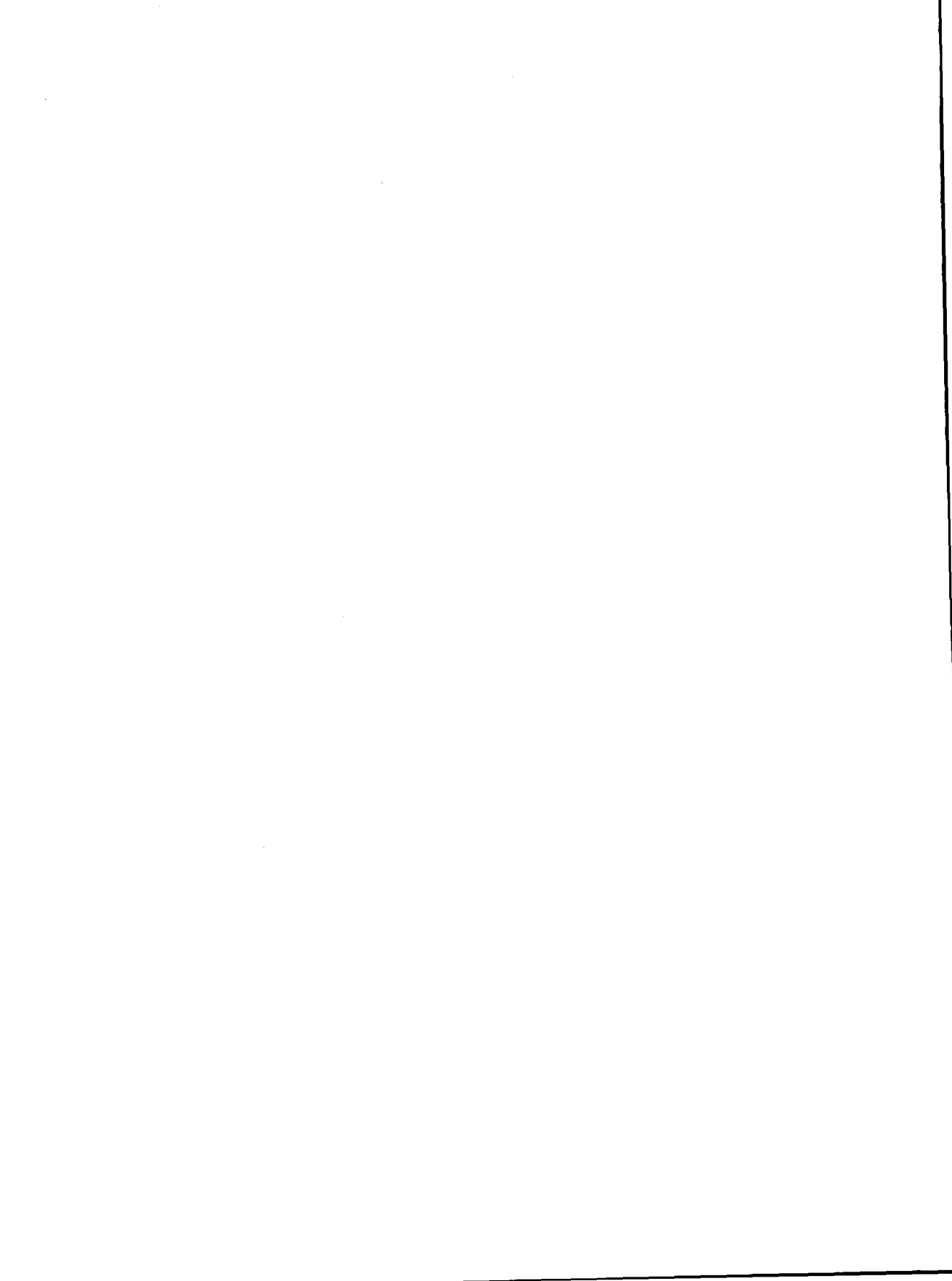


CONVEX

▪ ConvexMLIB
▪ User's Guide: LAPACK

Second Edition



CONVEX Computer Corporation
3000 Waterview Parkway
P.O. Box 833851
Richardson, TX 75083-3851
United States of America
(214)497-4000



ConvexMLIB User's Guide: LAPACK

Document No. 720-005630-005

Second Edition
September 1995

CONVEX Computer Corporation
Richardson, Texas USA

ConvexMLIB User's Guide: LAPACK
Order No. DSW-866
Second Edition

© 1993-95 CONVEX Computer Corporation
All rights reserved.

This document is copyrighted. This document may not, in whole or part, be copied, duplicated, reproduced, translated, stored electronically, or reduced to machine-readable form without prior written consent from CONVEX Computer Corporation.

Although the material contained herein has been carefully reviewed, CONVEX Computer Corporation (CONVEX) does not warrant it to be free of errors or omissions. CONVEX reserves the right to make corrections, updates, revisions or changes to the information contained herein. CONVEX does not warrant the material described herein to be free of patent infringement.

UNLESS PROVIDED OTHERWISE IN WRITING WITH CONVEX COMPUTER CORPORATION (CONVEX), THE PROGRAM DESCRIBED HEREIN IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. SOME STATES DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES. THE ABOVE EXCLUSION MAY NOT BE APPLICABLE TO ALL PURCHASERS BECAUSE WARRANTY RIGHTS CAN VARY FROM STATE TO STATE. IN NO EVENT WILL CONVEX BE LIABLE TO ANYONE FOR SPECIAL, COLLATERAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES, INCLUDING ANY LOST PROFITS OR LOST SAVINGS, ARISING OUT OF THE USE OR INABILITY TO USE THIS PROGRAM. CONVEX WILL NOT BE LIABLE EVEN IF IT HAS BEEN NOTIFIED OF THE POSSIBILITY OF SUCH DAMAGE BY THE PURCHASER OR ANY THIRD PARTY.

CONVEX and the CONVEX logo ("C") are registered trademarks of CONVEX Computer Corporation.
C1, C2, C3, C4, C Series, Exemplar, and ASAP are trademarks of CONVEX Computer Corporation.
ConvexOS and ConvexMLIB are trademarks of CONVEX Computer Corporation.
IEEE is a trademark of the Institute of Electrical and Electronics Engineers, Inc.
SPP-UX is a trademark of CONVEX Computer Corporation.

This entire book is recyclable.

Printed in the United States of America

Revision information for
ConvexMLIB User's Guide: LAPACK

Edition	Document No.	Description
Second	720-005630-005	Released with ConvexMLIB: Exemplar Edition V3.0, September 1995. Conforms to public domain LAPACK version 2.0. New subroutines added to chapters 7 and 9. Corrected miscellaneous errors.
First	720-005630-004	Released with ConvexMLIB: Exemplar Edition V2.0, ConvexMLIB: C Series Edition V9.0, and ConvexMLIB for PA-RISC workstations; October 1994. Former edition (First) was published under the title <i>LAPACK User's Guide</i> . This manual is consistent with public domain LAPACK version 1.1.

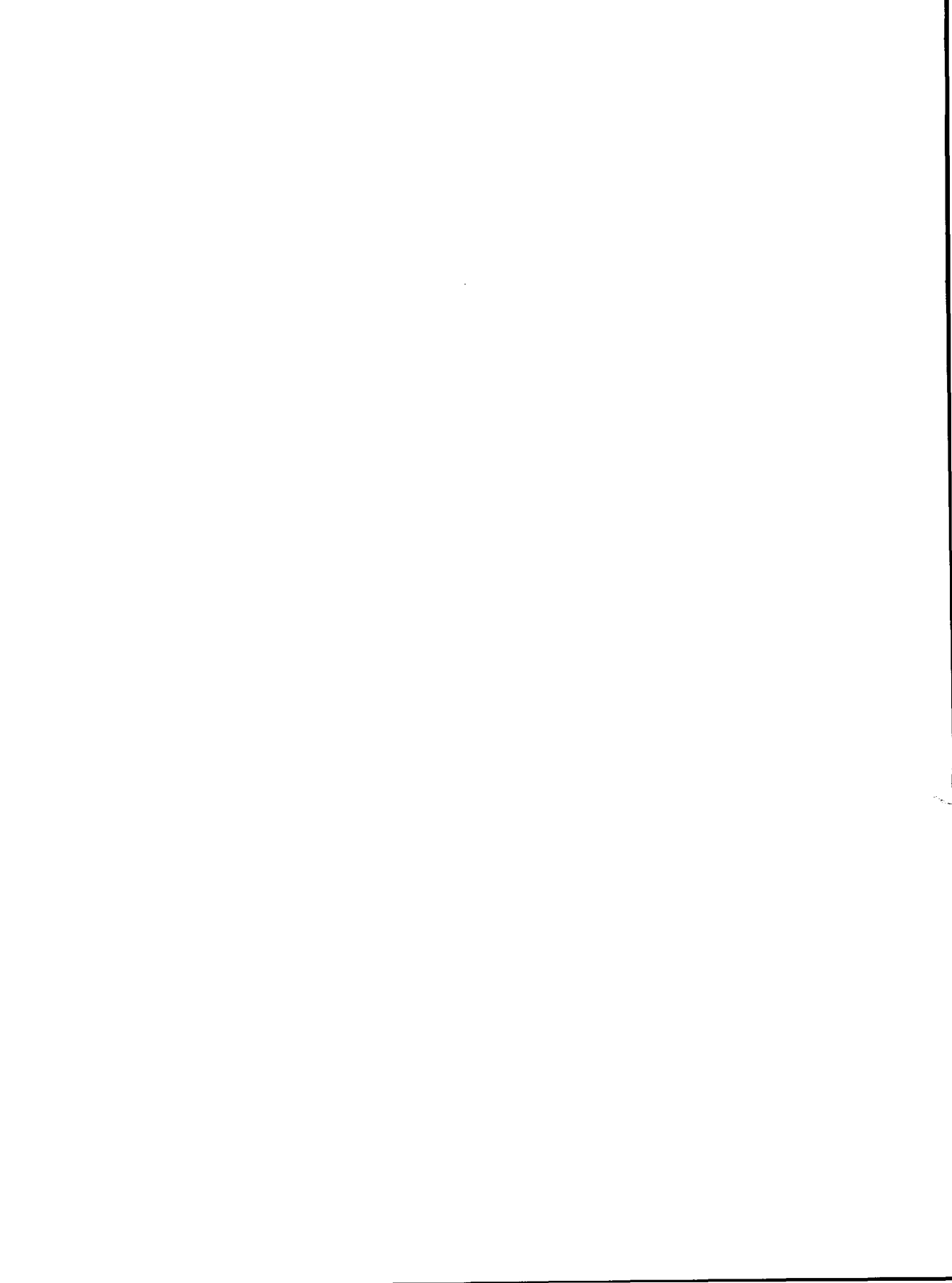


Table of Contents

1 Introduction to LAPACK

Overview	1-1
Chapter Objectives	1-1
What You Need to Know to Use LAPACK	1-2
Standardization	1-2
Two LAPACK Libraries	1-2
Accessing LAPACK	1-2
Interactions Between VECLIB, SCILIB, and LAPACK	1-3
Optimization	1-3
Parallel Processing	1-3
Profiling LAPACK Applications	1-4
Floating-Point Formats	1-5
Roundoff Effects	1-5
LAPACK Organization and Naming Convention	1-6
Required Data Item Byte Lengths and How to Get Them	1-13
Error Handling	1-14
ConvexMLIB Man Pages: LAPACK	1-15
Support Services	1-15
Suggestions	1-16
Supplemental Reading	1-16

2 Simple Drivers for Linear Equations

Overview	2-1
Chapter Objectives	2-1
What You Need to Know to Use These Subprograms	2-1
Supplemental Reading	2-1
Subprogram Descriptions	2-2

3 Expert Drivers for Linear Equations

Overview	3-1
Chapter Objectives	3-1
What You Need to Know to Use These Subprograms	3-1
Condition Number	3-2
Equilibration	3-3
Iterative Refinement	3-3
Matrix Inversion	3-4
Supplemental Reading	3-4
Subprogram Descriptions	3-5

4 Computational Subprograms for Linear Equations

Overview	4-1
Chapter Objectives	4-1
What You Need to Know to Use These Subprograms	4-1
Condition Number	4-2
Matrix Inversion	4-2
Combining Computational Subprograms	4-2
Supplemental Reading	4-5
Subprogram Descriptions	4-5
Subprograms not in this Guide	4-110

5 Drivers for Linear Least Squares Problems	
Overview	5-1
Chapter Objectives	5-1
What You Need to Know to Use These Subprograms	5-1
The Linear Least Squares Problem	5-1
The Method of Normal Equations	5-2
Supplemental Reading	5-2
Subprogram Descriptions	5-3
6 Computational Subprograms for Orthogonal Factorizations	
Overview	6-1
Chapter Objectives	6-1
What You Need to Know to Use These Subprograms	6-1
Combining Computational Subprograms	6-1
Supplemental Reading	6-3
Subprogram Descriptions	6-3
7 Simple Drivers for Ordinary Eigenvalue Problems	
Overview	7-1
Chapter Objectives	7-1
What You Need to Know to Use These Subprograms	7-1
Supplemental Reading	7-2
Subprogram Descriptions	7-2
8 Expert Drivers for Ordinary Eigenvalue Problems	
Overview	8-1
Chapter Objectives	8-1
What You Need to Know to Use These Subprograms	8-1
Supplemental Reading	8-2
Subprogram Descriptions	8-2
9 Drivers for Generalized Eigenvalue Problems	
Overview	9-1
Chapter Objectives	9-1
What You Need to Know to Use These Subprograms	9-1
Supplemental Reading	9-2
Subprogram Descriptions	9-2
10 Drivers for the Singular Value Decomposition	
Overview	10-1
Chapter Objectives	10-1
What You Need to Know to Use These Subprograms	10-1
Supplemental Reading	10-1
Subprogram Descriptions	10-1
11 LAPACK Auxiliary Subprograms	
Overview	11-1
Chapter Objectives	11-1
What You Need to Know to Use These Subprograms	11-1
Norms of Vectors and Matrices	11-1
Supplemental Reading	11-3
Subprogram Descriptions	11-3

List of Tables

1-1 LAPACK Naming Convention—Data Type	1-6
1-2 LAPACK Naming Convention—Matrix Form	1-7
1-3 LAPACK Naming Convention—Computation	1-8
1-4 Driver Subprograms	1-10
1-5 Computational Subprograms for Linear Equations	1-10
1-6 Computational Subprograms for Least Squares	1-11
1-7 Computational Subprograms for Eigenvalue Problems	1-11
1-8 Computational Subprograms for Singular Value Decomposition	1-12
1-9 LAPACK User-Visible Auxiliary Subprograms	1-12
1-10 Data Item Byte Length vs. Data Type and Library	1-13
1-11 Data Item Byte Length vs. Declaration and Compiler Option	1-14
4-1 Subprograms not in this Guide	4-110
11-1 Norms of Vectors and Matrices	11-2

Permuted Index

solve a general	band linear system	2-SGBSV
solve a positive definite	band linear system	2-SPBSV
solve a general	band linear system	3-SGBSVX
solve a positive definite	band linear system	3-SPBSVX
solve a general	band linear system	4-SGBTRS
solve a positive definite	band linear system	4-SPBTRS
solve a triangular	band linear system	4-STBTRS
and eigenvectors with symmetric or Hermitian	band matrices generalized eigenvalues	9-SSBGV
compute a norm of a general	band matrix	11-SLANGB
compute a norm of a symmetric or Hermitian	band matrix	11-SLANSB
estimate the condition number of a general	band matrix	4-SGBCON
factor a general	band matrix	4-SGBTRF
estimate the condition number of a positive definite	band matrix	4-SPBCON
factor a positive definite	band matrix	4-SPBTRF
estimate the condition number of a triangular	band matrix	4-STBCON
and eigenvectors of a symmetric or Hermitian	band matrix all eigenvalues	7-SSBEV
and eigenvectors of a symmetric or Hermitian	band matrix all eigenvalues	7-SSBEVD
and eigenvectors of a symmetric or Hermitian	band matrix selected eigenvalues	8-SSBEVX
subprograms	choose problem-dependent parameters for LAPACK	11-ILAENV
QR factorization of a general matrix with	column pivoting	6-SGEQPF
solve a general least squares problem using	complete orthogonal factorization	5-SGELSX
estimate the	condition number of a general band matrix	4-SGBCON
estimate the	condition number of a general full matrix	4-SGECON
estimate the	condition number of a general tridiagonal matrix	4-SGTCON
estimate the	condition number of a positive definite band matrix	4-SPBCON
estimate the	condition number of a positive definite full matrix	4-SPOCON
in packed form estimate the	condition number of a positive definite matrix stored	4-SPPCON
matrix estimate the	condition number of a positive definite tridiagonal	4-SPTCON
estimate the	condition number of a symmetric or Hermitian matrix	4-SSYCON
stored in packed form estimate the	condition number of a symmetric or Hermitian matrix	4-SSPCON
estimate the	condition number of a triangular band matrix	4-STBCON
estimate the	condition number of a triangular full matrix	4-STRCON
packed form estimate the	condition number of a triangular matrix stored in	4-STPCON
return machine	constants for LAPACK subprograms	11-SLAMCH
a general least squares problem with linear equality	constraints solve	5-SGGLSE
a general least squares problem using singular value	decomposition solve	5-SGELSS
singular value	decomposition of a general matrix	10-SGGSVD
generalized singular value	decomposition of general matrices	10-SGGSVD
all	eigenvalues and eigenvectors of a general matrix	7-SGEEV
selected	eigenvalues and eigenvectors of a general matrix	8-SGEEVX
tridiagonal matrix all	eigenvalues and eigenvectors of a real symmetric	7-SSTEVD
tridiagonal matrix all	eigenvalues and eigenvectors of a real symmetric	7-SSTEVD
tridiagonal matrix selected	eigenvalues and eigenvectors of a real symmetric	8-SSTEVD
Hermitian band matrix all	eigenvalues and eigenvectors of a symmetric or	7-SSBEV
Hermitian band matrix all	eigenvalues and eigenvectors of a symmetric or	7-SSBEVD
Hermitian band matrix selected	eigenvalues and eigenvectors of a symmetric or	8-SSBEVX
Hermitian matrix all	eigenvalues and eigenvectors of a symmetric or	7-SSYEV
Hermitian matrix all	eigenvalues and eigenvectors of a symmetric or	7-SSYEV
Hermitian matrix selected	eigenvalues and eigenvectors of a symmetric or	8-SSYEVX
Hermitian packed matrix all	eigenvalues and eigenvectors of a symmetric or	7-SSPEV
Hermitian packed matrix all	eigenvalues and eigenvectors of a symmetric or	7-SSPEVD
Hermitian packed matrix selected	eigenvalues and eigenvectors of a symmetric or	8-SSPEVX
generalized	eigenvalues and eigenvectors with general matrices	9-SGEGV
Hermitian band matrices generalized	eigenvalues and eigenvectors with symmetric or	9-SSBGV
Hermitian matrices generalized	eigenvalues and eigenvectors with symmetric or	9-SSYGV
Hermitian packed matrices generalized	eigenvalues and eigenvectors with symmetric or	9-SSPGV
all	eigenvalues and Schur Form of a general matrix	7-SGEES
selected	eigenvalues and Schur Form of a general matrix	8-SGEESX
generalized	eigenvalues and Schur Form with general matrices	9-SGEGS
all eigenvalues and	eigenvectors of a general matrix	7-SGEEV
selected eigenvalues and	eigenvectors of a general matrix	8-SGEEVX
all eigenvalues and	eigenvectors of a real symmetric tridiagonal matrix	7-SSTEVD
all eigenvalues and	eigenvectors of a real symmetric tridiagonal matrix	7-SSTEVD
selected eigenvalues and	eigenvectors of a real symmetric tridiagonal matrix	8-SSTEVD
all eigenvalues and	eigenvectors of a symmetric or Hermitian band matrix	7-SSBEV
all eigenvalues and	eigenvectors of a symmetric or Hermitian band matrix	7-SSBEVD

