B, LDB, W, WORK, LWORK, RWORK, LRWORK, IWORK, LIWORK, INFO)
Symmetric-definite	SSPGV(ITYPE, JOBZ, UPLO, N, AP, BP, W, Z, LDZ, WORK, INFO)
(Packed Storage)	CHPGV(ITYPE, JOBZ, UPLO, N, AP, BP, W, Z, LDZ, WORK, RWORK, INFO)
Eigenvalues/vectors	
Divide and Conquer	SSPGVD(ITYPE, JOBZ, UPLO, N, AP, BP, W, Z, LDZ, WORK, LWORK, IWORK, LIWORK, INFO)
	CHPGVD(ITYPE, JOBZ, UPLO, N, AP, BP, W, Z, LDZ, WORK, LWORK, RWORK, LRWORK, IWORK, LIWORK, INFO)
Symmetric-definite	SSBGV(JOBZ, UPLO, N, KA, KB, AB, LDAB, BB, LDBB, W, Z, LDZ, WORK, INFO)
(Band Storage)	CHBGV(JOBZ, UPLO, N, KA, KB, AB, LDAB, BB, LDBB, W, Z, LDZ, WORK, RWORK, INFO)
Eigenvalues/vectors	
Divide and Conquer	SSBGVD(JOBZ, UPLO, N, KA, KB, AB, LDAB, BB, LDBB, W, Z, LDZ, WORK, LWORK, IWORK, LIWORK, INFO)
	CHBGVD(JOBZ, UPLO, N, KA, KB, AB, LDAB, BB, LDBB, W, Z, LDZ, WORK, LWORK, RWORK, LRWORK, IWORK, LIWORK, INFO)
General	SGGES(JOBVSL, JOBVSR, SORT, SELCTG, N, A, LDA, B, LDB, SDIM, ALPHAR, ALPHAI, BETA, VSL, LDVSL, VSR, LDVSR, WORK, LWORK, BWORK, INFO)
Schur Factorization	CGGES(JOBVSL, JOBVSR, SORT, SELCTG, N, A, LDA, B, LDB, SDIM, ALPHA, BETA, VSL, LDVSL, VSR, LDVSR, WORK, LWORK, RWORK, BWORK, INFO)
General	SGGEV(JOBVL, JOBVR, N, A, LDA, B, LDB, ALPHAR, ALPHAI, BETA, VL, LDVL, VR, LDVR, WORK, LWORK, INFO)
Eigenvalues/vectors	CGGEV(JOBVL, JOBVR, N, A, LDA, B, LDB, ALPHA, BETA, VL, LDVL, VR, LDVR, WORK, LWORK, RWORK, INFO)
General	SGGSVD(JOBU, JOBV, JOBQ, M, N, P, K, L, A, LDA, B, LDB, ALPHA, BETA, U, LDU, V, LDV, Q, LDQ, WORK, IWORK, INFO)
Singular Values/Vectors	CGGSVD(JOBU, JOBV, JOBQ, M, N, P, K, L, A, LDA, B, LDB, ALPHA, BETA, U, LDU, V, LDV, Q, LDQ, WORK, RWORK, IWORK, INFO)