selected eigenvalues and	eigenvectors of a symmetric or Hermitian band matrix	8-SSBEVX
all eigenvalues and	eigenvectors of a symmetric or Hermitian matrix	7-SSYEV
all eigenvalues and	eigenvectors of a symmetric or Hermitian matrix	7-SSYEV
selected eigenvalues and	eigenvectors of a symmetric or Hermitian matrix	8-SSYEVX
matrix all eigenvalues and	eigenvectors of a symmetric or Hermitian packed	7-SSPEV
matrix all eigenvalues and	eigenvectors of a symmetric or Hermitian packed	7-SSPEVD
matrix selected eigenvalues and	eigenvectors of a symmetric or Hermitian packed	8-SSPEVX
generalized eigenvalues and	eigenvectors with general matrices	9-SGEGV
matrices generalized eigenvalues and	eigenvectors with symmetric or Hermitian band	9-SSBGV
generalized eigenvalues and	eigenvectors with symmetric or Hermitian matrices	9-SSYGV
matrices generalized eigenvalues and	eigenvectors with symmetric or Hermitian packed	9-SSPGV
solve a general least squares problem with linear	equality constraints	5-SGGLSE
LAPACK	error handler	11-XERBLA
matrix	estimate the condition number of a general band	4-SGBCON
matrix	estimate the condition number of a general full	4-SGECON
tridiagonal matrix	estimate the condition number of a general	4-SGTCON
band matrix	estimate the condition number of a positive definite	4-SBBCON
full matrix	estimate the condition number of a positive definite	4-SPOCON
matrix stored in packed form	estimate the condition number of a positive definite	4-SPPCON
tridiagonal matrix	estimate the condition number of a positive definite	4-SPTCON
Hermitian matrix	estimate the condition number of a symmetric or	4-SSYCON
Hermitian matrix stored in packed form	estimate the condition number of a symmetric or	4-SSPCON
matrix	estimate the condition number of a triangular band	4-STBCON
matrix	estimate the condition number of a triangular full	4-STRCON
stored in packed form	estimate the condition number of a triangular matrix	4-STPCON
	factor a general band matrix	4-SGBTRF
	factor a general full matrix	4-SGETRF
	factor a general tridiagonal matrix	4-SGTRF
	factor a positive definite band matrix	4-SPBTRF
	factor a positive definite full matrix	4-SPTTRF
form	factor a positive definite matrix stored in packed	4-SPPTRF
	factor a positive definite tridiagonal matrix	4-SPTRF
	factor a symmetric or Hermitian matrix	4-SSYTRF
packed form	factor a symmetric or Hermitian matrix stored in	4-SSPTRF
a general least squares problem using orthogonal	factorization solve	5-SGELS
least squares problem using complete orthogonal	factorization solve a general	5-SGELSX
Q matrix in unfactored form from an LQ	factorization generate the	6-SORGLQ
orthogonal matrix in unfactored form from an LQ	factorization generate the	6-SORGL
the Q matrix in unfactored form from a QL	factorization generate	6-SORGLQ
orthogonal matrix in unfactored form from a QL	factorization generate the	6-SORGLQ
the Q matrix in unfactored form from a QR	factorization generate	6-SORQR
orthogonal matrix in unfactored form from a QR	factorization generate the	6-SORQR
Q matrix in unfactored form from an RQ	factorization generate the	6-SORGRQ
orthogonal matrix in unfactored form from an RQ	factorization generate the	6-SORGRQ
general matrix by the Q matrix from an LQ	factorization multiply a	6-SORMLQ
matrix by the orthogonal matrix from an LQ	factorization multiply a general	6-SORMLQ
general matrix by the Q matrix from a QL	factorization multiply a	6-SORMQL
matrix by the orthogonal matrix from a QL	factorization multiply a general	6-SORMQL
general matrix by the Q matrix from a QR	factorization multiply a	6-SORMQR
matrix by the orthogonal matrix from a QR	factorization multiply a general	6-SORMQR
general matrix by the Q matrix from an RQ	factorization multiply a	6-SORMRQ
matrix by the orthogonal matrix from an RQ	factorization multiply a general	6-SORMRQ
	LQ factorization of a general matrix	6-SGELQF
	QL factorization of a general matrix	6-SGEQLF
	QR factorization of a general matrix	6-SGEQRF
	RQ factorization of a general matrix	6-SGERQF
pivoting QR	factorization of a general matrix with column	6-SGEQPF
RQ	factorization of a upper trapezoidal matrix	6-STZRQF
generalized QR	factorization of general matrices	6-SGGQRF
generalized QR	factorization of general matrices	6-SGGQRF
generalized RQ	factorization of general matrices	6-SGGQRF
generalized RQ	factorization of general matrices	6-SGGQRF
the Q matrix in unfactored form from an	LQ factorization generate	6-SORGLQ
the orthogonal matrix in unfactored form from an	LQ factorization generate	6-SORGLQ
a general matrix by the Q matrix from an	LQ factorization multiply	6-SORMLQ
a general matrix by the orthogonal matrix from an	LQ factorization multiply	6-SORMLQ
	LQ factorization of a general matrix	6-SGELQF
multiply a general matrix by the	Q matrix from a QL factorization	6-SORMQL
multiply a general matrix by the	Q matrix from a QR factorization	6-SORMQR
multiply a general matrix by the	Q matrix from an LQ factorization	6-SORMLQ
multiply a general matrix by the	Q matrix from an RQ factorization	6-SORMRQ

factorization generate the	Q matrix in unfactored form from a QL	6-SORGQL
factorization generate the	Q matrix in unfactored form from a QR	6-SORGQR
factorization generate the	Q matrix in unfactored form from an LQ	6-SORGLQ
factorization generate the	Q matrix in unfactored form from an RQ	6-SORGRQ
generate the Q matrix in unfactored form from a	QL factorization	6-SORGQL
the orthogonal matrix in unfactored form from a	QR factorization generate	6-SORGQR
a general matrix by the Q matrix from a	QL factorization multiply	6-SORMQL
a general matrix by the orthogonal matrix from a	QL factorization multiply	6-SORMQL
	QL factorization of a general matrix	6-SGEQLF
generate the Q matrix in unfactored form from a	QR factorization	6-SORGQR
the orthogonal matrix in unfactored form from a	QR factorization generate	6-SORGQR
a general matrix by the Q matrix from a	QR factorization multiply	6-SORMQR
a general matrix by the orthogonal matrix from a	QR factorization multiply	6-SORMQR
	QR factorization of a general matrix	6-SGEQRF
column pivoting	QR factorization of a general matrix with	6-SGEQPF
generalized	QR factorization of general matrices	6-SGGQRF
generalized	QR factorization of general matrices	6-SGGQRF
the Q matrix in unfactored form from an	RQ factorization generate	6-SORGRQ
the orthogonal matrix in unfactored form from an	RQ factorization generate	6-SORGRQ
a general matrix by the Q matrix from an	RQ factorization multiply	6-SORMRQ
a general matrix by the orthogonal matrix from an	RQ factorization multiply	6-SORMRQ
	RQ factorization of a general matrix	6-SGERQF
	RQ factorization of a upper trapezoidal matrix	6-STZRQF
generalized	RQ factorization of general matrices	6-SGGRQF
generalized	RQ factorization of general matrices	6-SGGRQF
compute a norm of a	full general matrix	11-SLANGE
solve a general	full linear system	2-SGESV
solve a positive definite	full linear system	2-SPOSV
solve a symmetric or Hermitian	full linear system	2-SSYSV
solve a general	full linear system	3-SGESVX
solve a positive definite	full linear system	3-SPOSVX
solve a symmetric	full linear system	3-SSYSVX
solve a general	full linear system	4-SGETRS
solve a positive definite	full linear system	4-SPOTRS
solve a triangular	full linear system	4-SITRTR
estimate the condition number of a general	full matrix	4-SGECON
factor a general	full matrix	4-SGETRF
invert a general	full matrix	4-SGETRI
estimate the condition number of a positive definite	full matrix	4-SPOCON
factor a positive definite	full matrix	4-SPOTRF
invert a positive definite	full matrix	4-SPOTRI
estimate the condition number of a triangular	full matrix	4-STRCON
invert a triangular	full matrix	4-SITRTR
solve a	general band linear system	2-SGBSV
solve a	general band linear system	3-SGBSVX
solve a	general band linear system	4-SGBTRS
compute a norm of a	general band matrix	11-SLANGB
estimate the condition number of a	general band matrix	4-SGBCON
factor a	general band matrix	4-SGBTRF
solve a	general full linear system	2-SGESV
solve a	general full linear system	3-SGESVX
solve a	general full linear system	4-SGETRS
estimate the condition number of a	general full matrix	4-SGECON
factor a	general full matrix	4-SGETRF
invert a	general full matrix	4-SGETRI
orthogonal factorization solve a	general least squares problem using complete	5-SGELSX
factorization solve a	general least squares problem using orthogonal	5-SGELS
decomposition solve a	general least squares problem using singular value	5-SGELSS
constraints solve a	general least squares problem with linear equality	5-SGGLSE
generalized singular value decomposition of	general matrices	10-SGGSVD
generalized QR factorization of	general matrices	6-SGGQRF
generalized QR factorization of	general matrices	6-SGGQRF
generalized RQ factorization of	general matrices	6-SGGRQF
generalized RQ factorization of	general matrices	6-SGGRQF
generalized eigenvalues and Schur Form with	general matrices	9-SGEGS
generalized eigenvalues and eigenvectors with	general matrices	9-SGEGV
singular value decomposition of a	general matrix	10-SGESVD
compute a norm of a full	general matrix	11-SLANGE
LQ factorization of a	general matrix	6-SGELQF
QL factorization of a	general matrix	6-SGEQLF
QR factorization of a	general matrix	6-SGEQRF

<i>RQ</i> factorization of a	general matrix	6-SGERQF
all eigenvalues and Schur Form of a	general matrix	7-SGEES
all eigenvalues and eigenvectors of a	general matrix	7-SGEEV
selected eigenvalues and Schur Form of a	general matrix	8-SGEESEX
selected eigenvalues and eigenvectors of a	general matrix	8-SGEEVX
factorization multiply a	general matrix by the <i>Q</i> matrix from a <i>QL</i>	6-SORMQL
factorization multiply a	general matrix by the <i>Q</i> matrix from a <i>QR</i>	6-SORMQR
factorization multiply a	general matrix by the <i>Q</i> matrix from an <i>LQ</i>	6-SORMLQ
factorization multiply a	general matrix by the <i>Q</i> matrix from an <i>RQ</i>	6-SORMRQ
<i>QL</i> factorization multiply a	general matrix by the orthogonal matrix from a	6-SORMQL
<i>QR</i> factorization multiply a	general matrix by the orthogonal matrix from a	6-SORMQR
<i>LQ</i> factorization multiply a	general matrix by the orthogonal matrix from an	6-SORMLQ
<i>RQ</i> factorization multiply a	general matrix by the orthogonal matrix from an	6-SORMRQ
<i>QR</i> factorization of a	general matrix with column pivoting	6-SGEQPF
solve a	general tridiagonal linear system	2-SGTSV
solve a	general tridiagonal linear system	3-SGTSVX
solve a	general tridiagonal linear system	4-SGTTRS
compute a norm of a	general tridiagonal matrix	11-SLANGT
estimate the condition number of a	general tridiagonal matrix	4-SGTCON
factor a	general tridiagonal matrix	4-SGTTRF
matrices	generalized eigenvalues and eigenvectors with general	9-SGEGV
symmetric or Hermitian band matrices	generalized eigenvalues and eigenvectors with	9-SSBGV
symmetric or Hermitian matrices	generalized eigenvalues and eigenvectors with	9-SSYGV
symmetric or Hermitian packed matrices	generalized eigenvalues and eigenvectors with	9-SSPGV
matrices	generalized eigenvalues and Schur Form with general	9-SGEGS
matrices	generalized <i>QR</i> factorization of general	6-SGGQRF
matrices	generalized <i>QR</i> factorization of general	6-SGGQRF
matrices	generalized <i>RQ</i> factorization of general	6-SGGRQF
matrices	generalized <i>RQ</i> factorization of general	6-SGGRQF
solve a	generalized linear regression model	5-SGGGLM
matrices	generalized singular value decomposition of general	10-SGGSVD
<i>QL</i> factorization	generate the <i>Q</i> matrix in unfactored form from a	6-SORGLQ
<i>QR</i> factorization	generate the <i>Q</i> matrix in unfactored form from a	6-SORGLQ
an <i>LQ</i> factorization	generate the <i>Q</i> matrix in unfactored form from	6-SORGLQ
an <i>RQ</i> factorization	generate the <i>Q</i> matrix in unfactored form from	6-SORGLQ
from a <i>QL</i> factorization	generate the orthogonal matrix in unfactored form	6-SORGQL
from a <i>QR</i> factorization	generate the orthogonal matrix in unfactored form	6-SORGQR
from an <i>LQ</i> factorization	generate the orthogonal matrix in unfactored form	6-SORGLQ
from an <i>RQ</i> factorization	generate the orthogonal matrix in unfactored form	6-SORGQR
LAPACK error	handler	11-XERBLA
eigenvalues and eigenvectors with symmetric or	Hermitian band matrices generalized	9-SSBGV
compute a norm of a symmetric or	Hermitian band matrix	11-SLANSB
all eigenvalues and eigenvectors of a symmetric or	Hermitian band matrix	7-SSBEV
all eigenvalues and eigenvectors of a symmetric or	Hermitian band matrix	7-SSBEVD
eigenvalues and eigenvectors of a symmetric or	Hermitian band matrix selected	8-SSBEVX
solve a symmetric or	Hermitian full linear system	2-SSYSV
solve a symmetric or	Hermitian linear system	4-SSYTRS
solve a symmetric or	Hermitian linear system stored in packed form	2-SSPSV
solve a symmetric or	Hermitian linear system stored in packed form	3-SSPSVX
eigenvalues and eigenvectors with symmetric or	Hermitian matrices generalized	9-SSYGV
compute a norm of a symmetric or	Hermitian matrix	11-SLANSY
estimate the condition number of a symmetric or	Hermitian matrix	4-SSYCON
factor a symmetric or	Hermitian matrix	4-SSYTRF
invert a symmetric or	Hermitian matrix	4-SSYTRI
all eigenvalues and eigenvectors of a symmetric or	Hermitian matrix	7-SSYEV
all eigenvalues and eigenvectors of a symmetric or	Hermitian matrix	7-SSYEVD
eigenvalues and eigenvectors of a symmetric or	Hermitian matrix selected	8-SSYEVX
compute a norm of a symmetric or	Hermitian matrix stored in packed form	11-SLANSP
estimate the condition number of a symmetric or	Hermitian matrix stored in packed form	4-SSPCON
factor a symmetric or	Hermitian matrix stored in packed form	4-SSPTRF
invert a symmetric or	Hermitian matrix stored in packed form	4-SSPTRI
solve a symmetric or	Hermitian packed linear system	4-SSPTRS
eigenvalues and eigenvectors with symmetric or	Hermitian packed matrices generalized	9-SSPGV
all eigenvalues and eigenvectors of a symmetric or	Hermitian packed matrix	7-SSPEV
all eigenvalues and eigenvectors of a symmetric or	Hermitian packed matrix	7-SSPEVD
eigenvalues and eigenvectors of a symmetric or	Hermitian packed matrix selected	8-SSPEVX
compute a norm of a symmetric or	Hermitian tridiagonal matrix	11-SLANST
invert a general full matrix	invert a general full matrix	4-SGETRI
invert a positive definite full matrix	invert a positive definite full matrix	4-SPOTRI
form	invert a positive definite matrix stored in packed	4-SPPTRI
invert a symmetric or Hermitian matrix	invert a symmetric or Hermitian matrix	4-SSYTRI

packed form	invert a symmetric or Hermitian matrix stored in	4-SSPTRI
	invert a triangular full matrix	4-STRTRI
	invert a triangular matrix stored in packed form	4-STPTRI
	LAPACK error handler	11-XERBLA
choose problem-dependent parameters for	LAPACK subprograms	11-LAENV
return machine constants for	LAPACK subprograms	11-SLAMCH
return machine-dependent parameters for	LAPACK subprograms	11-SLAMCH
factorization solve a general	least squares problem using complete orthogonal	5-SGELSX
solve a general	least squares problem using orthogonal factorization	5-SGELS
decomposition solve a general	least squares problem using singular value	5-SGELSS
constraints solve a general	least squares problem with linear equality	5-SGGLSE
solve a general least squares problem with	linear equality constraints	5-SGGLSE
solve a generalized	linear regression model	5-SGGGLM
solve a general band	linear system	2-SGBSV
solve a general full	linear system	2-SGESV
solve a general tridiagonal	linear system	2-SGTSV
solve a positive definite band	linear system	2-SPBSV
solve a positive definite full	linear system	2-SPOSV
solve a positive definite tridiagonal	linear system	2-SPTSV
solve a symmetric or Hermitian full	linear system	2-SSYSV
solve a general band	linear system	3-SGBSVX
solve a general full	linear system	3-SGESVX
solve a general tridiagonal	linear system	3-SGTSVX
solve a positive definite band	linear system	3-SPBSVX
solve a positive definite full	linear system	3-SPOSVX
solve a positive definite tridiagonal	linear system	3-SPTSVX
solve a symmetric full	linear system	3-SSYSVX
solve a general band	linear system	4-SGBTRS
solve a general full	linear system	4-SGETRS
solve a general tridiagonal	linear system	4-SGTTRS
solve a positive definite band	linear system	4-SPBTRS
solve a positive definite full	linear system	4-SPOTRS
solve a positive definite packed	linear system	4-SPPTRS
solve a positive definite tridiagonal	linear system	4-SPTTRS
solve a symmetric or Hermitian packed	linear system	4-SSPTRS
solve a symmetric or Hermitian	linear system	4-SSYTRS
solve a triangular band	linear system	4-STBTRS
solve a triangular packed	linear system	4-STPTRS
solve a triangular full	linear system	4-STRTRS
solve a positive definite	linear system stored in packed form	2-SPPSV
solve a symmetric or Hermitian	linear system stored in packed form	2-SSPSV
solve a positive definite	linear system stored in packed form	3-SPPSVX
solve a symmetric or Hermitian	linear system stored in packed form	3-SSPSVX
return	machine constants for LAPACK subprograms	11-SLAMCH
return	machine-dependent parameters for LAPACK subprograms	11-SLAMCH
generalized singular value decomposition of general	matrices	10-SGGSVD
generalized <i>QR</i> factorization of general	matrices	6-SGGQRF
generalized <i>QR</i> factorization of general	matrices	6-SGGQRF
generalized <i>RQ</i> factorization of general	matrices	6-SGGRQF
generalized <i>RQ</i> factorization of general	matrices	6-SGGRQF
generalized eigenvalues and Schur Form with general	matrices	9-SGEGS
generalized eigenvalues and eigenvectors with general	matrices	9-SGEGV
and eigenvectors with symmetric or Hermitian band	matrices generalized eigenvalues	9-SGBGV
and eigenvectors with symmetric or Hermitian packed	matrices generalized eigenvalues	9-SSPGV
and eigenvectors with symmetric or Hermitian	matrices generalized eigenvalues	9-SSYGV
singular value decomposition of a general	matrix	10-SGESVD
compute a norm of a general band	matrix	11-SLANGB
compute a norm of a full general	matrix	11-SLANGE
compute a norm of a general tridiagonal	matrix	11-SLANGT
compute a norm of a symmetric or Hermitian band	matrix	11-SLANSB
a norm of a symmetric or Hermitian tridiagonal	matrix compute	11-SLANST
compute a norm of a symmetric or Hermitian	matrix	11-SLANSY
estimate the condition number of a general band	matrix	4-SGBCON
factor a general band	matrix	4-SGBTRF
estimate the condition number of a general full	matrix	4-SGECON
factor a general full	matrix	4-SGETRF
invert a general full	matrix	4-SGETRI
the condition number of a general tridiagonal	matrix estimate	4-SGTCON
factor a general tridiagonal	matrix	4-SGITRF
the condition number of a positive definite band	matrix estimate	4-SPBCON
factor a positive definite band	matrix	4-SPBTRF

the condition number of a positive definite full factor a positive definite full invert a positive definite full	matrix estimate matrix matrix	4-SPOCON 4-SPOTRF 4-SPOTRI
condition number of a positive definite tridiagonal factor a positive definite tridiagonal	matrix estimate the matrix	4-SPTCON 4-SPTTRF
the condition number of a symmetric or Hermitian factor a symmetric or Hermitian invert a symmetric or Hermitian	matrix estimate matrix matrix	4-SSYCON 4-SSYTRF 4-SSYTRI
estimate the condition number of a triangular band estimate the condition number of a triangular full invert a triangular full	matrix matrix matrix	4-STBCON 4-STRCON 4-STRTRI
<i>LQ</i> factorization of a general	matrix	6-SGELQF
<i>QL</i> factorization of a general	matrix	6-SGEQLF
<i>QR</i> factorization of a general	matrix	6-SGEQRF
<i>RQ</i> factorization of a general	matrix	6-SGERQF
<i>RQ</i> factorization of an upper trapezoidal	matrix	6-STZRQF
all eigenvalues and Schur Form of a general	matrix	7-SGEEES
all eigenvalues and eigenvectors of a general	matrix	7-SGEEV
and eigenvectors of a symmetric or Hermitian band	matrix all eigenvalues	7-SSBEV
and eigenvectors of a symmetric or Hermitian band	matrix all eigenvalues	7-SSBEVD
and eigenvectors of a symmetric or Hermitian packed	matrix all eigenvalues	7-SSPEV
and eigenvectors of a symmetric or Hermitian packed	matrix all eigenvalues	7-SSPEVD
and eigenvectors of a real symmetric tridiagonal	matrix all eigenvalues	7-SSTEVD
and eigenvectors of a real symmetric tridiagonal	matrix all eigenvalues	7-SSTEVD
and eigenvectors of a symmetric or Hermitian	matrix all eigenvalues	7-SSYEV
and eigenvectors of a symmetric or Hermitian	matrix all eigenvalues	7-SSYEVD
selected eigenvalues and Schur Form of a general	matrix	8-SGEEEX
selected eigenvalues and eigenvectors of a general	matrix	8-SGEEVX
and eigenvectors of a symmetric or Hermitian band	matrix selected eigenvalues	8-SSBEVX
and eigenvectors of a symmetric or Hermitian packed	matrix selected eigenvalues	8-SSPEVX
and eigenvectors of a real symmetric tridiagonal	matrix selected eigenvalues	8-SSTEVD
and eigenvectors of a real symmetric or Hermitian	matrix selected eigenvalues	8-SSYEVX
factorization multiply a general	matrix by the <i>Q</i> matrix from a <i>QL</i>	6-SORMQL
factorization multiply a general	matrix by the <i>Q</i> matrix from a <i>QR</i>	6-SORMQR
factorization multiply a general	matrix by the <i>Q</i> matrix from an <i>LQ</i>	6-SORMLQ
factorization multiply a general	matrix by the <i>Q</i> matrix from an <i>RQ</i>	6-SORMRQ
factorization multiply a general	matrix by the orthogonal matrix from a <i>QL</i>	6-SORMQL
factorization multiply a general	matrix by the orthogonal matrix from a <i>QR</i>	6-SORMQR
factorization multiply a general	matrix by the orthogonal matrix from an <i>LQ</i>	6-SORMLQ
factorization multiply a general	matrix by the orthogonal matrix from an <i>RQ</i>	6-SORMRQ
multiply a general matrix by the <i>Q</i>	matrix from a <i>QL</i> factorization	6-SORMQL
multiply a general matrix by the orthogonal	matrix from a <i>QL</i> factorization	6-SORMQL
multiply a general matrix by the <i>Q</i>	matrix from a <i>QR</i> factorization	6-SORMQR
multiply a general matrix by the orthogonal	matrix from a <i>QR</i> factorization	6-SORMQR
multiply a general matrix by the <i>Q</i>	matrix from an <i>LQ</i> factorization	6-SORMLQ
multiply a general matrix by the orthogonal	matrix from an <i>LQ</i> factorization	6-SORMLQ
multiply a general matrix by the <i>Q</i>	matrix from an <i>RQ</i> factorization	6-SORMRQ
multiply a general matrix by the orthogonal	matrix from an <i>RQ</i> factorization	6-SORMRQ
factorization generate the <i>Q</i>	matrix in unfactored form from a <i>QL</i>	6-SORGQL
factorization generate the orthogonal	matrix in unfactored form from a <i>QL</i>	6-SORGQL
factorization generate the <i>Q</i>	matrix in unfactored form from a <i>QR</i>	6-SORGQR
factorization generate the orthogonal	matrix in unfactored form from a <i>QR</i>	6-SORGQR
factorization generate the <i>Q</i>	matrix in unfactored form from an <i>LQ</i>	6-SORGLQ
factorization generate the orthogonal	matrix in unfactored form from an <i>LQ</i>	6-SORGLQ
factorization generate the <i>Q</i>	matrix in unfactored form from an <i>RQ</i>	6-SORGRQ
factorization generate the orthogonal	matrix in unfactored form from an <i>RQ</i>	6-SORGRQ
compute a norm of a symmetric or Hermitian	matrix stored in packed form	11-SLANSP
estimate the condition number of a positive definite	matrix stored in packed form	4-SPPCON
factor a positive definite	matrix stored in packed form	4-SPPTRF
invert a positive definite	matrix stored in packed form	4-SPPTRI
the condition number of a symmetric or Hermitian	matrix stored in packed form estimate	4-SSPCON
factor a symmetric or Hermitian	matrix stored in packed form	4-SSPTRF
invert a symmetric or Hermitian	matrix stored in packed form	4-SSPTRI
estimate the condition number of a triangular	matrix stored in packed form	4-STPCON
invert a triangular	matrix stored in packed form	4-STPTRI
<i>QR</i> factorization of a general	matrix with column pivoting	6-SGEQPF
solve a generalized linear regression	model	5-SGGGLM
a <i>QL</i> factorization	multiply a general matrix by the <i>Q</i> matrix from	6-SORMQL
a <i>QR</i> factorization	multiply a general matrix by the <i>Q</i> matrix from	6-SORMQR
an <i>LQ</i> factorization	multiply a general matrix by the <i>Q</i> matrix from	6-SORMLQ
an <i>RQ</i> factorization	multiply a general matrix by the <i>Q</i> matrix from	6-SORMRQ