Expert Drivers

Expert Driver Routines for Linear Equations

Matrix Type	Routine
General	SGESVX(FACT, TRANS, N, NRHS, A, LDA, AF, LDAF, IPIV, EQUED, R, C, B, LDB, X, LDX, RCOND, FERR, BERR, WORK, IWORK, INFO)
	CGESVX(FACT, TRANS, N, NRHS, A, LDA, AF, LDAF, IPIV, EQUED, R, C, B, LDB, X, LDX, RCOND, FERR, BERR, WORK, RWORK, INFO)
General Band	SGBSVX(FACT, TRANS, N, KL, KU, NRHS, AB, LDAB, AFB, LDAFB, IPIV, EQUED, R, C, B, LDB, X, LDX, RCOND, FERR, BERR, WORK, IWORK, INFO)
	CGBSVX(FACT, TRANS, N, KL, KU, NRHS, AB, LDAB, AFB, LDAFB, IPIV, EQUED, R, C, B, LDB, X, LDX, RCOND, FERR, BERR, WORK, RWORK, INFO)
General Tridiagonal	SGTSVX(FACT, TRANS, N, NRHS, DL, D, DU, DLF, DF, DUF, DU2, IPIV, B, LDB, X, LDX, RCOND, FERR, BERR, WORK, IWORK, INFO)
	CGTSVX(FACT, TRANS, N, NRHS, DL, D, DU, DLF, DF, DUF, DU2, IPIV, B, LDB, X, LDX, RCOND, FERR, BERR, WORK, RWORK, INFO)
Symmetric/Hermitian Positive Definite	SPOSVX(FACT, UPLO, N, NRHS, A, LDA, AF, LDAF, EQUED, S, B, LDB, X, LDX, RCOND, FERR, BERR, WORK, IWORK, INFO)
	CPOSVX(FACT, UPLO, N, NRHS, A, LDA, AF, LDAF, EQUED, S, B, LDB, X, LDX, RCOND, FERR, BERR, WORK, RWORK, INFO)
Symmetric/Hermitian Positive Definite (Packed Storage)	SPPSVX(FACT, UPLO, N, NRHS, AP, AFP, EQUED, S, B, LDB, X, LDX, RCOND, FERR, BERR, WORK, IWORK, INFO)
	CPPSVX(FACT, UPLO, N, NRHS, AP, AFP, EQUED, S, B, LDB, X, LDX, RCOND, FERR, BERR, WORK, RWORK, INFO)
Symmetric/Hermitian Positive Definite Band	SPBSVX(FACT, UPLO, N, KD, NRHS, AB, LDAB, AFB, LDAFB, EQUED, S, B, LDB, X, LDX, RCOND, FERR, BERR, WORK, IWORK, INFO)
	CPBSVX(FACT, UPLO, N, KD, NRHS, AB, LDAB, AFB, LDAFB, EQUED, S, B, LDB, X, LDX, RCOND, FERR, BERR, WORK, RWORK, INFO)
Symmetric/Hermitian Positive Definite Tridiagonal	SPTSVX(FACT, N, NRHS, D, E, DF, EF, B, LDB, X, LDX, RCOND, FERR, BERR, WORK, INFO)
	CPTSVX(FACT, N, NRHS, D, E, DF, EF, B, LDB, X, LDX, RCOND, FERR, BERR, WORK, RWORK, INFO)
Symmetric/Hermitian Indefinite	SSYSVX(FACT, UPLO, N, NRHS, A, LDA, AF, LDAF, IPIV, B, LDB, X, LDX, RCOND, FERR, BERR, WORK, LWORK, IWORK, INFO)
	CSYSVX(FACT, UPLO, N, NRHS, A, LDA, AF, LDAF, IPIV, B, LDB, X, LDX, RCOND, FERR, BERR, WORK, LWORK, RWORK, INFO)
	CHESVX(FACT, UPLO, N, NRHS, A, LDA, AF, LDAF, IPIV, B, LDB, X, LDX, RCOND, FERR, BERR, WORK, LWORK, RWORK, INFO)
Symmetric/Hermitian Indefinite (Packed Storage)	SSPSVX(FACT, UPLO, N, NRHS, AP, AFP, IPIV, B, LDB, X, LDX, RCOND, FERR, BERR, WORK, IWORK, INFO)
	CSPSVX(FACT, UPLO, N, NRHS, AP, AFP, IPIV, B, LDB, X, LDX, RCOND, FERR, BERR, WORK, RWORK, INFO)
	CHPSVX(FACT, UPLO, N, NRHS, AP, AFP, IPIV, B, LDB, X, LDX, RCOND, FERR, BERR, WORK, RWORK, INFO)