from a <i>QL</i> factorization	multiply a general matrix by the orthogonal matrix	6-SORMQL
from a <i>QR</i> factorization	multiply a general matrix by the orthogonal matrix	6-SORMQR
from an <i>LQ</i> factorization	multiply a general matrix by the orthogonal matrix	6-SORMLQ
from an <i>RQ</i> factorization	multiply a general matrix by the orthogonal matrix	6-SORMRQ
compute a norm of a full general matrix		11-SLANGE
compute a norm of a general band matrix		11-SLANGB
compute a norm of a general tridiagonal matrix		11-SLANGT
compute a norm of a symmetric or Hermitian band matrix		11-SLANSB
compute a norm of a symmetric or Hermitian matrix		11-SLANSY
packed form compute a norm of a symmetric or Hermitian matrix stored in		11-SLANSP
compute a norm of a symmetric or Hermitian tridiagonal matrix		11-SLANST
solve a general least squares problem using orthogonal factorization		5-SGELS
solve a general least squares problem using complete orthogonal factorization		5-SGELSX
multiply a general matrix by the orthogonal matrix from a <i>QL</i> factorization		6-SORMQL
multiply a general matrix by the orthogonal matrix from a <i>QR</i> factorization		6-SORMQR
multiply a general matrix by the orthogonal matrix from an <i>LQ</i> factorization		6-SORMLQ
multiply a general matrix by the orthogonal matrix from an <i>RQ</i> factorization		6-SORMRQ
factorization generate the orthogonal matrix in unfactored form from a <i>QL</i>		6-SORGQL
factorization generate the orthogonal matrix in unfactored form from a <i>QR</i>		6-SORGQR
factorization generate the orthogonal matrix in unfactored form from an <i>LQ</i>		6-SORGLQ
factorization generate the orthogonal matrix in unfactored form from an <i>RQ</i>		6-SORGRQ
a norm of a symmetric or Hermitian matrix stored in packed form compute		11-SLANSP
solve a positive definite linear system stored in packed form		2-SPPSV
a symmetric or Hermitian linear system stored in packed form solve		2-SPSPV
solve a positive definite linear system stored in packed form		3-SPPSVX
a symmetric or Hermitian linear system stored in packed form solve		3-SPSPVX
number of a positive definite matrix stored in packed form estimate the condition		4-SPPCON
factor a positive definite matrix stored in packed form		4-SPPTRF
invert a positive definite matrix stored in packed form		4-SPPTRI
number of a symmetric or Hermitian matrix stored in packed form estimate the condition		4-SPPCON
factor a symmetric or Hermitian matrix stored in packed form		4-SPPTRF
invert a symmetric or Hermitian matrix stored in packed form		4-SPPTRI
the condition number of a triangular matrix stored in packed form estimate		4-STPCON
invert a triangular matrix stored in packed form		4-STPTRI
solve a positive definite packed linear system		4-SPPTRS
solve a symmetric or Hermitian packed linear system		4-SSPTRS
solve a triangular packed linear system		4-STPTRS
and eigenvectors with symmetric or Hermitian packed matrices generalized eigenvalues		9-SSPGV
and eigenvectors of a symmetric or Hermitian packed matrix all eigenvalues		7-SSPEV
and eigenvectors of a symmetric or Hermitian packed matrix all eigenvalues		7-SSPEVD
and eigenvectors of a symmetric or Hermitian packed matrix selected eigenvalues		8-SSPEVX
choose problem-dependent parameters for LAPACK subprograms		11-ILAENV
return machine-dependent parameters for LAPACK subprograms		11-SLAMCH
factorization of a general matrix with column pivoting <i>QR</i>		6-SGEPQF
solve a positive definite band linear system		2-SPBSV
solve a positive definite band linear system		3-SPBSVX
solve a positive definite band linear system		4-SPBTRS
estimate the condition number of a positive definite band matrix		4-SPBCON
factor a positive definite band matrix		4-SPBTRF
solve a positive definite full linear system		2-SPOSV
solve a positive definite full linear system		3-SPOSVX
solve a positive definite full linear system		4-SPOTRS
estimate the condition number of a positive definite full matrix		4-SPOCON
factor a positive definite full matrix		4-SPOTRF
invert a positive definite full matrix		4-SPOTRI
solve a positive definite linear system stored in packed form		2-SPSPV
solve a positive definite linear system stored in packed form		3-SPSPVX
estimate the condition number of a positive definite matrix stored in packed form		4-SPPCON
factor a positive definite matrix stored in packed form		4-SPPTRF
invert a positive definite matrix stored in packed form		4-SPPTRI
solve a positive definite packed linear system		4-SPPTRS
solve a positive definite tridiagonal linear system		2-SPTSV
solve a positive definite tridiagonal linear system		3-SPTSVX
solve a positive definite tridiagonal linear system		4-SPTTRS
estimate the condition number of a positive definite tridiagonal matrix		4-SPTCON
factor a positive definite tridiagonal matrix		4-SPTTRF
solve a general least squares problem using complete orthogonal factorization		5-SGELSX
solve a general least squares problem using orthogonal factorization		5-SGELS
solve a general least squares problem using singular value decomposition		5-SGELSS
solve a general least squares problem with linear equality constraints		5-SGGLSE
choose problem-dependent parameters for LAPACK subprograms		11-ILAENV

all eigenvalues and eigenvectors of a	real symmetric tridiagonal matrix	7-SSTEVD
all eigenvalues and eigenvectors of a	real symmetric tridiagonal matrix	7-SSTEVD
selected eigenvalues and eigenvectors of a	real symmetric tridiagonal matrix	8-SSTEVSX
solve a generalized linear	regression model	5-SGGGLM
subprograms	return machine constants for LAPACK subprograms	11-SLAMCH
all eigenvalues and	return machine-dependent parameters for LAPACK	11-SLAMCH
selected eigenvalues and	Schur Form of a general matrix	7-SGEES
generalized eigenvalues and	Schur Form of a general matrix	8-SGEESSX
matrix	Schur Form with general matrices	9-SGEGS
symmetric tridiagonal matrix	selected eigenvalues and eigenvectors of a general	8-SGEEVSX
or Hermitian band matrix	selected eigenvalues and eigenvectors of a real	8-SSTEVSX
or Hermitian matrix	selected eigenvalues and eigenvectors of a symmetric	8-SBSEVSX
or Hermitian packed matrix	selected eigenvalues and eigenvectors of a symmetric	8-SSYEVX
matrix	selected eigenvalues and eigenvectors of a symmetric	8-SSPEVSX
solve a general least squares problem using	selected eigenvalues and Schur Form of a general	8-SGEESSX
generalized	singular value decomposition	5-SGELSS
orthogonal factorization	singular value decomposition of a general matrix	10-SGESVD
orthogonal factorization	singular value decomposition of general matrices	10-SGGSVL
value decomposition	solve a general band linear system	2-SGBSV
equality constraints	solve a general band linear system	3-SGBSVX
	solve a general band linear system	4-SGBTRS
	solve a general full linear system	2-SGESV
	solve a general full linear system	3-SGESVSX
	solve a general full linear system	4-SGETRS
	solve a general least squares problem using complete	5-SGELSX
	solve a general least squares problem using	5-SGELS
	solve a general least squares problem using singular	5-SGELSS
	solve a general least squares problem with linear	5-SGGLSE
	solve a general tridiagonal linear system	2-SGTSV
	solve a general tridiagonal linear system	3-SGTSVSX
	solve a general tridiagonal linear system	4-SGTTRS
	solve a generalized linear regression model	5-SGGGLM
	solve a positive definite band linear system	2-SPBSV
	solve a positive definite band linear system	3-SPBSVSX
	solve a positive definite band linear system	4-SPBTRS
	solve a positive definite full linear system	2-SPOSV
	solve a positive definite full linear system	3-SPOSVSX
	solve a positive definite full linear system	4-SPOTRS
packed form	solve a positive definite linear system stored in	2-SPPSV
packed form	solve a positive definite linear system stored in	3-SPPSVX
	solve a positive definite packed linear system	4-SPPTRS
	solve a positive definite tridiagonal linear system	2-SPTSV
	solve a positive definite tridiagonal linear system	3-SPTSVX
	solve a positive definite tridiagonal linear system	4-SPTTRS
	solve a symmetric full linear system	3-SSYSVSX
	solve a symmetric or Hermitian full linear system	2-SSYSV
	solve a symmetric or Hermitian linear system	4-SSYTRS
in packed form	solve a symmetric or Hermitian linear system stored	2-SSPSV
in packed form	solve a symmetric or Hermitian linear system stored	3-SSPSVSX
	solve a symmetric or Hermitian packed linear system	4-SSPTRS
	solve a triangular band linear system	4-STBTRS
	solve a triangular full linear system	4-STRTRS
	solve a triangular packed linear system	4-STPTRS
factorization	solve a general least squares problem using complete orthogonal	5-SGELSX
solve a general least	squares problem using orthogonal factorization	5-SGELS
solve a general least	squares problem using singular value decomposition	5-SGELSS
solve a general least	squares problem with linear equality constraints	5-SGGLSE
compute a norm of a symmetric or Hermitian matrix	stored in packed form	11-SLANSP
solve a positive definite linear system	stored in packed form	2-SPPSV
solve a symmetric or Hermitian linear system	stored in packed form	2-SSPSV
solve a positive definite linear system	stored in packed form	3-SPPSVX
solve a symmetric or Hermitian linear system	stored in packed form	3-SSPSVSX
the condition number of a positive definite matrix	stored in packed form estimate	4-SPPCON
factor a positive definite matrix	stored in packed form	4-SPPTRF
invert a positive definite matrix	stored in packed form	4-SPPTRI
condition number of a symmetric or Hermitian matrix	stored in packed form estimate the	4-SSPCON
factor a symmetric or Hermitian matrix	stored in packed form	4-SSPTRF
invert a symmetric or Hermitian matrix	stored in packed form	4-SSPTRI
estimate the condition number of a triangular matrix	stored in packed form	4-STPCON
invert a triangular matrix	stored in packed form	4-STPTRI
choose problem-dependent parameters for LAPACK	subprograms	11-LAENV

return machine constants for LAPACK	subprograms	11-SLAMCH
return machine-dependent parameters for LAPACK	subprograms	11-SLAMCH
solve a	symmetric full linear system	3-SSYSVX
generalized eigenvalues and eigenvectors with	symmetric or Hermitian band matrices	9-SSBGV
compute a norm of a	symmetric or Hermitian band matrix	11-SLANSB
all eigenvalues and eigenvectors of a	symmetric or Hermitian band matrix	7-SSBEV
all eigenvalues and eigenvectors of a	symmetric or Hermitian band matrix	7-SSBEVD
selected eigenvalues and eigenvectors of a	symmetric or Hermitian band matrix	8-SSBEVX
solve a	symmetric or Hermitian full linear system	2-SSYSV
solve a	symmetric or Hermitian linear system	4-SSYTRS
form solve a	symmetric or Hermitian linear system stored in packed	2-SSPSV
form solve a	symmetric or Hermitian linear system stored in packed	3-SSPSVX
generalized eigenvalues and eigenvectors with	symmetric or Hermitian matrices	9-SSYGV
compute a norm of a	symmetric or Hermitian matrix	11-SLANSY
estimate the condition number of a	symmetric or Hermitian matrix	4-SSYCON
factor a	symmetric or Hermitian matrix	4-SSYTRF
invert a	symmetric or Hermitian matrix	4-SSYTRI
all eigenvalues and eigenvectors of a	symmetric or Hermitian matrix	7-SSYEV
all eigenvalues and eigenvectors of a	symmetric or Hermitian matrix	7-SSYEVD
selected eigenvalues and eigenvectors of a	symmetric or Hermitian matrix	8-SSYEVX
compute a norm of a	symmetric or Hermitian matrix stored in packed form	11-SLANSP
estimate the condition number of a	symmetric or Hermitian matrix stored in packed form	4-SSPCON
factor a	symmetric or Hermitian matrix stored in packed form	4-SSPTRF
invert a	symmetric or Hermitian matrix stored in packed form	4-SSPTRI
solve a	symmetric or Hermitian packed linear system	4-SSPTRS
generalized eigenvalues and eigenvectors with	symmetric or Hermitian packed matrices	9-SSPGV
all eigenvalues and eigenvectors of a	symmetric or Hermitian packed matrix	7-SSPEV
all eigenvalues and eigenvectors of a	symmetric or Hermitian packed matrix	7-SSPEVD
selected eigenvalues and eigenvectors of a	symmetric or Hermitian packed matrix	8-SSPEVX
compute a norm of a	symmetric or Hermitian tridiagonal matrix	11-SLANST
all eigenvalues and eigenvectors of a real	symmetric tridiagonal matrix	7-SSTEV
all eigenvalues and eigenvectors of a real	symmetric tridiagonal matrix	7-SSTEVD
selected eigenvalues and eigenvectors of a real	symmetric tridiagonal matrix	8-SSTEVX
solve a general band linear	system	2-SGBSV
solve a general full linear	system	2-SGESV
solve a general tridiagonal linear	system	2-SGTSV
solve a positive definite band linear	system	2-SPBSV
solve a positive definite full linear	system	2-SPOSV
solve a positive definite tridiagonal linear	system	2-SPTSV
solve a symmetric or Hermitian full linear	system	2-SSYSV
solve a general band linear	system	3-SGBSVX
solve a general full linear	system	3-SGESVX
solve a general tridiagonal linear	system	3-SGTSVX
solve a positive definite band linear	system	3-SPBSVX
solve a positive definite full linear	system	3-SPOSVX
solve a positive definite tridiagonal linear	system	3-SPTSVX
solve a symmetric full linear	system	3-SSYSVX
solve a general band linear	system	4-SGBTRS
solve a general full linear	system	4-SGETRS
solve a general tridiagonal linear	system	4-SGTRRS
solve a positive definite band linear	system	4-SPBTRS
solve a positive definite full linear	system	4-SPOTRS
solve a positive definite packed linear	system	4-SPPTRS
solve a positive definite tridiagonal linear	system	4-SPTRRS
solve a symmetric or Hermitian packed linear	system	4-SSPTRS
solve a symmetric or Hermitian linear	system	4-SSYTRS
solve a triangular band linear	system	4-STBTRS
solve a triangular packed linear	system	4-STPTRS
solve a triangular full linear	system	4-STRTRS
solve a positive definite linear	system stored in packed form	2-SPPSV
solve a symmetric or Hermitian linear	system stored in packed form	2-SSPSV
solve a positive definite linear	system stored in packed form	3-SPPSVX
solve a symmetric or Hermitian linear	system stored in packed form	3-SSPSVX
<i>RQ</i> factorization of a upper	trapezoidal matrix	6-STZRQF
solve a	triangular band linear system	4-STBTRS
estimate the condition number of a	triangular band matrix	4-STBCON
solve a	triangular full linear system	4-STRTRS
estimate the condition number of a	triangular full matrix	4-STRCON
invert a	triangular full matrix	4-STRTRI
estimate the condition number of a	triangular matrix stored in packed form	4-STPCON
invert a	triangular matrix stored in packed form	4-STPTRI

	solve a triangular packed linear system	4-STPTRS
	solve a general tridiagonal linear system	2-SGTSV
	solve a positive definite tridiagonal linear system	2-SPTSX
	solve a general tridiagonal linear system	3-SGTSVX
	solve a positive definite tridiagonal linear system	3-SPTSXV
	solve a general tridiagonal linear system	4-SGTTRS
	solve a positive definite tridiagonal linear system	4-SPTTRS
	compute a norm of a general tridiagonal matrix	11-SLANGT
	compute a norm of a symmetric or Hermitian tridiagonal matrix	11-SLANST
	estimate the condition number of a general tridiagonal matrix	4-SGTCO
	factor a general tridiagonal matrix	4-SGTTRF
	estimate the condition number of a positive definite tridiagonal matrix	4-SPTCO
	factor a positive definite tridiagonal matrix	4-SPTTRF
	all eigenvalues and eigenvectors of a real symmetric tridiagonal matrix	7-SSTEV
	all eigenvalues and eigenvectors of a real symmetric tridiagonal matrix	7-SSTEVD
	eigenvalues and eigenvectors of a real symmetric tridiagonal matrix selected	8-SSTEVX
	generate the Q matrix in unfactored form from a QL factorization	6-SORGQL
	generate the orthogonal matrix in unfactored form from a QL factorization	6-SORGQL
	generate the Q matrix in unfactored form from a QR factorization	6-SORGQR
	generate the orthogonal matrix in unfactored form from a QR factorization	6-SORGQR
	generate the Q matrix in unfactored form from an LQ factorization	6-SORGLQ
	generate the orthogonal matrix in unfactored form from an LQ factorization	6-SORGLQ
	generate the Q matrix in unfactored form from an RQ factorization	6-SORGRQ
	generate the orthogonal matrix in unfactored form from an RQ factorization	6-SORGRQ
	RQ factorization of a upper trapezoidal matrix	6-STZRQF
	solve a general least squares problem using singular value decomposition	5-SGELSS
	singular value decomposition of a general matrix	10-SGESVD
	generalized singular value decomposition of general matrices	10-SGGSD

Preface

Purpose and Audience

This guide describes the CONVEX LAPACK software library and shows how to use it. CONVEX LAPACK (hereafter addressed as LAPACK) is a collection of Fortran-callable subprograms that provides mathematical software for application programs involving linear algebra, including linear equations, least squares, eigenvalue problems, and the singular value decomposition. The name “LAPACK” is an acronym for “Linear Algebra PACKage.” The package is designed to supersede LINPACK and EISPACK. The National Science Foundation, the Defense Advanced Research Projects Agency, and the Department of Energy supported the development of the public-domain version of LAPACK, from which the CONVEX version of LAPACK was derived.

Much of the information in this manual was derived from the source code and its comments, and from various LAPACK working notes. These sources are in the public domain. We especially thank the development team of the public-domain LAPACK package for the accuracy, consistency, and completeness of the source code comments, which served as a machine-readable starting point for the subprogram descriptions in this guide.

The *ConvexMLIB User's Guide: LAPACK* addresses experienced Fortran programmers who:

- convert, develop, or optimize programs for use on CONVEX supercomputers and Hewlett-Packard workstations
- optimize existing software to improve performance and increase productivity

Organization

To learn fundamental information necessary for using the LAPACK library, read Chapter 1 and the introductory sections of the other chapters. These sections of background information will help you efficiently use the LAPACK library subprograms.

To learn more about the subject of any given chapter, refer to the literature cited in the “Supplemental Reading” section of each chapter.

To identify subprograms by function, refer to the Permuted Index which lists subprogram functions and their chapter numbers and names. To find the page number on which the subprogram is described, use the Index or refer to the “Subprogram Descriptions” section in the chapter introduction.

This guide is organized into the following chapters:

- Chapter 1 introduces general concepts about LAPACK.
- Chapter 2 describes simple drivers for linear equations.
- Chapter 3 explains expert drivers for linear equations.
- Chapter 4 describes computational linear equation subprograms.
- Chapter 5 explains the least squares capabilities of LAPACK.
- Chapter 6 describes computational subprograms for computing and using orthogonal factorizations.
- Chapter 7 explains the simple driver subprograms for ordinary eigenanalysis.
- Chapter 8 describes the expert driver subprograms for ordinary eigenanalysis.
- Chapter 9 explains the generalized eigenvalue subprograms.
- Chapter 10 describes singular value decomposition subprograms.
- Chapter 11 describes user-visible auxiliary subprograms in LAPACK.
- An index is included at the back of the manual.

Notational Conventions

The following conventions are used in this manual:

- *Italics* within text indicate mathematical entities used or manipulated by the program: for example, solve the n -by- n system of linear equations $Ax = b$.

Italics within command lines indicate generic commands, file names, or subprogram names. Substitute actual commands, file names, or subprograms for the *italicized* words. For example, the command line

fc prog_name.o

instructs you to type the command *fc*, followed by the name of a program or subprogram object file.

- **UPPERCASE BOLDFACE** within text and in prototype Fortran statements indicates Fortran keywords and subprogram names that must be typed just as they appear: for example, **CALL SGESV**.
- Type in **lowercase boldface** indicates Fortran generic variable or array names. You should substitute actual variable or array names. The *italicized* mathematical entities and the **lowercase boldface** variable and array names usually correspond. For example, *A* will be a matrix and **a** will be the Fortran array containing the matrix:

CALL SGESV (n, nrhs, a, lda, ipvt, b, ldb, info)

Within command lines, **lowercase boldface** indicates ASCII characters that must be typed just as they appear. For example, the command line

fc prog_name.o

instructs you to type the command **fc**, followed by the name of a program or subprogram file.

- **UPPERCASE CONSTANT WIDTH** represents Fortran programs.
- Brackets (**[]**) enclose optional entries.

- The terms *C Series*, *Exemplar*, and *PA-RISC*, found in the **Usage** section of each subprogram description, respectively denote: CONVEX C Series machines; CONVEX Exemplar machines (having parallel PA-RISC processing capability); and other PA-RISC architectures.

Associated Documents

Using this guide successfully may require information not specific to the tasks described herein or not within the scope of this guide. The following CONVEX documents are provided by CONVEX Computer Corporation to help you:

- *Ada User's Guide* (DSW-147). This guide describes the Ada compiler and support tools and reviews basic Ada concepts.
- *C Guide* (DSW-086). This guide describes the CONVEX C compiler.
- *Consultant User's Guide* (DSW-025). This guide describes the functions and operations of the CONVEX *csd* debugger, the post-mortem dump (*pmd*) facility, and the *prof*, *bprof*, and *gprof* profilers.
- *ConvexMLIB User's Guide: VECLIB* (DSW-132). This guide provides information on the subprograms provided with the VECLIB library.
- *ConvexMLIB User's Guide: SCILIB* (DSW-360). This guide provides information on the subprograms and functions provided with the SCILIB library.
- *Exemplar Programming Guide* (DSW-067). This manual describes efficient programming techniques for the Exemplar family of computers.
- *Fortran Language Reference* (DSW-037). This manual is a reference for the Fortran programming language and is designed to provide a thorough working definition of the language.
- *Fortran User's Guide* (DSW-038). This guide tells you how to use the Fortran compiler, including how to compile, load, and execute programs.
- *Fortran Optimization Guide* (DSW-034). This guide describes methods for optimizing Fortran programs.
- *Interlanguage Programming Guide* (DSW-043). This guide explains how to call procedures written in one language from a program written in another language. The languages covered are Fortran, C, C++, and Ada.
- *Performance Analyzer (CXpa) User's Guide* (DSW-251). This guide explains the operation of the CONVEX Performance Analyzer (CXpa) and the steps needed to create and interpret a CXpa profile.

In addition to the CONVEX documents listed above, the following Hewlett-Packard documents are provided by CONVEX to help you with applications on Exemplar and other PA-RISC platforms:

- *HP-UX Assembly Language Reference Manual* (DHP-180). This manual describes the HP-UX Assembler for the PA-RISC processor.
- *PA-RISC 1.1 Architecture and Instruction Set Reference Manual* (DHP-181). This manual describes the architecture and the instruction set of the Hewlett-Packard PA-RISC processor.
- *PA-RISC Procedure Calling Conventions Reference Manual* (DHP-182). This manual describes the conventions for creating PA-RISC assembly language procedure calls.

The LAPACK project produced the following book about LAPACK. It is available directly from the publisher.

- Anderson, E. *et al.* *LAPACK User's Guide*, Second Edition. Philadelphia, PA: Society for Industrial and Applied Mathematics. 1995.

Ordering Documentation

To order the current edition of this or any other CONVEX document, send requests to:

CONVEX Computer Corporation
Customer Service
P.O. Box 833851
Richardson, TX 75083-3851

Customers with Hewlett-Packard workstations should send requests to:

CXSOFI
P.O. Box 833851
Richardson, TX 75083-3851
1-800-CXSOFI1 (U.S. and Canada)
1-214-497-4027 (elsewhere)

Include the order number (beginning with the letters "DSW", "DHW", or "DHP") or the exact title, as listed on the front cover.

Technical Assistance

If you have questions that are not answered in this book, contact the CONVEX Technical Assistance Center (TAC). Use the following phone numbers:

- Within the continental U.S. call 1(800)952-0379.
- Outside the continental U.S. contact the local CONVEX office.

Introduction to LAPACK

Overview

LAPACK, a component of ConvexMLIB, is a collection of Fortran-callable subprograms that provides mathematical software for application programs involving linear algebra, including linear equations, least squares, eigenvalue problems, and the singular value decomposition. The name "LAPACK" is an acronym for "Linear Algebra PACKage." The package is designed to supercede LINPACK and EISPACK. The National Science Foundation, the Defense Advanced Research Projects Agency, and the Department of Energy supported the development of the public-domain version of LAPACK, from which the CONVEX version of LAPACK was derived.

Although LAPACK was designed for use with Fortran programs, C and Ada programs can call LAPACK subprograms, as described in appendices A, B, C, and D of the *ConvexMLIB User's Guide: VECLIB*, which is included in this documentation set.

This chapter provides information necessary for efficient use of LAPACK, including discussions of LAPACK conformance to public-domain LAPACK standards, the LAPACK and LAPACK8 library files, accessing LAPACK subprograms, optimizations, including parallel processing and interactions with other CONVEX analysis and optimization products, supported floating-point formats, roundoff effects, the naming convention, LAPACK library contents, how to use the two libraries and various compiler options, error handling, online documentation, and CONVEX support services.

Chapter Objectives

After reading this chapter you will:

- know why there are two libraries and how to access them
- understand how LAPACK works in a parallel computing environment
- know how LAPACK interacts with the CONVEX Performance Analyzer and other profilers, and CONVEX's two floating-point formats
- know LAPACK naming conventions
- know the overall structure and contents of LAPACK
- be able to use Fortran type declarations and compiler options
- understand how LAPACK handles errors
- know how to access the online *ConvexMLIB Man Pages: LAPACK*
- know what to do if you are having trouble using LAPACK subprograms

What You Need to Know to Use LAPACK

You should be familiar with the following sections to make efficient use of LAPACK.

Standardization

LAPACK fully conforms with public domain version 1.0b of LAPACK in all user-visible usage conventions. (“User-visible” means all subprograms indicated in Tables 1-4 through 1-9 or documented in this manual.) However, even though the user interface is standardized, internal workings of some subprograms have been specialized for supported computers.

Two LAPACK Libraries

Often, it is desirable to run a single precision program in double precision. To support changing the precision without changing the code, CONVEX Fortran Compilers provide several compilation options, namely, `-cfc`, `-p8`, and `-pd8`, that affect the size of Fortran data types. For compatibility with these compiler options, LAPACK provides two libraries, LAPACK and LAPACK8.

- The LAPACK library works with default-sized Fortran INTEGER, REAL, DOUBLE PRECISION, COMPLEX, and LOGICAL data types, that is, the sizes you get when you don't use the “*n” size specifications or any of the above compiler options.
- The LAPACK8 library is a subset of the LAPACK library that is designed for use by programs that are compiled with one of the above compiler options. The LAPACK8 library is available only on computer systems with compilers that support the INTEGER*8 data type.

For more information about these options, refer to a Fortran language reference and the “Required Data Item Byte Lengths and How to Get Them” section in this chapter. To determine if a subprogram is included in LAPACK8, consult the LAPACK8 section under each subprogram specification in the following chapters.

Accessing LAPACK

The LAPACK and LAPACK8 libraries are compiled subprograms ready for you to incorporate into your programs with the linker. Simply include the appropriate declarations and `CALL` statements in your Fortran source program and specify that LAPACK or LAPACK8 be used as an object library at link time by using the `-l` option on the command line; e.g.,

```
fc [options] file -llapack  
or  
fc [options] file -llapack8
```

Do not try to use subprograms from both `-llapack` and `-llapack8` in the same program.

VECLIB is documented in the *ConvexMLIB User's Guide: VECLIB*. If your program uses subprograms from both LAPACK and VECLIB, use one of the following:

```
fc [options] file -llapack -lveclib  
or  
fc [options] file -llapack8 -lveclib8
```

See “Interactions Between VECLIB, SCILIB, and LAPACK” for details about how to order the two `-l` options. Do not try to use subprograms from both `-llapack` and `-lveclib8`, or both `-llapack8` and `-lveclib`, in the same program.

SCILIB is documented in the *ConvexMLIB User's Guide: SCILIB*. If you use subprograms from both LAPACK and SCILIB, access them as follows:

```
fc [options] file -llapack8 -lscilib
```

Add the linker option `-lveclib8` if VECLIB is also used. See "Interactions Between VECLIB, SCILIB, and LAPACK" for details about the order of the `-l` options. Do not try to use subprograms from both `-llapack` and `-lscilib`, in the same program.

Interactions Between VECLIB, SCILIB, and LAPACK

Each of the five library files in VECLIB, SCILIB, and LAPACK is complete in itself, meaning that you will not need to load one library merely because you have used subprograms from another. This is accomplished by including various subprograms in more than one library. For example, subroutine SGEMV is in all of these products, but with identical functionality. Thus, in general, you have to load only the libraries you need, and you may list them in any order on your load command line, as described in the previous section. However, there are a few differences between the libraries that may force you to put the libraries into a specific order to obtain the results you expect.

Differences between VECLIB8 and SCILIB

Five subprograms common to VECLIB8 and SCILIB differ slightly in functionality. Subprograms ICAMAX, ISAMAX, ISAMIN, ISMAX, and ISMIN in VECLIB8 handle a negative `incx` argument by taking its absolute value and searching the `x` vector in forward order, while in SCILIB, a negative `incx` argument results in searching the array `x` in backward order. No VECLIB8 subprograms call any of these subprograms with a negative `incx` argument, so you may safely load SCILIB before VECLIB8 if you need the SCILIB functionality.

Two other subprograms in both VECLIB8 and SCILIB have the same functionality but different numbers of arguments. Subroutines SGEMMS and CGEMMS from the two libraries implement Strassen's method for matrix multiplication, but the SCILIB versions have an extra argument, for working storage, that is not needed in the VECLIB8 versions. Be certain that your calls to these subprograms have 14 arguments if you load SCILIB before VECLIB8.

Differences between LAPACK8 and SCILIB

Two subprograms common to LAPACK8 and SCILIB differ slightly in functionality. Subprograms ICAMAX and ISAMAX in LAPACK8 handle a negative `incx` argument by taking its absolute value and searching the `x` vector in forward order, while in SCILIB, a negative `incx` argument results in searching the array `x` in backward order. No LAPACK8 subprograms call either of these subprograms with a negative `incx` argument, so you may safely load SCILIB before LAPACK8 if you need the SCILIB functionality.

Optimization

LAPACK has been optimized by using a highly efficient implementation of the Basic Linear Algebra Subprograms (BLAS) and the Level 2 and Level 3 BLAS. In addition, certain algorithmic improvements have been made, and several tunable parameters have been adjusted for good execution performance.

Parallel Processing

Parallel processing is available on the CONVEX Exemplar family of supercomputers and on CONVEX C2, C3, and C4 Series computers that have multiple processors. These systems can divide a single computational process into small streams of execution, called *threads*. The result is that you can have more than one processor executing on behalf of the same process.

LAPACK works in both single-processor and parallel-processor environments. At run time, a LAPACK subprogram determines if more than one processor is available and how many are being used. It uses this information to choose automatically between a single-processor or parallel-processor algorithm. Consequently, you can move programs that use LAPACK between single-processor and parallel-processor systems freely, without losing compatibility or the advantages of the architectures.

If you are computing on a CONVEX system with multiple processors, you can realize the performance benefits of parallel processing in two ways. First, you can call any parallelized LAPACK subprogram and let it use parallelism internally. Alternatively, you can call LAPACK subprograms in a parallelized loop or region. To obtain parallelism via the second mechanism, you need to be familiar with the techniques of parallel programming on your computer system.

LAPACK subprograms are reentrant, which means that they may be called several times in parallel to do independent computations without one call interfering with another. You can use this feature to call LAPACK subprograms in a parallelized loop or region. The compiler does not automatically parallelize loops containing a function reference or subroutine call. You can force it to parallelize such a loop by inserting a FORCE_PARALLEL compiler directive before the loop. For example, the following Fortran code makes parallel calls to subprogram SGETRS:

```
CIR CSERIES FORCE_PARALLEL
CIR SPP LOOP_PRIVATE (I)
CIR SPP LOOP_PARALLEL
      DO 10 J=1, N
          CALL SGETRS ('N', N, 1, A, LDA, IPIV, B(1,J), LDB, INFO(J))
10    CONTINUE
```

While optimizing a parallel program, you might want to make parallel calls to a LAPACK subprogram to execute independent operations, but where the call statements are not in a loop. The Fortran compiler does not automatically parallelize code outside a loop, but you can use the BEGIN_TASKS, NEXT_TASK, and END_TASKS compiler directives to tell the compiler to parallelize such code.

Normally, a LAPACK subprogram is called in a parallel loop or region unless the problem size is small. Most LAPACK subprograms are parallelized internally. If a subprogram is called from a parallelized loop or region, the internal parallelism is disabled because the ASAP (automatic self-allocating processors) mechanism does not support nested parallelism.

For more information on compiler directives, including usage cautions and warnings, refer to the *Fortran User's Guide* and the *Fortran Optimization Guide*.

Profiling LAPACK Applications

The CONVEX Performance Analyzer, CXpa, is an interactive tool for CONVEX computer systems that gathers and analyzes program execution timing (profiling) data. CXpa provides the programmer with the means to study the timing behavior of a program for the purposes of optimizing, benchmarking, and debugging. To use the performance analyzer, you must first compile your Fortran program with either the `-pa`, `-pab`, or `-par` compiler option. These options instrument the compiled program so that its performance can be measured at the subprogram level, the loop level, the block level, or the region level.

LAPACK has been instrumented at the subprogram level so that the performance of LAPACK subprograms can be included in the analysis. This instrumentation is nonintrusive, so it is not necessary to use a different version of LAPACK when you desire to profile your program. Also, the CXpa

instrumentation does not interfere with the *prof*, *bprof*, or *gprof* instrumentation in your program. However, you may not profile your program with both *CXpa* and *prof*, *bprof*, or *gprof* at the same time.

Subprogram-level profiling produces summary information about the subprograms that are called during profiled execution of the program. This information includes:

- the number of times each subprogram is called
- the CPU time in each subprogram and the percentage of the program total, either including or excluding the cumulative time in called subprograms
- a dynamic call graph, listing the subprogram calls that take place within a computer program

CXpa is an optional product. For more information about *CXpa*, refer to the *Performance Analyzer (CXpa) User's Guide*, or contact your CONVEX sales representative.

Floating-Point Formats

CONVEX C Series computers operate on data represented in either of two floating-point formats, called *native* format and *IEEE* format. ANSI/IEEE Standard 754 defines IEEE format, and both formats are described in the *Architecture Reference (C Series)*. LAPACK operates in either floating-point format by automatically determining the format the calling program is using, so you need not do anything special to incorporate LAPACK subprograms into programs whether you compile them to use native or IEEE format.

CONVEX Exemplar and other computers with PA-RISC-based architectures operate only on floating-point data in the IEEE format. For further information on CONVEX floating-point formats, refer to the *Fortran User's Guide*.

Roundoff Effects

LAPACK subprograms may use a different arithmetic order of evaluation than other implementations. Different roundoff characteristics may result. Accuracy of results is usually about the same, so using LAPACK should not materially affect the accumulation of roundoff errors in a complete application program. If it does, you should examine the mathematical analysis of the problem, which will likely show that the problem is ill-conditioned. Ill-conditioned means that the small roundoff errors that are inadvertently introduced into any computation are magnified out of proportion to the desired result. Similarly, if results with and without LAPACK differ materially, both sets of answers are probably inaccurate and you should investigate further. If the program correctly applies stable computational algorithms, the problem itself is probably ill-posed.

LAPACK Organization and Naming Convention

Data Types and Precision

LAPACK provides the same range of functionality for both real and complex data. Matching pairs of subprograms, one for real data and one for complex data, are available for most computations, but there are a few exceptions. For example, subprograms for both complex Hermitian and complex symmetric systems of linear equations correspond to the subprograms for real symmetric indefinite systems, because both types of complex systems occur in practical applications. For another example, there is no complex analogue of the subprograms 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 subprograms for real and complex data have been coded to maintain a close correspondence between the two, wherever possible; but in some areas (especially the nonsymmetric eigenproblem), the correspondence is necessarily weaker.

All subprograms in LAPACK are provided in both 32-bit and 64-bit precision versions. All LAPACK8 subprograms use only 64-bit precision.

LAPACK Naming Convention

The name of each LAPACK subprogram is a coded specification of its function, within the limits of standard FORTRAN 77 6-character names. All driver and computational subprograms, as well as most of the auxiliary subprograms, (see "Classes of Subprograms") have names of the form TXXYYYY, although for some subprograms the sixth character is blank.

The first letter, T, shows the predominant data type according to Table 1-1:

Table 1-1: LAPACK Naming Convention—Data Type

T	Data Type	LAPACK	LAPACK8
S	Single Precision	REAL*4	REAL*8
D	Double Precision	REAL*8	—
C	Complex	COMPLEX*8	COMPLEX*16
Z	Double Complex	COMPLEX*16	—

To refer to a LAPACK subprogram generically, without regard to data type, we replace the first letter by '_'. Thus, "_GBSV" refers to any or all of the subprograms SGBSV, CGBSV, DGBSV, and ZGBSV.

The next two letters, **XX**, designate either the form of the matrix or 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, according to Table 1-2.

Table 1-2: LAPACK Naming Convention—Matrix Form

XX	Matrix Form
BD	Bidiagonal
DI	Diagonal
GB	General band
GE	General full
GG	General full generalized
GT	General tridiagonal
HB	Complex Hermitian band
HE	Complex Hermitian
HG	Complex Hermitian generalized
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	Symmetric band
SP	Symmetric, packed storage
ST	Symmetric tridiagonal
SY	Symmetric
TB	Triangular band
TG	Triangular generalized
TP	Triangular, packed storage
TR	Triangular (or in some cases quasi-triangular)
TZ	Trapezoidal
UN	Complex unitary
UP	Complex unitary, packed storage

The last three letters *YYY* designate the computation performed. Their meanings are listed in Table 1-3.

Table 1-3: LAPACK Naming Convention—Computation

YYY	Computation
BAK	Back transformation of eigenvectors after balancing
BAL	Permute and/or balance to isolate eigenvalues
BRD	Reduce to bidiagonal form by orthogonal transformations
CON	Estimate condition number
EBZ	Compute selected eigenvalues by bisection
EDC	Compute eigenvalues and eigenvectors by divide-and-conquer method
EIN	Compute selected eigenvectors by inverse transformations
EQR	Compute eigenvalues and/or Schur Form using the <i>QR</i> algorithm
EQU	Equilibrate a matrix to reduce its condition number
EQZ	Compute eigenvalues and/or Schur Form using the <i>QZ</i> algorithm
ERF	Compute eigenvectors using the Pal-Walker-Kahan variant of the <i>QL</i> or <i>QR</i> algorithm
ES	Simple driver to compute all eigenvalues, Schur Form, and/or Schur vectors
ESX	Simple driver to compute all eigenvalues, Schur Form, and/or Schur vectors
EV	Simple driver to compute all eigenvalues and/or eigenvectors
EVC	Compute eigenvectors from Schur factorization
EVD	Simple driver to compute all eigenvalues and/or eigenvectors
EVX	Expert driver to compute selected eigenvalues and eigenvectors
EXC	Swap adjacent diagonal blocks in a quasi-upper-triangular matrix
GBR	Generate the orthogonal/unitary matrix from <i>_GEBRD</i>
GHR	Generate the orthogonal/unitary matrix from <i>_GEHRD</i>
GLM	Driver to solve generalized linear regression model problem
GLQ	Generate the orthogonal/unitary matrix from <i>_GELQF</i>
GQL	Generate the orthogonal/unitary matrix from <i>_GEQLF</i>
GQR	Generate the orthogonal/unitary matrix from <i>_GEQRF</i>
GRQ	Generate the orthogonal/unitary matrix from <i>_GERQF</i>
GS	Driver to compute generalized eigenvalues, Schur Form, and/or Schur vectors
GST	Reduce a symmetric definite generalized eigenvalue problem to standard form
GTR	Generate the orthogonal/unitary matrix from <i>_XXTRD</i>
GV	Driver to compute generalized eigenvalues and/or generalized eigenvectors
HRD	Reduce to upper Hessenberg form by orthogonal transformations
LQF	Compute an <i>LQ</i> factorization without pivoting
LS	Driver to solve over- or underdetermined linear system using orthogonal factorizations
LSE	Driver to solve linear equality constrained least squares problem
LSS	Driver to solve least squares problem using the singular value decomposition
LSX	Driver to compute minimum-norm solution using a complete orthogonal factorization
MBR	Multiply by the orthogonal/unitary matrix from <i>_GEBRD</i>
MHR	Multiply by the orthogonal/unitary matrix from <i>_GEHRD</i>
MLQ	Multiply by the orthogonal/unitary matrix from <i>_GELQF</i>
MLQ	Multiply by the orthogonal/unitary matrix from <i>_GEQLF</i>
MQR	Multiply by the orthogonal/unitary matrix from <i>_GEQRF</i>
MRQ	Multiply by the orthogonal/unitary matrix from <i>_GERQF</i>
MTR	Multiply by the orthogonal/unitary matrix from <i>_XXTRD</i>
QLF	Compute a <i>QL</i> factorization without pivoting
QPF	Compute a <i>QR</i> factorization with column pivoting
QRF	Compute a <i>QR</i> factorization without pivoting

Table 1-3: LAPACK Naming Convention—Computation (cont.)

YYY	Computation
RFS	Refine initial solution returned by the TRS subprograms
RQF	Compute an RQ factorization without pivoting
SEN	Compute a basis and/or reciprocal condition number of an invariant subspace
SJA	Compute singular value decomposition
SNA	Estimate reciprocal condition numbers of eigenvalue/vector pairs
SQR	Compute singular values and/or singular vectors using the QR algorithm
STF	Compute a split Cholesky factorization
SV	Simple driver to solve a system of linear equations
SVD	Driver to compute the singular value decomposition
SVP	Preprocessing step for computing the singular value decomposition
SVX	Expert driver to solve a system of linear equations
SYL	Solve the Sylvester matrix equation
TRD	Reduce a symmetric matrix to real symmetric tridiagonal form
TRF	Compute a triangular factorization (LU , Cholesky, etc.)
TRI	Compute inverse (based on triangular factorization)
TRS	Solve systems of linear equations (based on triangular factorization)

Classes of Subprograms

LAPACK contains several different classes of subprograms:

- Driver subprograms, each of which solves a complete problem, such as solving a system of linear equations or computing the eigenvalues of a matrix. For some problems, there are both simple and expert driver subprograms. We encourage you to use a driver subprogram if one meets your requirements.
- Computational subprograms, each of which does a distinct computational task, such as an LU factorization or the reduction of a real symmetric matrix to tridiagonal form. Driver subprograms typically call a sequence of computational subprograms, usually with computational and flow-control code sequences interspersed between the calls. Users may want to call computational subprograms directly to perform tasks, or task sequences, that cannot conveniently be performed by the driver subprograms.
- Auxiliary subprograms, which, in turn, can be subclassified as follows:
 - Subprograms to do subtasks of block algorithms—including subprograms that implement unblocked versions of the algorithms; this subclass has subprogram names of the form `_XXYYY` with `YYY` containing the digit “2”.
 - Subprograms to do some commonly-required low-level computations, such as scaling a matrix, computing a matrix norm, or generating an elementary Householder matrix; these subprograms have names of the form `_LAYYY`, where `YYY` designates the specific computation performed.
 - Extensions to the BLAS, such as subprograms for applying complex plane rotations, or matrix-vector operations involving complex symmetric matrices; the names of these subprograms are BLAS-like.