Divide and Conquer and Expert Driver Routines for Linear Least Squares Problems

Problem Type	Routine
Solve Using Orthogonal Factor	SGELSY(M, N, NRHS, A, LDA, B, LDB, JPVT, RCOND, RANK, WORK, LWORK, INFO)
	CGELSY(M, N, NRHS, A, LDA, B, LDB, JPVT, RCOND, RANK, WORK, LWORK, RWORK, INFO)
Solve Using SVD, Allowing for Rank-Deficiency	SGELSS(M, N, NRHS, A, LDA, B, LDB, S, RCOND, RANK, WORK, LWORK, INFO)
	CGELSS(M, N, NRHS, A, LDA, B, LDB, S, RCOND, RANK, WORK, LWORK, RWORK, INFO)
Solve Using D&C SVD, Allowing for Rank-Deficiency	SGELSD(M, N, NRHS, A, LDA, B, LDB, S, RCOND, RANK, WORK, LWORK, IWORK, INFO)
	CGELSD(M, N, NRHS, A, LDA, B, LDB, S, RCOND, RANK, WORK, LWORK, RWORK, IWORK, INFO)

Expert and RRR Driver Routines for Standard and Generalized Symmetric Eigenvalue Problems

Matrix/Problem Type	Routine
Symmetric/Hermitian Eigenvalues/vectors	SSYEVX(JOBZ, RANGE, UPLO, N, A, LDA, VL, VU, IL, IU, ABSTOL, M, W, Z, LDZ, WORK, LWORK, IWORK, IFAIL, INFO)
	CHEEVX(JOBZ, RANGE, UPLO, N, A, LDA, VL, VU, IL, IU, ABSTOL, M, W, Z, LDZ, WORK, LWORK, RWORK, IWORK, IFAIL, INFO)
	SSYEVR(JOBZ, RANGE, UPLO, N, A, LDA, VL, VU, IL, IU, ABSTOL, M, W, Z, LDZ, WORK, LWORK, IWORK, IFAIL, INFO)
	CHEEVR(JOBZ, RANGE, UPLO, N, A, LDA, VL, VU, IL, IU, ABSTOL, M, W, Z, LDZ, WORK, LWORK, RWORK, IWORK, IFAIL, INFO)
	SSYGVX(ITYPE, JOBZ, RANGE, UPLO, N, A, LDA, B, LDB, VL, VU, IL, IU, ABSTOL, M, W, Z, LDZ, WORK, LWORK, IWORK, IFAIL, INFO)
	CHEGVX(ITYPE, JOBZ, RANGE, UPLO, N, A, LDA, B, LDB, VL, VU, IL, IU, ABSTOL, M, W, Z, LDZ, WORK, LWORK, RWORK, IWORK, IFAIL, INFO)
Symmetric/Hermitian (Packed Storage) Eigenvalues/vectors	SSPEVX(JOBZ, RANGE, UPLO, N, AP, VL, VU, IL, IU, ABSTOL, M, W, Z, LDZ, WORK, IWORK, IFAIL, INFO)
	CHPEVX(JOBZ, RANGE, UPLO, N, AP, VL, VU, IL, IU, ABSTOL, M, W, Z, LDZ, WORK, RWORK, IWORK, IFAIL, INFO)
	SSPGVX(ITYPE, JOBZ, RANGE, UPLO, N, AP, BP, VL, VU, IL, IU, ABSTOL, M, W, Z, LDZ, WORK, IWORK, IFAIL, INFO)
	CHPGVX(ITYPE, JOBZ, RANGE, UPLO, N, AP, BP, VL, VU, IL, IU, ABSTOL, M, W, Z, LDZ, WORK, RWORK, IWORK, IFAIL, INFO)
Symmetric/Hermitian Band Eigenvalues/vectors	SSBEVX(JOBZ, RANGE, UPLO, N, KD, AB, LDAB, Q, LDQ, VL, VU, IL, IU, ABSTOL, M, W, Z, LDZ, WORK, IWORK, IFAIL, INFO)
	CHBEVX(JOBZ, RANGE, UPLO, N, KD, AB, LDAB, Q, LDQ, VL, VU, IL, IU, ABSTOL, M, W, Z, LDZ, WORK, RWORK, IWORK, IFAIL, INFO)
	SSBGVX(JOBZ, RANGE, UPLO, N, KA, KB, AB, LDAB, BB, LDBB, Q, LDQ, VL, VU, IL, IU, ABSTOL, M, W, Z, LDZ, WORK, IWORK, IFAIL, INFO)
	CHBGVX(JOBZ, RANGE, UPLO, N, KA, KB, AB, LDAB, BB, LDBB, Q, LDQ, VL, VU, IL, IU, ABSTOL, M, W, Z, LDZ, WORK, RWORK, IWORK, IFAIL, INFO)
Symmetric Tridiagonal Eigenvalues/vectors	SSTEVX(JOBZ, RANGE, N, D, E, VL, VU, IL, IU, ABSTOL, M, W, Z, LDZ, WORK, IWORK, IFAIL, INFO)
	SSTEVR(JOBZ, RANGE, N, D, E, VL, VU, IL, IU, ABSTOL, M, W, Z, LDZ, WORK, IWORK, IFAIL, INFO)