Most auxiliary subprograms are not user-visible, and therefore are not documented in this manual and are not guaranteed to be included in future releases of LAPACK.

LAPACK Contents

The driver subprograms provided in LAPACK are shown in Table 1-4. Table 1-5 identifies the computational subroutines for the solution of systems of linear equations. The computational subroutine names for the solution of least squares problems are given in Table 1-6. Table 1-7 lists the computational subroutines in LAPACK for finding the eigenvalues and eigenvectors of matrices. Table 1-8 shows the names of the LAPACK computational subprograms for the singular value decomposition. Finally, see Table 1-9 for a list of LAPACK auxiliary subprograms.

Following is the key for the table entries in Tables 1-4 to 1-8:

- “√” — Subprogram name begins with S, D, C, and Z.
- “S” — Subprogram name begins with S and D only.
- “C” — Subprogram name begins with C and Z only.
- “-” — There are no subprograms with that XYYYY combination.

Table 1-4: Driver Subprograms

Computation (YYY)	Matrix Form (XX)														
	GB	GE	GG	GT	HB	HE	HP	PB	PO	PP	PT	SB	SP	ST	SY
ES	-	√	-	-	-	-	-	-	-	-	-	-	-	-	-
ESX	-	√	-	-	-	-	-	-	-	-	-	-	-	-	-
EV	-	√	-	-	C	C	C	-	-	-	-	S	S	S	S
EVD	-	-	-	-	C	C	C	-	-	-	-	S	S	S	S
EVX	-	√	-	-	C	C	C	-	-	-	-	S	S	S	S
GLM	-	-	√	-	-	-	-	-	-	-	-	-	-	-	-
GS	-	√	-	-	-	-	-	-	-	-	-	-	-	-	-
GV	-	√	-	-	C	C	C	-	-	-	-	S	S	-	S
LS	-	√	-	-	-	-	-	-	-	-	-	-	-	-	-
LSE	-	-	√	-	-	-	-	-	-	-	-	-	-	-	-
LSS	-	√	-	-	-	-	-	-	-	-	-	-	-	-	-
LSX	-	√	-	-	-	-	-	-	-	-	-	-	-	-	-
SV	√	√	-	√	-	C	C	√	√	√	√	-	√	-	√
SVD	-	√	√	-	-	-	-	-	-	-	-	-	-	-	-
SVX	√	√	-	√	-	C	C	√	√	√	√	-	√	-	√

Table 1-5: Computational Subprograms for Linear Equations

Computation (YYY)	Matrix Form (XX)													
	GB	GE	GT	HE	HP	PB	PO	PP	PT	SP	SY	TB	TP	TR
CON	√	√	√	C	C	√	√	√	√	√	√	√	√	√
EQU	√	√	-	-	-	√	√	√	-	-	-	-	-	-
RFS	√	√	√	C	C	√	√	√	√	√	√	√	√	√
TRF	√	√	√	C	C	√	√	√	√	√	√	-	-	-
TRI	-	√	-	C	C	-	√	√	-	√	√	-	√	√
TRS	√	√	√	C	C	√	√	√	√	√	√	√	√	√

Table 1-6: Computational Subprograms for Least Squares

Computation (YYY)	Matrix Form (XX)				
	GE	GG	OR	TZ	UN
GLQ	-	-	S	-	C
GQL	-	-	S	-	C
GQR	-	-	S	-	C
GRQ	-	-	S	-	C
LQF	√	-	-	-	-
MLQ	-	-	S	-	C
MQL	-	-	S	-	C
MQR	-	-	S	-	C
MRQ	-	-	S	-	C
QLF	√	-	-	-	-
QPF	√	-	-	-	-
QRF	√	√	-	-	-
RQF	√	√	-	√	-

Table 1-7: Computational Subprograms for Eigenvalue Problems

Computation (YYY)	Matrix Form (XX)																	
	GE	GG	HB	HE	HG	HP	HS	OP	OR	PT	SB	SP	ST	SY	TG	TR	UN	UP
BAK	√	√	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
BAL	√	√	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
EBZ	-	-	-	-	-	-	-	-	-	-	-	-	S	-	-	-	-	-
EDC	-	-	-	-	-	-	-	-	-	-	-	-	√	-	-	-	-	-
EIN	-	-	-	-	-	-	√	-	-	-	-	-	√	-	-	-	-	-
EQR	-	-	-	-	-	-	√	-	-	√	-	-	√	-	-	-	-	-
EQZ	-	-	-	-	√	-	-	-	-	-	-	-	-	-	-	-	-	-
ERF	-	-	-	-	-	-	-	-	-	-	-	-	S	-	-	-	-	-
EVC	-	-	-	-	-	-	-	-	-	-	-	-	-	-	√	√	-	-
EXC	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	√	-	-
GHR	-	-	-	-	-	-	-	-	S	-	-	-	-	-	-	-	C	-
GST	-	-	C	C	-	C	-	-	-	-	S	S	-	S	-	-	-	-
GTR	-	-	-	-	-	-	-	S	S	-	-	-	-	-	-	-	C	C
HRD	√	√	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
MHR	-	-	-	-	-	-	-	-	S	-	-	-	-	-	-	-	C	-
MTR	-	-	-	-	-	-	-	S	S	-	-	-	-	-	-	-	C	C
SEN	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	√	-	-
SNA	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	√	-	-
STF	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
SYL	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	√	-	-
TRD	-	-	C	C	-	C	-	-	-	-	S	S	-	S	-	-	-	-

Table 1-8: Computational Subprograms for Singular Value Decomposition

Computation (YYY)	Matrix Form (XX)						
	BD	GB	GE	GG	OR	TG	UN
BRD	-	√	√	-	-	-	-
GBR	-	-	-	-	S	-	C
MBR	-	-	-	-	S	-	C
SJA	-	-	-	-	-	√	-
SQR	√	-	-	-	-	-	-
SVP	-	-	-	√	-	-	-

Most LAPACK auxiliary subprograms are internal implementation details and are not user-visible. In addition, their subprogram names do not follow the naming convention in Tables 1-1 to 1-3. Table 1-9 lists the names and operations or functions performed by those auxiliary subprograms that are user-visible. Following is the key for the first character of the subprogram names in Table 1-9:

- “I” — Subprogram name begins with I only.
- “ ” — Subprogram name begins with S, D, C, and Z.
- “C” — Subprogram name begins with C and Z only.
- “S” — Subprogram name begins with S and D only.
- “X” — Subprogram name begins with X only.

Table 1-9: LAPACK User-Visible Auxiliary Subprograms

Subprogram Name	Operation or Function Performed
ILAENV	Choose Problem-Dependent Parameters
SLAMCH	Choose Machine-Dependent Parameters
_LANGB	Compute a Norm of a General Band Matrix
_LANGE	Compute a Norm of a General Full Matrix
_LANGT	Compute a Norm of a General Tridiagonal Matrix
_LANSB	Compute a Norm of a Symmetric Band Matrix
CLANHB	Compute a Norm of a Hermitian Band Matrix
_LANSP	Compute a Norm of a Symmetric Matrix Stored in Packed Form
CLANHP	Compute a Norm of a Hermitian Matrix Stored in Packed Form
SLANST	Compute a Norm of a Symmetric Tridiagonal Matrix
CLANHT	Compute a Norm of a Hermitian Tridiagonal Matrix
_LANSY	Compute a Norm of a Symmetric Full Matrix
CLANHE	Compute a Norm of a Hermitian Full Matrix
XERBLA	LAPACK Error Handler

Required Data Item Byte Lengths and How to Get Them

Throughout LAPACK, there is a relationship between the data type of a subprogram, designated by the first character of its name, which is denoted by T in Table 1-1, and the byte lengths of its arguments. This relationship, which differs between the LAPACK and the LAPACK8 libraries, is shown in Table 1-10:

Table 1-10: Data Item Byte Length vs. Data Type and Library

T	LAPACK Argument Lengths			LAPACK8 Argument Lengths		
	INTEGER LOGICAL	REAL	COMPLEX	INTEGER LOGICAL	REAL	COMPLEX
S	4	4	†	8	8	†
D	4	8	†	‡	‡	‡
C	4	4	8	8	8	16
Z	4	8	16	‡	‡	‡

† - no user-visible LAPACK S and D subprograms have COMPLEX arguments.

‡ - the D and Z subprograms are not included in LAPACK8.

As mentioned in “Two LAPACK Libraries,” you may want to run a 32-bit precision program with 64 bits of precision. To support changing the precision without changing the code, CONVEX Fortran Compilers provide several compilation options, namely, `-cfc`, `-p8`, and `-pd8`, that affect the size of Fortran data types. By taking care in your use of Fortran data type declarations, you can write a program that will compile with one set of compiler options and run correctly in 32-bit precision with the LAPACK library or compile with another set of compiler options and run correctly in 64-bit precision with LAPACK8. One constraint is that the program must not use any D or Z subprograms from LAPACK, because the D and Z subprograms are not included in LAPACK8. (Note that a program that calls D or Z LAPACK subprograms would not be a 32-bit program.)

Another scenario is that you might be porting a program from a computer with default 8-byte integer and real data items, and you want to use 4-byte integers while continuing to use 8-byte reals. A program change is required, since the original program would be using the S and C LAPACK subprograms, while the CONVEX version would have to use the D and Z subprograms from the LAPACK library, because they are the only ones that combine 4-byte integers with 8-byte reals. The Fortran preprocessor (see the *Fortran User's Guide*) `#define` statement may be an appropriate conversion tool. Or, for example, the `fc` command line options `-fpp -DSGBSV=DGBSV -Dsgbsv=dgbsv` may be used to translate all SGBSV references to DGBSV. You can deal with constants by replacing them with variables or using the Fortran `PARAMETER` statement.

Table 1-11 shows how the lengths of data items depends on their declarations and the compiler options used.

Table 1-11: Data Item Byte Length vs. Declaration and Compiler Option

Fortran Declaration	Fortran Compiler Option			
	none	-cfc	-p8	-pd8
INTEGER	4	8	8	8
INTEGER*4	4	8	4	4
INTEGER*8	8	8	8	8
Integer by default	4	8	8	8
REAL	4	8	8	8
REAL*4	4	8	4	4
REAL*8	8	8	8	8
Real by default	4	8	8	8
DOUBLE PRECISION	8	16	16	8
Double Precision constant	8	16	16	8
COMPLEX	8	16	16	16
COMPLEX*8	8	16	8	8
COMPLEX*16	16	16	16	16
Complex constant	8	16	16	16
DOUBLE COMPLEX	16	16	16	16
Double Complex constant	16	16	16	16
LOGICAL	4	8	8	8
LOGICAL*4	4	8	4	4
LOGICAL*8	8	8	8	8
Logical constant	4	8	8	8

By comparing Tables 1-10 and 1-11, we notice that if the Fortran data types are not given length specifiers (for example, **REAL** is used instead of **REAL*4**) and none of the compiler options, **-cfc**, **-p8**, or **-pd8**, are used, the LAPACK library is type compatible for the I, S, C prefixes and also for D if the type is **DOUBLE PRECISION**. On the other hand, if no length specifiers are used and one of these compiler options is chosen, LAPACK8 is type-compatible for the I, S, and C prefixes. This provides the easiest interchangeability between 32-bit and 64-bit execution.

In cases of mixed data types, you must choose the correct LAPACK library and subprogram carefully. In particular, LAPACK8 accepts no **INTEGER*4** arguments. For more information on Fortran data types and Fortran compiler options refer to a Fortran language reference.

Error Handling

Most documented subprograms have a diagnostic argument **info**, which reports the success or failure of the computation, as follows:

- **info** = 0: Successful exit
- **info** < 0: Invalid value of an argument—computation not completed
- **info** > 0: Failure during the computation

All LAPACK driver and computational subprograms, and some auxiliary subprograms, check that numeric-valued input arguments such as **n** and **lda**, and character-valued option arguments such as **trans** and **uplo**, have permitted values. If the *k*-th argument is invalid, the subprogram sets **info** = $-k$ and calls the error handling subprogram XERBLA.

The standard version of XERBLA writes an error message onto the standard error file. If you are running on a CONVEX C Series system and your main program is in Fortran, a call traceback is also written onto the standard error file. XERBLA then terminates execution with a nonzero exit status. Thus, using the standard version of XERBLA, no LAPACK subprogram would ever return to the calling program with `info < 0`. You may supply a version of XERBLA that alters this action.

The description of each high level subprogram defines the specific error code numbers and the related error conditions when `info ≠ 0`. Always check the output argument `info` after calling an LAPACK subprogram that has `info` as an argument and take appropriate action should the output argument suggest a problem.

ConvexMLIB Man Pages: LAPACK

The *ConvexMLIB Man Pages* contain online documentation that includes information from the *ConvexMLIB User's Guide: LAPACK*. This reference contains an introduction to LAPACK and to each set of subprograms in LAPACK, and reference entries for each subprogram.

This reference is provided for users to easily and efficiently obtain online information on LAPACK. Because of the limited number of fonts supported and the difficulty of presenting mathematical equations in the *man* (1) system, the *ConvexMLIB Man Pages* is not a substitute for the user's guides; the most detailed information on LAPACK will be in the user's guide.

The *ConvexMLIB Man Pages* are installed in the directory `/usr/convex/man/man3` on Exemplar systems, `/usr/man/man3` on C Series systems, and `/usr/cxmaster/hppa/man/man3` on Hewlett-Packard machines. You must have the parent directory in your MANPATH environment variable to access man pages for VECLIB, SCILIB, or LAPACK. If you are using ConvexMLIB on an Exemplar system, for example, you will add `/usr/convex/man` to your MANPATH environment variable. Refer to the *mllib*(3m) master man page for details about the MANPATH variable and the other ConvexMLIB man pages.

To access reference entries, use the shell command

```
man mlib
```

For further explanation and a table of contents of reference entries for LAPACK, refer to the *lapack*(3m) entry by typing

```
man lapack
```

after you have added `/usr/convex/man`, `/usr/man`, or `/usr/cxmaster/hppa/man` to your MANPATH environment variable.

Support Services

CONVEX maintains a staff to provide technical help if you have difficulty. Located in the CONVEX Technical Assistance Center (TAC), these people are the primary link between you and the company, and they stand ready to assist you with any difficulties. Note, though, that LAPACK has been tested extensively and is very reliable. Therefore, before contacting the TAC about a LAPACK problem, follow this procedure to isolate the cause of the trouble and to simplify the job of resolving it:

- Check any error response provided by the subprogram in question. The subprogram descriptions in this manual describe how to check an error response. If the answer is wrong because an error has been detected, correct the cause of the error and run the job again.

- Verify that the subprogram usage in the program matches the subprogram specifications in this manual. Pay special attention to the number of arguments in the **CALL** statement and to the declarations of arrays and integer constants or variables that describe them. If everything is in order, write out all the arguments immediately before and after the **CALL** statement.
- Make sure there really is a problem. For example, if an apparently incorrect answer is being computed, check to see if the answer does satisfy the problem as defined in the program. Also, for problems with more than one answer, LAPACK may produce a different answer or give the answers in a different order than expected. If the problem is ill-conditioned, LAPACK may not be able to compute a reliable answer at all. Again, error messages often suggest the cause of the problem.
- Isolate the problem. If possible, write a small test program that encounters the same difficulty. Perhaps data causing the problem may be written out from the original program and read into the small one. Try to remove the problem area from a large program and concentrate it in a small program. In this way, you eliminate extraneous code from suspicion. If the problem area is large, try to pare it to a manageable size. For example, if a 50-by-50 linear system fails, try to produce a 2-by-2 system that fails in the same way. Clearly, this is not always possible, but the process often leads to insight.

You will frequently discover a usage error and resolve the problem by following the steps above. If the trouble persists, contact the TAC for help. Providing a small test program and expected answers will help the TAC further analyze the problem. To report a software or documentation problem to the TAC, use the *contact* utility. The *contact* utility allows you to submit a problem or suggest an enhancement directly to the TAC from your own system.

For information about *contact*, use the shell command

```
man contact
```

Suggestions

You are encouraged to use the *contact* utility to suggest capabilities you would like included in future releases of LAPACK. Using *contact* helps maintain an orderly record of customer requests and facilitates timely responses. Please submit a *contact* report with a priority of 6. Your suggestion will be routed to the CONVEX Mathematical Software Group for review.

Supplemental Reading

The VECLIB documentation set includes the *ConvexMLIB User's Guide: VECLIB*, the *ConvexMLIB User's Guide: SCILIB*, and the *ConvexMLIB User's Guide: LAPACK*. The following additional documents provide supplemental help.

Dongarra, J.J., J.R. Bunch, C.B. Moler, and G.W. Stewart. *LINPACK Users' Guide*. Philadelphia, PA: SIAM Publications. 1979.

Garbow, B.S., *et al.* "Matrix Eigensystem Routines—EISPACK Guide Extension." *Lecture Notes in Computer Science*, Vol. 51. New York: Springer-Verlag. 1977.

Smith, B.T., *et al.* "Matrix Eigensystem Routines—EISPACK Guide." *Lecture Notes in Computer Science*, Vol. 6, 2nd edition. New York: Springer-Verlag. 1976.

Simple Drivers for Linear Equations

Overview

This chapter explains how to use LAPACK simple drivers to solve systems of linear equations for a variety of types of matrices, including:

- real and complex general full matrices
- real and complex general band matrices
- real symmetric and complex Hermitian positive definite full matrices
- real symmetric and complex Hermitian positive definite band matrices
- real and complex general tridiagonal matrices
- real symmetric and complex Hermitian positive definite tridiagonal matrices
- real and complex symmetric and complex Hermitian indefinite matrices

Chapter Objectives

After reading this chapter you will:

- know how to use the described subprograms
- know when to use the expert drivers for linear equations described in Chapter 3 or the computational subprograms for linear equations, described in Chapter 4

What You Need to Know to Use These Subprograms

LAPACK simple drivers for linear equations are organized so that it is usually necessary to call only one subprogram to solve one or more systems of linear equations. All systems must use the same coefficient matrix, and the different right hand sides must be given all at once. If these conditions do not hold, use the subprograms in Chapter 4.

Simple drivers for linear equations use a somewhat faster test for singularity than do the expert drivers described in Chapter 3. The simple drivers simply look for a pivot element that is zero, while the expert driver test is based on an estimate of the condition number of the coefficient matrix. The condition number test is significantly more reliable, so if you are more concerned about singularity than short run time, use the expert driver subprograms in Chapter 3.

Supplemental Reading

Forsythe, G. and C.B. Moler. *Computer Solution of Linear Algebraic Systems*. Englewood Cliffs, NJ: Prentice-Hall, Inc. 1967.

Golub, G.H. and C.F. Van Loan. *Matrix Computations*, 2nd Edition. Baltimore, MD: The Johns Hopkins University Press. 1989.

Subprogram Descriptions

Solve a General Band Linear System SGBSV, DGBSV, CGBSV, ZGBSV	2-3
Solve a General Full Linear System SGESV, DGESV, CGESV, ZGESV	2-6
Solve a General Tridiagonal Linear System SGTSV, DGTSV, CGTSV, ZGTSV	2-8
Solve a Positive Definite Band Linear System SPBSV, DPBSV, CPBSV, ZPBSV	2-10
Solve a Positive Definite Full Linear System SPOSV, DPOSV, CPOSV, ZPOSV	2-13
Solve a Positive Definite Full Linear System Stored in Packed Form SPPSV, DPPSV, CPPSV, ZPPSV	2-16
Solve a Positive Definite Tridiagonal Linear System SPTSV, DPTSV, CPTSV, ZPTSV	2-19
Solve a Symmetric or Hermitian Linear System Stored in Packed Form SSPSV, DSPSV, CHPSV, CSPSV, ZHPSV, ZSPSV	2-21
Solve a Symmetric or Hermitian Full Linear System SSYSV, DSYSV, CHESV, CSYSV, ZHESV, ZSYSV	2-25

Solve General Band Linear System

SGBSV/DGBSV/CGBSV/ZGBSV

Purpose These subprograms solve a system of linear equations $AX = B$, where A is a band matrix of order n and X and B are n -by- $nrhs$ matrices. A band matrix is a matrix whose nonzero elements all lie near the principal diagonal. Specifically, $a_{ij} = 0$ if $i-j > kl$ or $j-i > ku$ for some integers kl and ku . The smallest such kl and ku for a given matrix are called the lower and upper bandwidths, respectively, and $k = kl+ku+1$ is the total bandwidth.

Tridiagonal matrices are the special case $kl = ku = 1$. They can be handled more efficiently by LAPACK subprograms SGTSV, DGTSV, CGTSV, or ZGTSV. For positive definite band matrices, use SPBSV, DPBSV, CPBSV, or ZPBSV. These subprograms are documented elsewhere in this chapter.

Gaussian elimination with partial pivoting and row interchanges is used to factor A as $A = PLU$, where P is a permutation matrix, L is unit lower triangular 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 $AX = B$.

Matrix Storage Because it is not necessary to store or operate on the zeros outside the band of A , you need only provide the elements within the band of A . Compared to storing the entire matrix, this can save memory if $2kl+ku+1 < n$.

The following example illustrates the storage of general band matrices. Consider the following matrix A of order $n = 9$ and lower and upper bandwidths $kl = 2$ and $ku = 3$, respectively:

11	12	13	14	0	0	0	0	0
21	22	23	24	25	0	0	0	0
31	32	33	34	35	36	0	0	0
0	42	43	44	45	46	47	0	0
0	0	53	54	55	56	57	58	0
0	0	0	64	65	66	67	68	69
0	0	0	0	75	76	77	78	79
0	0	0	0	0	86	87	88	89
0	0	0	0	0	0	97	98	99

When Gaussian elimination is performed on a general band matrix, pivoting introduces nonzero elements above the band. L can be stored with a lower bandwidth of kl , but U requires an upper bandwidth of $kl+ku$. You must, therefore, provide storage for the extra kl diagonals. This is done by presenting the original matrix to the subprogram in an array large enough to satisfy the additional storage requirements. Thus, for the above matrix, A is given in an array ab with at least $2kl+ku+1 = 8$ rows and $n = 9$ columns as follows:

*	*	*	*	*	+	+	+	+
*	*	*	*	+	+	+	+	+
*	*	*	14	25	36	47	58	69
*	*	13	24	35	46	57	68	79
*	12	23	34	45	56	67	78	89
11	22	33	44	55	66	77	88	99
21	32	43	54	65	76	87	98	*
31	42	53	64	75	86	97	*	*

The asterisks in the $(kl+ku)$ -by- $(kl+ku)$ triangle at the upper left corner and in the kl -by- kl triangle at the lower right corner represent elements of **ab** that are not referenced, and the plus signs in the first kl rows indicate elements that may be filled in during the factorization. Thus, if a_{ij} is an element within the band of A , then it is stored in **ab** $(kl+ku+1+i-j, j)$. Therefore, the columns of A are stored in the columns of **ab**, and the diagonals of A are stored in the rows of **ab**, such that the principal diagonal is stored in row $kl+ku+1$ of **ab**.

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```
INTEGER*4 info, kl, ku, ldab, ldb, n, nrhs
INTEGER*4 ipiv(n)
REAL*4    ab(ldab, n), b(ldb, nrhs)
CALL SGBSV (n, kl, ku, nrhs, ab, ldab, ipiv, b, ldb, info)
```

```
INTEGER*4 info, kl, ku, ldab, ldb, n, nrhs
INTEGER*4 ipiv(n)
REAL*8    ab(ldab, n), b(ldb, nrhs)
CALL DGBSV (n, kl, ku, nrhs, ab, ldab, ipiv, b, ldb, info)
```

```
INTEGER*4 info, kl, ku, ldab, ldb, n, nrhs
INTEGER*4 ipiv(n)
COMPLEX*8 ab(ldab, n), b(ldb, nrhs)
CALL CGBSV (n, kl, ku, nrhs, ab, ldab, ipiv, b, ldb, info)
```

```
INTEGER*4 info, kl, ku, ldab, ldb, n, nrhs
INTEGER*4 ipiv(n)
COMPLEX*16 ab(ldab, n), b(ldb, nrhs)
CALL ZGBSV (n, kl, ku, nrhs, ab, ldab, ipiv, b, ldb, info)
```

LAPACK8, available on C Series and Exemplar architectures:

```
INTEGER*8 info, kl, ku, ldab, ldb, n, nrhs
INTEGER*8 ipiv(n)
REAL*8    ab(ldab, n), b(ldb, nrhs)
CALL SGBSV (n, kl, ku, nrhs, ab, ldab, ipiv, b, ldb, info)
```

```
INTEGER*8 info, kl, ku, ldab, ldb, n, nrhs
INTEGER*8 ipiv(n)
COMPLEX*16 ab(ldab, n), b(ldb, nrhs)
CALL CGBSV (n, kl, ku, nrhs, ab, ldab, ipiv, b, ldb, info)
```

Input	n	The number of linear equations, that is, the order of the matrix A . $n \geq 0$.
	kl	The number of subdiagonals within the band of A . $kl \geq 0$.
	ku	The number of superdiagonals within the band of A . $ku \geq 0$.
	nrhs	The number of right hand sides, that is, the number of columns of the matrix B . $nrhs \geq 0$.
	ab	The matrix A in band storage, in rows $kl+1$ to $2kl+ku+1$; rows 1 to kl of the array need not be set. The j -th column of A is stored in the j -th column of the array ab as follows: $ab(kl+ku+1+i-j, j) = A(i, j)$ for $\max(1, j-ku) \leq i \leq \min(n, j+kl)$.

	ldab	The leading dimension of array ab in the calling program unit. $\text{ldab} \geq 2\text{kl} + \text{ku} + 1$.
	b	The n -by- nrhs matrix of right hand side vectors for the system of equations $AX = B$.
	ldb	The leading dimension of array b in the calling program unit. $\text{ldb} \geq \max(1, n)$.
Output	ab	On successful exit, details of the factorization. U is an upper triangular band matrix with $\text{kl} + \text{ku}$ superdiagonals, stored in rows 1 to $\text{kl} + \text{ku} + 1$. The multipliers, L , used during the factorization, are stored in rows $\text{kl} + \text{ku} + 2$ to $2\text{kl} + \text{ku} + 1$.
	ipiv	On successful exit, the pivot indices that define the permutation matrix P ; row i of the matrix was interchanged with row $\text{ipiv}(i)$.
	b	On successful exit, the n -by- nrhs matrix of solution vectors X overwrites the input.
	info	Status response: info = 0: Successful exit. info < 0: If info = $-k$, the k -th argument had an invalid value. info > 0: If info = k , $U(k, k)$ is zero. The factorization has been completed, but the factor U is singular, and the solution has not been computed.

Notes

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

$n < 0$,
 $\text{kl} < 0$,
 $\text{ku} < 0$,
 $\text{nrhs} < 0$,
 $\text{ldab} < 2\text{kl} + \text{ku} + 1$, and
 $\text{ldb} < \max(1, n)$.

Purpose These subprograms solve a system of linear equations $AX = B$, where A is an n -by- n matrix and X and B are n -by- $nrhs$ matrices.

Gaussian elimination with partial pivoting and row interchanges is used to factor A as $A = PLU$, 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 $AX = B$.

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```
INTEGER*4 info, lda, ldb, n, nrhs
INTEGER*4 ipiv(n)
REAL*4    a(lda, n), b(ldb, nrhs)
CALL SGESV (n, nrhs, a, lda, ipiv, b, ldb, info)
```

```
INTEGER*4 info, lda, ldb, n, nrhs
INTEGER*4 ipiv(n)
REAL*8    a(lda, n), b(ldb, nrhs)
CALL DGESV (n, nrhs, a, lda, ipiv, b, ldb, info)
```

```
INTEGER*4 info, lda, ldb, n, nrhs
INTEGER*4 ipiv(n)
COMPLEX*8 a(lda, n), b(ldb, nrhs)
CALL CGESV (n, nrhs, a, lda, ipiv, b, ldb, info)
```

```
INTEGER*4 info, lda, ldb, n, nrhs
INTEGER*4 ipiv(n)
COMPLEX*16 a(lda, n), b(ldb, nrhs)
CALL ZGESV (n, nrhs, a, lda, ipiv, b, ldb, info)
```

LAPACK8, available on C Series and Exemplar architectures:

```
INTEGER*8 info, lda, ldb, n, nrhs
INTEGER*8 ipiv(n)
REAL*8    a(lda, n), b(ldb, nrhs)
CALL SGESV (n, nrhs, a, lda, ipiv, b, ldb, info)
```

```
INTEGER*8 info, lda, ldb, n, nrhs
INTEGER*8 ipiv(n)
COMPLEX*16 a(lda, n), b(ldb, nrhs)
CALL CGESV (n, nrhs, a, lda, ipiv, b, ldb, info)
```

Input

n The number of linear equations, that is, the order of the matrix A . $n \geq 0$.

nrhs The number of right hand sides, that is, the number of columns of the matrix B . $nrhs \geq 0$.

a The n -by- n matrix of coefficients A .

lda The leading dimension of array a in the calling program unit. $lda \geq \max(1, n)$.

b The n -by- $nrhs$ matrix of right hand side vectors for the system of equations $AX = B$.

ldb The leading dimension of array b in the calling program unit. $ldb \geq \max(1, n)$.

Continued

- Output**
- a** On successful exit, the factors L and U from the factorization $A = PLU$; the unit diagonal elements of L are not stored.
 - ipiv** On successful exit, the pivot indices that define the permutation matrix P ; row i of the matrix was interchanged with row **ipiv**(i).
 - b** On successful exit, the **n-by-nrhs** matrix of solution vectors X overwrites the input.
 - info** Status response:
 - info = 0:** Successful exit.
 - info < 0:** If **info** = $-k$, the k -th argument had an invalid value.
 - info > 0:** If **info** = k , $U(k,k)$ is zero. The factorization has been completed, but the factor U is singular, so the solution could not be computed.

Notes If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

n < 0,
nrhs < 0,
lda < max(1,**n**), and
ldb < max(1,**n**).

Purpose These subprograms solve the equation $AX = B$, where A is an n -by- n tridiagonal matrix, by Gaussian elimination with partial pivoting. A tridiagonal matrix $A = (a_{ij})$ is a matrix whose nonzero elements lie only on the principal diagonal ($i = j$), the subdiagonal ($i = j+1$), and the superdiagonal ($i = j-1$) of the matrix.

Note that the equation $A^T X = B$ may be solved by interchanging the order of the arguments **du** and **dl**, where A^T is the transpose of A .

Matrix Storage The following example illustrates the storage of general tridiagonal matrices. Consider the following tridiagonal matrix of order $n = 7$:

11	12	0	0	0	0	0
21	22	23	0	0	0	0
0	32	33	34	0	0	0
0	0	43	44	45	0	0
0	0	0	54	55	56	0
0	0	0	0	65	66	67
0	0	0	0	0	76	77

The subdiagonal is stored in array **dl**, the principal diagonal is stored in array **d**, and the superdiagonal is stored in array **du**, as follows:

i	dl (i)	d (i)	du (i)
1	21	11	12
2	32	22	23
3	43	33	34
4	54	44	45
5	65	55	56
6	76	66	67
7		77	

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```
INTEGER*4 info, ldb, n, nrhs
REAL*4    b(ldb, nrhs), d(n), dl(n-1), du(n-1)
CALL SGTSV (n, nrhs, dl, d, du, b, ldb, info)
```

```
INTEGER*4 info, ldb, n, nrhs
REAL*8    b(ldb, nrhs), d(n), dl(n-1), du(n-1)
CALL DGTSV (n, nrhs, dl, d, du, b, ldb, info)
```

```
INTEGER*4 info, ldb, n, nrhs
COMPLEX*8 b(ldb, nrhs), d(n), dl(n-1), du(n-1)
CALL CGTSV (n, nrhs, dl, d, du, b, ldb, info)
```

```
INTEGER*4 info, ldb, n, nrhs
COMPLEX*16 b(ldb, nrhs), d(n), dl(n-1), du(n-1)
CALL ZGTSV (n, nrhs, dl, d, du, b, ldb, info)
```

LAPACK8, available on C Series and Exemplar architectures:

```
INTEGER*8 info, ldb, n, nrhs
REAL*8    b(ldb, nrhs), d(n), dl(n-1), du(n-1)
CALL SGTSV (n, nrhs, dl, d, du, b, ldb, info)
```

INTEGER*8 info, ldb, n, nrhs
 COMPLEX*16 b(ldb, nrhs), d(n), dl(n-1), du(n-1)
 CALL CGTSV (n, nrhs, dl, d, du, b, ldb, info)

Input	n	The order of the matrix A . $n \geq 0$.
	nrhs	The number of right hand sides, that is, the number of columns of the matrix B . $\text{nrhs} \geq 0$.
	dl	The $n-1$ subdiagonal elements of A .
	d	The diagonal elements of A .
	du	The $n-1$ superdiagonal elements of A .
	b	The n -by- nrhs matrix of right hand side vectors for the system of equations $AX = B$.
	ldb	The leading dimension of array b in the calling program unit. $\text{ldb} \geq \max(1, n)$.
Output	dl	On successful exit, overwritten by the $n-2$ elements of the second superdiagonal of the upper triangular matrix U from the LU factorization of A , in $\text{dl}(1), \dots, \text{dl}(n-2)$.
	d	On successful exit, overwritten by the n diagonal elements of U .
	du	On successful exit, overwritten by the $n-1$ elements of the first superdiagonal of U .
	b	On successful exit, the n -by- nrhs matrix of solution vectors X overwrites the input.
	info	Status response: info = 0: Successful exit. info < 0: If $\text{info} = -k$, the k -th argument had an invalid value. info > 0: If $\text{info} = k$, $U(k, k)$ is zero, and the solution has not been computed. The factorization has not been completed unless $\text{info} = n$.

Notes These subprograms have different functionality and usage than those with the same names in VECLIB. Be sure to load LAPACK before VECLIB if you want these subroutines and use both libraries.

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11) may be replaced with a user-supplied version to change the error procedure. Error conditions are

$n < 0$,
 $\text{nrhs} < 0$, and
 $\text{ldb} < \max(1, n)$.

Purpose These subprograms solve a system of linear equations $AX = B$, where A is an n -by- n real symmetric or complex Hermitian positive definite band matrix and X and B are n -by- $nrhs$ matrices. A real matrix is symmetric if $A = A^T$, its transpose; a complex matrix is Hermitian if $A = A^*$, its conjugate transpose.

A real symmetric matrix A is positive definite if the quadratic form $x^T Ax$ is positive for all nonzero real vectors x ; a complex Hermitian matrix A is positive definite if the quadratic form $x^* Ax$ is positive for all nonzero complex vectors x .

A positive definite band matrix is a positive definite matrix whose nonzero elements all lie fairly near the principal diagonal. Specifically, $a_{ij} = 0$ if $|i-j| > kd$ for some integer kd . The smallest such kd for a given matrix is called the half bandwidth, and $2kd+1$ is called the total bandwidth.

Tridiagonal matrices are the special case $kd = 1$. They can be handled more efficiently by the LAPACK subprograms SPTSV, DPTSV, CPTSV, and ZPTSV.

Cholesky decomposition is used to factor A as $A = U^*U$, if **uplo** = 'U' or 'u', or $A = LL^*$, if **uplo** = 'L' or '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 $AX = B$.

Matrix Storage Because it is not necessary to store or operate on the zeros outside the band of A , and since either triangle of A may be obtained from the other, you need only provide the band within one triangle of A . Compared to storing the entire matrix, this can save memory in two ways: only the elements within the band are stored, and of them, only the upper or the lower triangle.

The following examples illustrate the storage of positive definite band matrices. Consider the following matrix A of order $n = 7$ and half bandwidth $kd = 2$:

11	12	13	0	0	0	0
12	22	23	24	0	0	0
13	23	33	34	35	0	0
0	24	34	44	45	46	0
0	0	35	45	55	56	57
0	0	0	46	56	66	67
0	0	0	0	57	67	77

Upper triangular storage. The upper triangle of A is stored in an array **ab** with at least $kd+1 = 3$ rows and 7 columns as follows:

*	*	13	24	35	46	57
*	12	23	34	45	56	67
11	22	33	44	55	66	77

The asterisks represent elements in the kd -by- kd triangle at the upper left corner of **ab** that are not referenced. Thus, if a_{ij} is an element within the band of the upper triangle of A , it is stored in $ab(kd+1+i-j, j)$. Therefore, the columns of the upper triangle of A are stored in the columns of **ab**, and the diagonals of the upper triangle of A are stored in the rows of **ab**.

Lower triangular storage. The lower triangle of A is stored in the array **ab** as follows:

11	22	33	44	55	66	77
12	23	34	45	56	67	*
13	24	35	46	57	*	*

The asterisks represent elements in the kd -by- kd triangle at the lower right corner of **ab** that are not referenced. Thus, if a_{ij} is an element within the band of the lower triangle of A , it is stored in $ab(1+i-j, j)$. Therefore, the columns of the lower triangle of A are stored in the columns of **ab**, and the diagonals of the lower triangle of A are stored in the rows of **ab**.

Usage

LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```

CHARACTER*1 uplo
INTEGER*4 info, kd, ldab, ldb, n, nrhs
REAL*4 ab(ldab, n), b(ldb, nrhs)
CALL SPBSV (uplo, n, kd, nrhs, ab, ldab, b, ldb, info)

```

```

CHARACTER*1 uplo
INTEGER*4 info, kd, ldab, ldb, n, nrhs
REAL*8 ab(ldab, n), b(ldb, nrhs)
CALL DPBSV (uplo, n, kd, nrhs, ab, ldab, b, ldb, info)

```

```

CHARACTER*1 uplo
INTEGER*4 info, kd, ldab, ldb, n, nrhs
COMPLEX*8 ab(ldab, n), b(ldb, nrhs)
CALL CPBSV (uplo, n, kd, nrhs, ab, ldab, b, ldb, info)

```

```

CHARACTER*1 uplo
INTEGER*4 info, kd, ldab, ldb, n, nrhs
COMPLEX*16 ab(ldab, n), b(ldb, nrhs)
CALL ZPBSV (uplo, n, kd, nrhs, ab, ldab, b, ldb, info)

```

LAPACK8, available on C Series and Exemplar architectures:

```

CHARACTER*1 uplo
INTEGER*8 info, kd, ldab, ldb, n, nrhs
REAL*8 ab(ldab, n), b(ldb, nrhs)
CALL SPBSV (uplo, n, kd, nrhs, ab, ldab, b, ldb, info)

```

```

CHARACTER*1 uplo
INTEGER*8 info, kd, ldab, ldb, n, nrhs
COMPLEX*16 ab(ldab, n), b(ldb, nrhs)
CALL CPBSV (uplo, n, kd, nrhs, ab, ldab, b, ldb, info)

```

Input

uplo Specifies whether the upper or lower triangular part of the symmetric or Hermitian matrix A is stored, as follows:

uplo = 'U' or 'u': The upper triangular part of A is stored.
uplo = 'L' or 'l': The lower triangular part of A is stored.

n The number of linear equations, that is, the order of the matrix A . $n \geq 0$.

kd The number of superdiagonals of the matrix A if **uplo = 'U' or 'u'**, or the number of subdiagonals if **uplo = 'L' or 'l'**. $kd \geq 0$.

	nrhs	The number of right hand sides, that is, the number of columns of the matrix B . $\text{nrhs} \geq 0$.
	ab	The upper or lower triangle of the symmetric or Hermitian band matrix A , stored in the first $\text{kd}+1$ rows of the array. The j -th column of A is stored in the j -th column of the array ab as follows: If uplo = 'U' or 'u', $\text{ab}(\text{kd}+1+i-j, j) = A(i, j)$ for $\max(1, j-\text{kd}) \leq i \leq j$; If uplo = 'L' or 'l', $\text{ab}(1+i-j, j) = A(i, j)$ for $j \leq i \leq \min(n, j+\text{kd})$.
	ldab	The leading dimension of array ab in the calling program unit. $\text{ldab} \geq \text{kd}+1$.
	b	The n -by- nrhs matrix of right hand side vectors for the system of equations $AX = B$.
	ldb	The leading dimension of array b in the calling program unit. $\text{ldb} \geq \max(1, n)$.
Output	ab	On successful exit, the triangular factor U or L from the Cholesky factorization $A = U*U$ or $A = LL^*$ of the band matrix A , in the same storage format as A .
	b	On successful exit, the n -by- nrhs matrix of solution vectors X overwrites the input.
	info	Status response: info = 0: Successful exit. info < 0: If info = $-k$, the k -th argument had an invalid value. info > 0: If info = k , the leading minor of order k of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

Notes If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

uplo ≠ 'L' or 'l' or 'U' or 'u',
n < 0,
kd < 0,
nrhs < 0,
ldab < $\text{kd}+1$, and
ldb < $\max(1, n)$.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the CALL statement may be improved by coding the **uplo** argument as 'Lower' for 'L' or 'Upper' for 'U'.

Solve Positive Definite Linear System**SPOSV/DPOSV/CPOSV/ZPOSV**

Purpose These subprograms solve a system of linear equations $AX = B$, where A is an n -by- n real symmetric or complex Hermitian positive definite matrix and X and B are n -by- $nrhs$ matrices. A real matrix is symmetric if $A = A^T$, its transpose; a complex matrix is Hermitian if $A = A^*$, its conjugate transpose.

A real symmetric matrix A is positive definite if the quadratic form $x^T Ax$ is positive for all nonzero real vectors x ; a complex Hermitian matrix A is positive definite if the quadratic form $x^* Ax$ is positive for all nonzero complex vectors x .

Cholesky decomposition is used to factor A as $A = U^*U$ if **uplo** = 'U' or 'u', or $A = LL^*$ if **uplo** = 'L' or '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 $AX = B$.

Matrix Storage Because either triangle of A may be obtained from the other, you need only provide one triangle of A . You may supply either the upper or the lower triangle of A , in a two-dimensional array large enough to hold the entire array. The other triangle of the array is not referenced.