Expert Driver Routines for Standard and Generalized Nonsymmetric Eigenvalue Problems

Problem Type	Routine
Schur Factorization	SGEESX(JOBVS, SORT, SELECT, SENSE, N, A, LDA, SDIM, WR, WI, VS, LDVS, RCONDE, RCONDV, WORK, LWORK, IWORK, LIWORK, BWORK, INFO)
	CGEESX(JOBVS, SORT, SELECT, SENSE, N, A, LDA, SDIM, W, VS, LDVS, RCONDE, RCONDV, WORK, LWORK, RWORK, BWORK, INFO)
	SGGESX(JOBVSL, JOBVSR, SORT, SELCTG, SENSE, N, A, LDA, B, LDB, SDIM, ALPHAR, ALPHAI, BETA, VSL, LDVSL, VSR, LDVSR, RCONDE, RCONDV, WORK, LWORK, IWORK, LIWORK, BWORK, INFO)
	CGGESX(JOBVSL, JOBVSR, SORT, SELCTG, SENSE, N, A, LDA, B, LDB, SDIM, ALPHAR, ALPHAI, BETA, VSL, LDVSL, VSR, LDVSR, RCONDE, RCONDV, WORK, LWORK, RWORK, IWORK, LIWORK, BWORK, INFO)
Eigenvalues/ vectors	SGEEVX(BALANC, JOBVL, JOBVR, SENSE, N, A, LDA, WR, WI, VL, LDVL, VR, LDVR, ILO, IHI, SCALE, ABNRM, RCONDE, RCONDV, WORK, LWORK, IWORK, INFO)
	CGEEVX(BALANC, JOBVL, JOBVR, SENSE, N, A, LDA, W, VL, LDVL, VR, LDVR, ILO, IHI, SCALE, ABNRM, RCONDE, RCONDV, WORK, LWORK, RWORK, INFO)
	SGGEVX(BALANC, JOBVL, JOBVR, SENSE, N, A, LDA, B, LDB, ALPHAR, ALPHAI, BETA, VL, LDVL, VR, LDVR, ILO, IHI, LSCALE, RSCALE, ABNRM, BBNRM, RCONDE, RCONDV, WORK, LWORK, IWORK, BWORK, INFO)
	CGGEVX(BALANC, JOBVL, JOBVR, SENSE, N, A, LDA, B, LDB, ALPHAR, ALPHAI, BETA, VL, LDVL, VR, LDVR, ILO, IHI, LSCALE, RSCALE, ABNRM, BBNRM, RCONDE, RCONDV, WORK, LWORK, RWORK, IWORK, BWORK, INFO)

Meaning of prefixes

Routines beginning with "S" are available in:

S - REAL

D - DOUBLE PRECISION

Routines beginning with "C" are available in:

C - COMPLEX

Z - COMPLEX*16

Note: **COMPLEX*16** may not be supported by all machines