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```
CHARACTER*1 uplo
INTEGER*4    info, lda, ldb, n, nrhs
REAL*4      a(lda, n), b(ldb, nrhs)
CALL SPOSV (uplo, n, nrhs, a, lda, b, ldb, info)
```

```
CHARACTER*1 uplo
INTEGER*4    info, lda, ldb, n, nrhs
REAL*8      a(lda, n), b(ldb, nrhs)
CALL DPOSV (uplo, n, nrhs, a, lda, b, ldb, info)
```

```
CHARACTER*1 uplo
INTEGER*4    info, lda, ldb, n, nrhs
COMPLEX*8   a(lda, n), b(ldb, nrhs)
CALL CPOSV (uplo, n, nrhs, a, lda, b, ldb, info)
```

```
CHARACTER*1 uplo
INTEGER*4    info, lda, ldb, n, nrhs
COMPLEX*16  a(lda, n), b(ldb, nrhs)
CALL ZPOSV (uplo, n, nrhs, a, lda, b, ldb, info)
```

LAPACK8, available on C Series and Exemplar architectures:

```
CHARACTER*1 uplo
INTEGER*8    info, lda, ldb, n, nrhs
REAL*8      a(lda, n), b(ldb, nrhs)
CALL SPOSV (uplo, n, nrhs, a, lda, b, ldb, info)
```

```
CHARACTER*1 uplo
INTEGER*8    info, lda, ldb, n, nrhs
COMPLEX*16  a(lda, n), b(ldb, nrhs)
CALL CPOSV (uplo, n, nrhs, a, lda, b, ldb, info)
```

Input	uplo	Specifies whether the upper or lower triangular part of the symmetric or Hermitian matrix A is stored, as follows: uplo = 'U' or 'u': The upper triangular part of A is stored. uplo = 'L' or 'l': The lower triangular part of A is stored.
	n	The number of linear equations, that is, the order of the matrix A . $n \geq 0$.
	nrhs	The number of right hand sides, that is, the number of columns of the matrix B . $nrhs \geq 0$.
	a	The upper or lower triangle of the symmetric or Hermitian matrix A . If uplo = 'U' or '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' or '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	The leading dimension of array a in the calling program unit. $lda \geq \max(1,n)$.
	b	The n -by- $nrhs$ matrix of right hand side vectors for the system of equations $AX = B$.
	ldb	The leading dimension of array b in the calling program unit. $ldb \geq \max(1,n)$.
Output	a	On successful exit, if uplo = 'U' or 'u', the upper triangular part of A has been overwritten by U , or if uplo = 'L' or 'l', the lower triangular part of A has been overwritten by L .
	b	On successful exit, the n -by- $nrhs$ matrix of solution vectors X overwrites the input.
	info	Status response: info = 0: Successful exit. info < 0: If info = $-k$, the k -th argument had an invalid value. info > 0: If info = k , the leading minor of order k of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

Notes If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

uplo \neq 'L' or 'l' or 'U' or 'u',
n < 0,
nrhs < 0,
lda < $\max(1,n)$, and
ldb < $\max(1,n)$.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding the **uplo** argument as 'Lower' for 'L' or 'Upper' for 'U'.

Purpose

These subprograms solve a system of linear equations $AX = B$, where A is an n -by- n real symmetric or complex Hermitian positive definite matrix stored in packed form and X and B are n -by- $nrhs$ matrices. A real matrix is symmetric if $A = A^T$, its transpose; a complex matrix is Hermitian if $A = A^*$, its conjugate transpose.

A real symmetric matrix A is positive definite if the quadratic form $x^T Ax$ is positive for all nonzero real vectors x ; a complex Hermitian matrix A is positive definite if the quadratic form $x^* Ax$ is positive for all nonzero complex vectors x .

The Cholesky decomposition is used to factor A as $A = U^*U$ if **uplo** = 'U' or 'u', or $A = LL^*$ if **uplo** = 'L' or '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 $AX = B$.

Matrix Storage

Because either triangle of A may be obtained from the other, you need only provide one triangle of A , either the upper or the lower triangle. Compared to storing the entire matrix, you save memory by supplying that triangle stored column-by-column in packed form in a 1-dimensional array.

The following examples illustrate the packed storage of symmetric or Hermitian matrices.

Upper triangular storage. If the upper triangle of A is

11	12	13	14
	22	23	24
		33	34
			44

then A is packed column-by-column into an array **ap** as follows:

k	1	2	3	4	5	6	7	8	9	10
ap (k)	11	12	22	13	23	33	14	24	34	44

Upper triangular matrix element a_{ij} is stored in array element **ap**($i+j \times (j-1)/2$).

Lower triangular storage. If the lower triangle of A is

11										
21	22									
31	32	33								
41	42	43	44							

then A is packed column-by-column into an array **ap** as follows:

k	1	2	3	4	5	6	7	8	9	10
ap (k)	11	21	31	41	22	32	42	33	43	44

Lower triangular matrix element a_{ij} is stored in array element **ap**($i+(j-1) \times (2n-j)/2$).

Usage

LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```

CHARACTER*1 uplo
INTEGER*4 info, ldb, n, nrhs
REAL*4 ap((n*(n+1))/2), b(ldb, nrhs)
CALL SPPSV (uplo, n, nrhs, ap, b, ldb, info)

```

CHARACTER*1 uplo
INTEGER*4 info, ldb, n, nrhs
REAL*8 ap((n*(n+1))/2), b(ldb, nrhs)
CALL DPPSV (uplo, n, nrhs, ap, b, ldb, info)

CHARACTER*1 uplo
INTEGER*4 info, ldb, n, nrhs
COMPLEX*8 ap((n*(n+1))/2), b(ldb, nrhs)
CALL CPSPV (uplo, n, nrhs, ap, b, ldb, info)

CHARACTER*1 uplo
INTEGER*4 info, ldb, n, nrhs
COMPLEX*16 ap((n*(n+1))/2), b(ldb, nrhs)
CALL ZPPSV (uplo, n, nrhs, ap, b, ldb, info)

LAPACK8, available on C Series and Exemplar architectures:

CHARACTER*1 uplo
INTEGER*8 info, ldb, n, nrhs
REAL*8 ap((n*(n+1))/2), b(ldb, nrhs)
CALL SPPSV (uplo, n, nrhs, ap, b, ldb, info)

CHARACTER*1 uplo
INTEGER*8 info, ldb, n, nrhs
COMPLEX*16 ap((n*(n+1))/2), b(ldb, nrhs)
CALL CPSPV (uplo, n, nrhs, ap, b, ldb, info)

Input	uplo	<p>Specifies whether the upper or lower triangular part of the symmetric or Hermitian matrix A is stored, as follows:</p> <p>uplo = 'U' or 'u': The upper triangular part of A is stored. uplo = 'L' or 'l': The lower triangular part of A is stored.</p>
	n	The number of linear equations, that is, the order of the matrix A . $n \geq 0$.
	nrhs	The number of right hand sides, that is, the number of columns of the matrix B . $nrhs \geq 0$.
	ap	<p>The upper or lower triangle of the symmetric or Hermitian matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array ap as follows:</p> <p>If uplo = 'U' or 'u', $ap(i + (j-1) \times j/2) = A(i, j)$ for $1 \leq i \leq j$; If uplo = 'L' or 'l', $ap(i + (j-1) \times (2n-j)/2) = A(i, j)$ for $j \leq i \leq n$.</p>
	b	The n -by- nrhs matrix of right hand side vectors for the system of equations $AX = B$.
	ldb	The leading dimension of array b in the calling program unit. $ldb \geq \max(1, n)$.
Output	ap	On successful exit, the factor U or L , from the Cholesky factorization $A = U^*U$ or $A = LL^*$, overwrites the input in the same storage format as A .

b On successful exit, the n -by- $nrhs$ matrix of solution vectors X overwrites the input.

info Status response:

info = 0: Successful exit.

info < 0: If **info** = $-k$, the k -th argument had an invalid value.

info > 0: If **info** = k , the leading minor of order k of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

Notes

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

uplo \neq 'L' or 'l' or 'U' or 'u',
n < 0,
nrhs < 0, and
ldb < max(1,**n**).

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding the **uplo** argument as 'Lower' for 'L' or 'Upper' for 'U'.

Solve Positive Definite Tridiagonal Linear System**SPTSV/.../ZPTSV**

Purpose These subprograms solve a system of linear equations $AX = B$, where A is an n -by- n real symmetric or complex Hermitian positive definite tridiagonal matrix, and X and B are n -by- $nrhs$ matrices. A tridiagonal matrix $A = (a_{ij})$ is a matrix whose nonzero elements lie only on the principal diagonal ($i = j$), the subdiagonal ($i = j+1$), and the superdiagonal ($i = j-1$) of the matrix.

A real matrix is symmetric if $A = A^T$, its transpose; a complex matrix is Hermitian if $A = A^*$, its conjugate transpose.

A real symmetric matrix A is positive definite if the quadratic form $x^T Ax$ is positive for all nonzero real vectors x ; a complex Hermitian matrix A is positive definite if the quadratic form $x^* Ax$ is positive for all nonzero complex vectors x .

Matrix Storage The following example illustrates the storage of symmetric or Hermitian tridiagonal matrices. Consider the following symmetric tridiagonal matrix of order $n = 7$:

11	21	0	0	0	0	0
21	22	32	0	0	0	0
0	32	33	43	0	0	0
0	0	43	44	54	0	0
0	0	0	54	55	65	0
0	0	0	0	65	66	76
0	0	0	0	0	76	77

The subdiagonal is stored in array **e** and the principal diagonal is stored in array **d**, as follows:

i	e (i)	d (i)
1	21	11
2	32	22
3	43	33
4	54	44
5	65	55
6	76	66
7		77

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```

INTEGER*4 info, ldb, n, nrhs
REAL*4    b(ldb, nrhs), d(n), e(n-1)
CALL SPTSV (n, nrhs, d, e, b, ldb, info)

```

```

INTEGER*4 info, ldb, n, nrhs
REAL*8    b(ldb, nrhs), d(n), e(n-1)
CALL DPTSV (n, nrhs, d, e, b, ldb, info)

```

```

INTEGER*4 info, ldb, n, nrhs
REAL*4    d(n)
COMPLEX*8 b(ldb, nrhs), e(n-1)
CALL CPTSV (n, nrhs, d, e, b, ldb, info)

```

```

INTEGER*4 info, ldb, n, nrhs
REAL*8    d(n)
COMPLEX*16 b(ldb, nrhs), e(n-1)
CALL ZPTSV (n, nrhs, d, e, b, ldb, info)

```

LAPACK8, available on C Series and Exemplar architectures:

```

INTEGER*8 info, ldb, n, nrhs
REAL*8     b(ldb, nrhs), d(n), e(n-1)
CALL SPTSV (n, nrhs, d, e, b, ldb, info)

```

```

INTEGER*8  info, ldb, n, nrhs
REAL*8     d(n)
COMPLEX*16 b(ldb, nrhs), e(n-1)
CALL CPTSV (n, nrhs, d, e, b, ldb, info)

```

Input	n	The order of the matrix A . $n \geq 0$.
	nrhs	The number of right hand sides, that is, the number of columns of the matrix B . $nrhs \geq 0$.
	d	The n diagonal elements of the tridiagonal matrix A .
	e	The $n-1$ subdiagonal elements of the tridiagonal matrix A .
	b	The n -by- $nrhs$ matrix of right hand side vectors for the system of equations $AX = B$.
	ldb	The leading dimension of array b in the calling program unit. $ldb \geq \max(1, n)$.
Output	d	On successful exit, the n diagonal elements of the diagonal matrix D from the LDL^* factorization of A .
	e	On successful exit, the $n-1$ subdiagonal elements of the unit bidiagonal factor L from the LDL^* factorization of A . e can also be regarded as the superdiagonal of the unit bidiagonal factor U from the U^*DU factorization of A .
	b	On successful exit, the n -by- $nrhs$ matrix of solution vectors X overwrites the input.
	info	Status response: info = 0: Successful exit. info < 0: If info = $-k$, the k -th argument had an invalid value. info > 0: If info = k , the leading minor of order k is not positive definite, and the solution has not been computed. The factorization has not been completed unless info = n .

Notes If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

```

n < 0,
nrhs < 0, and
ldb < max(1, n).

```

Solve Symmetric or Hermitian Packed Linear System**SSPSV/.../ZSPSV**

Purpose These subprograms solve a system of linear equations $AX = B$, where A is an n -by- n real or complex symmetric or complex Hermitian matrix stored in packed form and X and B are n -by- $nrhs$ matrices. If A is known to be positive definite, other subprograms in this chapter, specifically designed for positive definite matrices, will be more efficient than these subprograms.

A matrix is symmetric if $A = A^T$, its transpose; a matrix is Hermitian if $A = A^*$, its conjugate transpose.

The structure of A is indicated by the name of the subprogram used:

SSPSV or DSPSV A is a real symmetric packed matrix.
 CSPSV or ZSPSV A is a complex symmetric packed matrix.
 CHPSV or ZHPSV A is a complex Hermitian packed matrix.

If A is real or complex symmetric, the factorization has the form $A = UDU^T$ or $A = LDL^T$ where U is a product of permutation and unit upper triangular matrices, L is a product of permutation and unit lower triangular matrices, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

If A is complex Hermitian, the factorization has the form $A = UDU^*$ or $A = LDL^*$ where U and L are as above, and D is Hermitian 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 $AX = B$.

Matrix Storage Because either triangle of A may be obtained from the other, you need only provide one triangle of A , either the upper or the lower triangle. Compared to storing the entire matrix, you save memory by supplying that triangle stored column-by-column in packed form in a 1-dimensional array.

The following examples illustrate the packed storage of symmetric or Hermitian matrices.

Upper triangular storage. If the upper triangle of A is

11	12	13	14
	22	23	24
		33	34
			44

then A is packed column-by-column into an array **ap** as follows:

k	1	2	3	4	5	6	7	8	9	10
ap(k)	11	12	22	13	23	33	14	24	34	44

Upper triangular matrix element a_{ij} is stored in array element **ap**($i+j \times (j-1)/2$).

Lower triangular storage. If the lower triangle of A is

```

11
21 22
31 32 33
41 42 43 44

```

then A is packed column-by-column into an array \mathbf{ap} as follows:

k	1	2	3	4	5	6	7	8	9	10
$\mathbf{ap}(k)$	11	21	31	41	22	32	42	33	43	44

Lower triangular matrix element a_{ij} is stored in array element $\mathbf{ap}(i+(j-1)\times(2n-j)/2)$.

Usage

LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```

CHARACTER*1 uplo
INTEGER*4 info, ldb, n, nrhs
INTEGER*4 ipiv(n)
REAL*4 ap((n*(n+1))/2), b(ldb, nrhs)
CALL SSPSV (uplo, n, nrhs, ap, ipiv, b, ldb, info)

```

```

CHARACTER*1 uplo
INTEGER*4 info, ldb, n, nrhs
INTEGER*4 ipiv(n)
REAL*8 ap((n*(n+1))/2), b(ldb, nrhs)
CALL DSPSV (uplo, n, nrhs, ap, ipiv, b, ldb, info)

```

```

CHARACTER*1 uplo
INTEGER*4 info, ldb, n, nrhs
INTEGER*4 ipiv(n)
COMPLEX*8 ap((n*(n+1))/2), b(ldb, nrhs)
CALL CHPSV (uplo, n, nrhs, ap, ipiv, b, ldb, info)

```

```

CHARACTER*1 uplo
INTEGER*4 info, ldb, n, nrhs
INTEGER*4 ipiv(n)
COMPLEX*8 ap((n*(n+1))/2), b(ldb, nrhs)
CALL CSPSV (uplo, n, nrhs, ap, ipiv, b, ldb, info)

```

```

CHARACTER*1 uplo
INTEGER*4 info, ldb, n, nrhs
INTEGER*4 ipiv(n)
COMPLEX*16 ap((n*(n+1))/2), b(ldb, nrhs)
CALL ZHPSV (uplo, n, nrhs, ap, ipiv, b, ldb, info)

```

```

CHARACTER*1 uplo
INTEGER*4 info, ldb, n, nrhs
INTEGER*4 ipiv(n)
COMPLEX*16 ap((n*(n+1))/2), b(ldb, nrhs)
CALL ZSPSV (uplo, n, nrhs, ap, ipiv, b, ldb, info)

```

LAPACK8, available on C Series and Exemplar architectures:

```

CHARACTER*1 uplo
INTEGER*8    info, ldb, n, nrhs
INTEGER*8    ipiv(n)
REAL*8      ap((n*(n+1))/2), b(ldb, nrhs)
CALL SSPSV (uplo, n, nrhs, ap, ipiv, b, ldb, info)

```

```

CHARACTER*1 uplo
INTEGER*8    info, ldb, n, nrhs
INTEGER*8    ipiv(n)
COMPLEX*16  ap((n*(n+1))/2), b(ldb, nrhs)
CALL CHPSV (uplo, n, nrhs, ap, ipiv, b, ldb, info)

```

```

CHARACTER*1 uplo
INTEGER*8    info, ldb, n, nrhs
INTEGER*8    ipiv(n)
COMPLEX*16  ap((n*(n+1))/2), b(ldb, nrhs)
CALL CSPSV (uplo, n, nrhs, ap, ipiv, b, ldb, info)

```

Input	uplo	Specifies whether the upper or lower triangular part of the symmetric or Hermitian matrix A is stored, as follows: uplo = 'U' or 'u': The upper triangular part of A is stored. uplo = 'L' or 'l': The lower triangular part of A is stored.
	n	The number of linear equations, that is, the order of the matrix A . $n \geq 0$.
	nrhs	The number of right hand sides, that is, the number of columns of the matrix B . $nrhs \geq 0$.
	ap	The upper or lower triangle as part of the symmetric matrix A , packed columnwise in a linear array. The j -th column of A is stored in the array ap as follows: If uplo = 'U' or 'u' , $ap(i + (j-1) \times j/2) = A(i, j)$ for $1 \leq i \leq j$; If uplo = 'L' or 'l' , $ap(i + (j-1) \times (2n-j)/2) = A(i, j)$ for $j \leq i \leq n$.
	b	The n -by- $nrhs$ matrix of right hand side vectors for the system of equations $AX = B$.
	ldb	The leading dimension of array b in the calling program unit. $ldb \geq \max(1, n)$.
Output	ap	On successful exit, the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization $A = U^T D U$ or $A = LDL^T$, stored as a packed triangular matrix in the same storage format as A .

- ipiv** On successful exit, details of the interchanges and the block structure of D :
- If $\text{ipiv}(k) > 0$, then rows and columns k and $\text{ipiv}(k)$ were interchanged, and $D(k,k)$ is a 1-by-1 diagonal block.
- If $\text{uplo} = \text{'U'}$ or 'u' and $\text{ipiv}(k) = \text{ipiv}(k-1) < 0$, then rows and columns $k-1$ and $-\text{ipiv}(k)$ were interchanged and $D(k-1:k,k-1:k)$ is a 2-by-2 diagonal block.
- If $\text{uplo} = \text{'L'}$ or 'l' and $\text{ipiv}(k) = \text{ipiv}(k+1) < 0$, then rows and columns $k+1$ and $-\text{ipiv}(k)$ were interchanged and $D(k:k+1,k:k+1)$ is a 2-by-2 diagonal block.
- b** On successful exit, the n -by- nrhs matrix of solution vectors X overwrites the input.
- info** Status response:
- info** = 0: Successful exit.
- info** < 0: If **info** = $-k$, the k -th argument had an invalid value.
- info** > 0: If **info** = k , $D(k,k)$ is zero. The factorization has been completed, but the block diagonal matrix D is singular, so the solution could not be computed.

Notes

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

uplo \neq 'L' or 'l' or 'U' or 'u' ,
n < 0,
nrhs < 0, and
ldb < $\max(1,n)$.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding the **uplo** argument as 'Lower' for 'L' or 'Upper' for 'U' .

Solve Symmetric or Hermitian Linear System**SSYSV/.../ZHESV/ZSYSV**

Purpose These subprograms solve a system of linear equations $AX = B$, where A is an n -by- n real or complex symmetric or complex Hermitian matrix and X and B are n -by- $nrhs$ matrices. If A is known to be positive definite, other subprograms in this chapter, specifically designed for positive definite matrices, will be more efficient than these subprograms.

A matrix is symmetric if $A = A^T$, its transpose; a matrix is Hermitian if $A = A^*$, its conjugate transpose.

The structure of A is indicated by the name of the subprogram used:

SSYSV or DSYSV A is a real symmetric matrix.
 CSYSV or ZSYSV A is a complex symmetric matrix.
 CHESV or ZHESV A is a complex Hermitian matrix.

If A is real or complex symmetric, the factorization has the form $A = UDU^T$ or $A = LDL^T$ where U is a product of permutation and unit upper triangular matrices, L is a product of permutation and unit lower triangular matrices, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

If A is complex Hermitian, the factorization has the form $A = UDU^*$ or $A = LDL^*$ where U and L are as above, and D is Hermitian 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 $AX = B$.

Matrix Storage Because either triangle of A may be obtained from the other, you need only provide one triangle of A . You may supply either the upper or the lower triangle of A , in a two-dimensional array large enough to hold the entire matrix. The other triangle of the array is not referenced.

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```
CHARACTER*1 uplo
INTEGER*4   info, lda, ldb, lwork, n, nrhs
INTEGER*4   ipiv(n)
REAL*4     a(lda, n), b(ldb, nrhs), work(lwork)
CALL SSYSV (uplo, n, nrhs, a, lda, ipiv, b, ldb, work, lwork, info)
```

```
CHARACTER*1 uplo
INTEGER*4   info, lda, ldb, lwork, n, nrhs
INTEGER*4   ipiv(n)
REAL*8     a(lda, n), b(ldb, nrhs), work(lwork)
CALL DSYSV (uplo, n, nrhs, a, lda, ipiv, b, ldb, work, lwork, info)
```

```
CHARACTER*1 uplo
INTEGER*4   info, lda, ldb, lwork, n, nrhs
INTEGER*4   ipiv(n)
COMPLEX*8  a(lda, n), b(ldb, nrhs), work(lwork)
CALL CHESV (uplo, n, nrhs, a, lda, ipiv, b, ldb, work, lwork, info)
```

```
CHARACTER*1 uplo
INTEGER*4   info, lda, ldb, lwork, n, nrhs
INTEGER*4   ipiv(n)
COMPLEX*8  a(lda, n), b(ldb, nrhs), work(lwork)
CALL CSYSV (uplo, n, nrhs, a, lda, ipiv, b, ldb, work, lwork, info)
```

```

CHARACTER*1 uplo
INTEGER*4   info, lda, ldb, lwork, n, nrhs
INTEGER*4   ipiv(n)
COMPLEX*16  a(lda, n), b(ldb, nrhs), work(lwork)
CALL ZHESV (uplo, n, nrhs, a, lda, ipiv, b, ldb, work, lwork, info)

```

```

CHARACTER*1 uplo
INTEGER*4   info, lda, ldb, lwork, n, nrhs
INTEGER*4   ipiv(n)
COMPLEX*16  a(lda, n), b(ldb, nrhs), work(lwork)
CALL ZSYSV (uplo, n, nrhs, a, lda, ipiv, b, ldb, work, lwork, info)

```

LAPACK8, available on C Series and Exemplar architectures:

```

CHARACTER*1 uplo
INTEGER*8   info, lda, ldb, lwork, n, nrhs
INTEGER*8   ipiv(n)
REAL*8      a(lda, n), b(ldb, nrhs), work(lwork)
CALL SSYSV (uplo, n, nrhs, a, lda, ipiv, b, ldb, work, lwork, info)

```

```

CHARACTER*1 uplo
INTEGER*8   info, lda, ldb, lwork, n, nrhs
INTEGER*8   ipiv(n)
COMPLEX*16  a(lda, n), b(ldb, nrhs), work(lwork)
CALL CHESV (uplo, n, nrhs, a, lda, ipiv, b, ldb, work, lwork, info)

```

```

CHARACTER*1 uplo
INTEGER*8   info, lda, ldb, lwork, n, nrhs
INTEGER*8   ipiv(n)
COMPLEX*16  a(lda, n), b(ldb, nrhs), work(lwork)
CALL CSYSV (uplo, n, nrhs, a, lda, ipiv, b, ldb, work, lwork, info)

```

Input **uplo** Specifies whether the upper or lower triangular part of the symmetric or Hermitian matrix *A* is stored, as follows:

```

uplo = 'U' or 'u':    The upper triangular part of A is stored.
uplo = 'L' or 'l':    The lower triangular part of A is stored.

```

n The number of linear equations, that is, the order of the matrix *A*. $n \geq 0$.

nrhs The number of right hand sides, that is, the number of columns of the matrix *B*. $nrhs \geq 0$.

a The upper or lower triangle part of the symmetric matrix *A*.

If **uplo** = 'U' or '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' or '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.

Continued

SSYSV/DSYSV/CHESV/CSYSV/ZHESV/ZSYSV

	lda	The leading dimension of array a in the calling program unit. $lda \geq \max(1,n)$.
	b	The n -by- nrhs matrix of right hand side vectors for the system of equations $AX = B$.
	ldb	The leading dimension of array b in the calling program unit. $ldb \geq \max(1,n)$.
	lwork	The length of array work . $lwork \geq \max(1,n)$. For good performance, lwork must generally be larger. The optimum value of lwork for high performance is returned in work (1).
Working Storage	work	An array used for work space. On successful exit, work (1) contains the optimal work space length lwork for high performance.
Output	a	On successful exit, the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization $A = UDU^*$ or $A = LDL^*$.
	ipiv	On successful exit, details of the interchanges and the block structure of D : If ipiv (<i>k</i>) > 0, then rows and columns <i>k</i> and ipiv (<i>k</i>) were interchanged, and D (<i>k</i> , <i>k</i>) is a 1-by-1 diagonal block. If uplo = 'U' or 'u' and ipiv (<i>k</i>) = ipiv (<i>k</i> -1) < 0, then rows and columns <i>k</i> -1 and - ipiv (<i>k</i>) were interchanged and D (<i>k</i> -1: <i>k</i> , <i>k</i> -1: <i>k</i>) is a 2-by-2 diagonal block. If uplo = 'L' or 'l' and ipiv (<i>k</i>) = ipiv (<i>k</i> +1) < 0, then rows and columns <i>k</i> +1 and - ipiv (<i>k</i>) were interchanged and D (<i>k</i> : <i>k</i> +1, <i>k</i> : <i>k</i> +1) is a 2-by-2 diagonal block.
	b	On successful exit, the n -by- nrhs matrix of solution vectors X overwrites the input.
	info	Status response: info = 0: Successful exit. info < 0: If info = - <i>k</i> , the <i>k</i> -th argument had an invalid value. info > 0: If info = <i>k</i> , D (<i>k</i> , <i>k</i>) is zero. The factorization has been completed, but the block diagonal matrix D is singular, so the solution could not be computed.

Notes

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

uplo ≠ 'L' or 'l' or 'U' or 'u',
n < 0,
nrhs < 0,
lda < $\max(1,n)$,
ldb < $\max(1,n)$, and
lwork < 1.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding the **uplo** argument as 'Lower' for 'L' or 'Upper' for 'U'.

Expert Drivers for Linear Equations

Overview

This chapter explains how to use LAPACK expert driver subprograms to solve systems of linear equations.

This operation is performed for a variety of types of matrices including:

- real and complex general full matrices
- real and complex general band matrices
- real symmetric and complex Hermitian positive definite full matrices
- real symmetric and complex Hermitian positive definite band matrices
- real and complex general tridiagonal matrices
- real symmetric and complex Hermitian positive definite tridiagonal matrices
- real and complex symmetric and complex Hermitian indefinite matrices

Chapter Objectives

After you read this chapter you will:

- understand the role of the condition number in solving linear equations
- know how to compute the inverse of a matrix
- know when not to compute the inverse of a matrix
- know how to use the described subprograms

What You Need to Know to Use These Subprograms

LAPACK expert drivers for linear equations are organized so that it is usually necessary to call only one subprogram to solve one or more systems of linear equations. All systems must use the same coefficient matrix, and the different right hand sides must be given all at once. If these conditions do not hold, use the subprograms in Chapter 4.

The expert drivers for linear equations use a significantly more reliable test for singularity than do the simple drivers described in Chapter 2. The expert driver test is based on an estimate of the condition number of the coefficient matrix, while the simple drivers merely look for a pivot element that is zero. The condition number test is slower, so if you are more concerned about run time than singularity, use the simple driver subprograms in Chapter 2.

Condition Number

The standard method for solving a linear system $Ax = b$ is to use Gaussian elimination, usually with some pivoting strategy, to factor a permuted A , for example $PA = LU$, and then to solve the two triangular systems $Ly = Pb$ and $Ux = y$.

Under reasonable assumptions on the floating-point arithmetic and with reasonable assumptions about the matrix A , useful bounds on the errors in the factoring and solution phases can be proven. Ignoring the permutation for now (although some pivoting strategy is necessary to make the bounds valid, unless the matrix has some special property such as positive definiteness) and letting the primed symbols represent computed quantities, some classical error bounds are:

- The computed factorization is the exact factorization of a slightly perturbed A , that is,

$$\|A - L'U'\|_{\infty} \leq \varepsilon p_2(n) r(n) \|A\|_{\infty}$$

- The computed solution x' nearly satisfies the specified equations, meaning that the residual is small, that is,

$$\|b - Ax'\|_{\infty} \leq \varepsilon p_3(n) r(n) \|A\|_{\infty} \|x'\|_{\infty}$$

- A *backward* error bound: the computed solution x' is an exact solution to a perturbed problem $(A + \delta A)x' = b$, where δA is the perturbation in A and

$$\|\delta A\|_{\infty} \leq \varepsilon p_3(n) r(n) \|A\|_{\infty}$$

- A *forward* error bound: the error in x' itself is small if A is not too badly conditioned, that is,

$$\|x' - x\|_{\infty} \leq \varepsilon p_3(n) r(n) \kappa(A) \|x'\|_{\infty}$$

In the above, $\|\cdot\|_{\infty}$ represents the ∞ vector norm and its subordinate matrix norm, ε is the machine precision (the distance from 1.0 to the next greater floating-point number), p_2 and p_3 are polynomials of degree two and three, respectively, with moderate lead coefficients, $r(n)$ is the maximal growth factor of elements of U in the factorization process, and $\kappa(A)$ is the condition number of A , defined as $\|A\|_{\infty} \|A^{-1}\|_{\infty}$.

The rigorous bounds on $r(n)$ for partial pivoting are pessimistic: $r(n) = O(2^n)$. Such growth in $r(n)$ is never seen in practice, however. By the "consensus of the numerical analysis community," $r(n)$ safely can be replaced by a modestly growing function such as $0.15n$ or even by a constant such as 20. Applying these bounds to a particular problem requires a combination of theoretical knowledge and experience with the problem.

The condition number, $\kappa(A)$, appears frequently in error analysis. Large condition numbers are associated with numerical instability, and infinite or overflowing condition numbers are usually taken to suggest *numerical singularity*. The LAPACK expert drivers for linear equations return an estimate of the condition number. Since $1 < \kappa(A) \leq \infty$, it is more convenient to compute the reciprocal condition number, $1/\kappa(A)$, than $\kappa(A)$ itself. Roughly speaking, the reciprocal condition number has the interpretation that if $1/\kappa(A)$ is about 10^{-d} , elements of x' can be expected to have about d fewer significant digits of accuracy than the elements of A or b . Consequently, if uncertainty in the elements of the coefficient matrix and right hand side exceeds $1/\kappa(A)$, or if $1/\kappa(A)$ is on the order of ε , then x' may have no significant digits at all.

Equilibration

Transforming the problem in an attempt to reduce the condition number is a common technique. The expert drivers offer the option of transforming the problem using an operation called *equilibration*. The basic idea is straightforward: letting R and C denote diagonal matrices (standing for “row” and “column” scaling), the problem is transformed from $Ax = b$ to $(RAC)y = Rb$, where $Cy = x$. The goal is to choose R and C to make $\kappa(RAC)$ small, and, in turn, to put an error bound on $\|y' - y\|$ that uses a smaller condition number. This technique often works well in practice. The theoretical justification for this procedure is not air-tight, however, and there are cases where equilibration can lead to larger errors. The following points should be kept in mind:

- The equilibration is not guaranteed to reduce the condition number. In practice, $\kappa(RAC)$ will generally be less than $\kappa(A)$, but that is a heuristically plausible statement that is generally corroborated by experience, not a theorem.
- Reducing the condition number does not guarantee a reduced error bound. It is possible that the maximal growth factor $r(n)$ for matrix RAC could be larger than that for the matrix A .
- Reducing the error bound does not guarantee a reduced error. For example, even if $error_1 \leq bound_1$, $error_2 \leq bound_2$, and $bound_1 < bound_2$, it can still happen that $error_1 > error_2$.
- The error bound for the transformed system is on the y vector. Equivalently, the error bound on the error in x is in a different norm, the C -norm defined by $\|z\|_C = \|C^{-1}z\|$. The appropriateness of the C -norm to the application at hand needs to be considered.

These points in no way deny the usefulness of equilibration. Used with judgment, equilibration is an effective technique that can improve accuracy and broaden the class of solvable problems. But it should not be viewed as a black box guaranteed to eliminate all problems stemming from machine arithmetic and numerical instability.

Iterative Refinement

The classical bounds are unsatisfying in the sense that nothing is known about the relative errors in the individual components of, say, the residual. That is, the error bounds are bounds for the relative errors in vector and matrix norms rather than bounds for the relative error of individual elements. Given knowledge about the structure of A , it might be possible to place stricter bounds on some elements of the residual than on other elements. Except in pathological cases, the method of iterative refinement used by the expert drivers attains such componentwise bounds and gives estimates for the bounds.

Iterative refinement is motivated by an optimistic but naive idea: if x' is the computed solution and $x' + \delta x$ is the exact solution, then $A(x' + \delta x) = b$, so $A\delta x = b - Ax'$, the residual. Thus, we can compute a correction term for the cost of a matrix-vector multiplication and two triangular solves. Unless the residual is computed in double precision, a cursory analysis leads to pessimism about reducing the error in x' . Somewhat surprisingly, though, recent results have shown that iterative refinement, even without the higher precision residual calculation, can improve the computed solution in certain senses described below.

If M is a matrix or vector, let $|M|$ denote the matrix or vector whose elements are the absolute values of the elements of M . The i -th component of the residual $b - Ax'$ is made small compared to the i -th component of $|A| |x| + |b|$, that is, compared to a quantity depending only on the i -th equation. The computed solution is the exact solution of a perturbed system $(A + \delta A)x' = b + \delta b$ where the elements of δA and δb are small compared to their corresponding elements in A and b . That is the solution is *backward stable in a componentwise relative sense*. Although the componentwise relative error in the computed solution cannot be bounded, the standard condition number $\|A\| \|A^{-1}\|$ is replaced by a *componentwise condition number*, $\|A^{-1}(|A| |x'| + |b|)\|$, that depends on b and x' and takes more advantage of any

special structure A and A^{-1} may have. Furthermore, although some terms in the error bounds may not be known (for example $\|A^{-1}\|$), the expert drivers estimate various condition numbers and compute *a posteriori* estimates for the backward and forward error.

The overall situation with equilibration, iterative refinement, and condition number estimation is complicated. See the LAPACK references and the vast literature on error analysis for more details. See (Forsythe and Moler) for an early and easily accessible account of the basic principles. A more recent and more complete account, together with an extensive bibliography, can be found in (Golub and Van Loan). But it should be emphasized that:

- These bounds vary depending on the matrix type. The literature for the particular matrix type should be consulted.
- These bounds depend on “reasonable assumptions” about the floating point arithmetic.
- Some of the error bounds depend on reasonable hypotheses that seldom appear in bold type, such as $n \epsilon < 0.05$ or $n \epsilon \kappa(A) < 1$.
- There are many other potential sources of error in addition to the LAPACK subroutines, such as experimental error, errors in the mathematical model, and truncation errors when the problem is stored in floating-point representation.

Matrix Inversion

Subprograms for computing the inverse of a matrix are provided, although it is almost never necessary to compute one. While papers and reference books extensively use the notation “ $A^{-1}b$ ” to mean “the solution x of the system of linear equations $Ax = b$,” it is more efficient and accurate to compute the solution directly than to invert the coefficient matrix and multiply the inverse times the right hand side vector. This is true even if there are many systems of equations, all using the same coefficient matrix; the matrix may be factored once and the systems may be solved from the factors usually just as efficiently—and more accurately—than by matrix multiplication by the inverse.

Supplemental Reading

- Anderson, E. *et al.* *LAPACK Users' Guide*, Second Edition. Philadelphia, PA: Society for Industrial and Applied Mathematics. 1995.
- Forsythe, G. and C.B. Moler. *Computer Solution of Linear Algebraic Systems*. Englewood Cliffs, NJ: Prentice-Hall, Inc. 1967.
- Golub, G.H. and C.F. Van Loan. *Matrix Computations*, 2nd Edition. Baltimore, MD: The Johns Hopkins University Press. 1989.

Subprogram Descriptions

Solve a General Band Linear System SGBSVX, DGBSVX, CGBSVX, ZGBSVX	3-6
Solve a General Full Linear System SGESVX, DGESVX, CGESVX, ZGESVX	3-14
Solve a General Tridiagonal Linear System SGTSVX, DGTSVX, CGTSVX, ZGTSVX	3-21
Solve a Positive Definite Band Linear System SPBSVX, DPBSVX, CPBSVX, ZPBSVX	3-26
Solve a Positive Definite Full Linear System SPOSVX, DPOSVX, CPOSVX, ZPOSVX	3-33
Solve a Positive Definite Full Linear System Stored in Packed Form SPPSVX, DPPSVX, CPPSVX, ZPPSVX	3-39
Solve a Positive Definite Tridiagonal Linear System SPTSVX, DPTSVX, CPTSVX, ZPTSVX	3-45
Solve a Symmetric or Hermitian Linear System Stored in Packed Form SSPSVX, DSPSVX, CHPSVX, CSPSVX, ZHPSVX, ZSPSVX	3-49
Solve a Symmetric or Hermitian Full Linear System SSYSVX, DSYSVX, CHESVX, CSYSVX, ZHESVX, ZSYSVX	3-55

Purpose These subprograms solve a system of linear equations $AX = B$, where A is a band matrix of order n with kl subdiagonals and ku superdiagonals, and X and B are n -by- $nrhs$ matrices. You may supply either a new matrix A or an A that was used in a previous call, together with its previously computed scaling matrices, if any, and its L and U factors.

These subroutines perform the following:

1. If you provide a new A matrix and request equilibration (**fact** = 'E' or 'e'), A is analyzed to determine whether or not to do row equilibration, and, independently, whether or not to do column equilibration. If both equilibrations are done, real diagonal scaling matrices, R and C , are computed to equilibrate the system. If row equilibration is not done, R is the identity; if column equilibration is not done, C is the identity. Output argument **equed** indicates which equilibrations were done.

If you supply a previously factored A matrix (**fact** = 'F' or 'f') and it was equilibrated (**equed** = 'R' or 'r' or 'C' or 'c' or 'B' or 'b'), then the previously computed scaling matrices will be used in the solution phase.

In either case, if equilibration is done, the system actually solved depends upon the **trans** argument as follows:

If **trans** = 'N' or 'n', equilibrate the system as $(RAC)(C^{-1}X) = RB$.

If **trans** = 'T' or 't', equilibrate the system as $(RAC)^T(R^{-1}X) = CB$.

If **trans** = 'C' or 'c', equilibrate the system as $(RAC)*(R^{-1}X) = CB$.

If equilibration is done, A is overwritten by RAC .

If equilibration is done, or if A was factored in a previous call where equilibration was done, then B may be overwritten. Specifically, if **trans** = 'N' or 'n' and **equed** is 'R' or 'r' or 'B' or 'b', then B is overwritten by RB . If **trans** = 'T' or 't' or 'C' or 'c' and **equed** is 'C' or 'c' or 'B' or 'b', then B is overwritten by CB .

2. If **fact** = 'N' or 'n' or 'E' or 'e', A is copied from array **ab** to array **afb** (after equilibration, if performed), where it is factored as $A = PLU$, where P is a permutation matrix, L is a unit lower triangular matrix with kl subdiagonals, and U is upper triangular with as many as $kl+ku$ superdiagonals due to fill-in.
3. 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 epsilon, steps 4 through 6 are skipped.
4. The system of equations $AX = B$ is solved for X using the factored form of A .
5. Iterative refinement is applied to improve the computed solution vectors and calculate error bounds and backward error estimates.
6. If equilibration was used before factoring the A matrix, X is scaled, if necessary, so as to solve the original system. Specifically, if **trans** = 'N' or 'n' and the final value of **equed** is 'C' or 'c' or 'B' or 'b', then X is premultiplied by C . If **trans** = 'T' or 't' or 'C' or 'c' and the final value of **equed** is 'R' or 'r' or 'B' or 'b', then X is premultiplied by R .

Matrix Storage Because it is not necessary to store or operate on the zeros outside the band of A , you need only provide the elements within the band of A . Compared to the expert driver for general full matrices, this expert driver can save memory if $3kl/2+ku+1 < n$.

The following example illustrates the storage of general band matrices. Consider the following matrix A of order $n=9$ and lower and upper bandwidths $kl=2$ and $ku=3$, respectively:

11	12	13	14	0	0	0	0	0
21	22	23	24	25	0	0	0	0
31	32	33	34	35	36	0	0	0
0	42	43	44	45	46	47	0	0
0	0	53	54	55	56	57	58	0
0	0	0	64	65	66	67	68	69
0	0	0	0	75	76	77	78	79
0	0	0	0	0	86	87	88	89
0	0	0	0	0	0	97	98	99

A is given in an array \mathbf{ab} with at least $kl+ku+1=6$ rows and $n=9$ columns as follows:

*	*	*	14	25	36	47	58	69
*	*	13	24	35	46	57	68	79
*	12	23	34	45	56	67	78	89
11	22	33	44	55	66	77	88	99
21	32	43	54	65	76	87	98	*
31	42	53	64	75	86	97	*	*

The asterisks in the ku -by- ku triangle at the upper left corner and in the kl -by- kl triangle at the lower right corner represent elements of \mathbf{ab} that are not referenced. Thus, if a_{ij} is an element within the band of A , then it is stored in $\mathbf{ab}(ku+1+i-j, j)$. Therefore, the columns of A are stored in the columns of \mathbf{ab} , and the diagonals of A are stored in the rows of \mathbf{ab} , such that the principal diagonal is stored in row $ku+1$ of \mathbf{ab} .

Note that this storage format omits the first kl rows reserved for fill-in in the general band storage for `_GBSV` and `_GBTRF`.

Usage

LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

CHARACTER*1 equed, fact, trans
INTEGER*4 info, kl, ku, ldab, ldafb, ldb, ldx, n, nrhs
REAL*4 rcond
INTEGER*4 ipiv(n), iwork(n)
REAL*4 ab(ldab, n), afb(ldafb, n), b(ldb, nrhs),
 berr(nrhs), c(n), ferr(nrhs), r(n),
 work(3*n), x(ldx, nrhs)
CALL 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*1 equed, fact, trans
INTEGER*4 info, kl, ku, ldab, ldafb, ldb, ldx, n, nrhs
REAL*8 rcond
INTEGER*4 ipiv(n), iwork(n)
REAL*8 ab(ldab, n), afb(ldafb, n), b(ldb, nrhs),
 berr(nrhs), c(n), ferr(nrhs), r(n),
 work(3*n), x(ldx, nrhs)
CALL DGBSVX (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*1 equed, fact, trans
INTEGER*4 info, kl, ku, ldab, ldafb, ldb, ldx, n, nrhs
REAL*4 rcond
INTEGER*4 ipiv(n)
REAL*4 berr(nrhs), c(n), ferr(nrhs), r(n), rwork(n)
COMPLEX*8 ab(ldab, n), afb(ldafb, n), b(ldb, nrhs),
 work(2*n), x(ldx, nrhs)
CALL 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*1 equed, fact, trans
INTEGER*4 info, kl, ku, ldab, ldafb, ldb, ldx, n, nrhs
REAL*8 rcond
INTEGER*4 ipiv(n)
REAL*8 berr(nrhs), c(n), ferr(nrhs), r(n), rwork(n)
COMPLEX*16 ab(ldab, n), afb(ldafb, n), b(ldb, nrhs),
 work(2*n), x(ldx, nrhs)
CALL ZGBSVX (fact, trans, n, kl, ku, nrhs, ab, ldab, afb, ldafb,
 ipiv, equed, r, c, b, ldb, x, ldx, rcond, ferr, berr,
 work, rwork, info)

LAPACK8, available on C Series and Exemplar architectures:

CHARACTER*1 equed, fact, trans
INTEGER*8 info, kl, ku, ldab, ldafb, ldb, ldx, n, nrhs
REAL*8 rcond
INTEGER*8 ipiv(n), iwork(n)
REAL*8 ab(ldab, n), afb(ldafb, n), b(ldb, nrhs),
 berr(nrhs), c(n), ferr(nrhs), r(n),
 work(3*n), x(ldx, nrhs)
CALL 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*1 equed, fact, trans
INTEGER*8 info, kl, ku, ldab, ldafb, ldb, ldx, n, nrhs
REAL*8 rcond
INTEGER*8 ipiv(n)
REAL*8 berr(nrhs), c(n), ferr(nrhs), r(n), rwork(n)
COMPLEX*16 ab(ldab, n), afb(ldafb, n), b(ldb, nrhs),
work(2*n), x(ldx, nrhs)
CALL 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)

Input

fact 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, as follows:

fact = 'F' or 'f': **afb** and **ipiv** contain the factored form of A . If **equed = 'R'** or 'r' or 'C' or 'c' or 'B' or 'b', A was equilibrated before factoring and the scaling matrices are provided in one or both of **r** and **c**.

fact = 'N' or 'n': The matrix A is copied to **afb** and factored.

fact = 'E' or 'e': The matrix A is equilibrated, if necessary, then copied to **afb** and factored.

trans Specifies the form of the system of equations, as follows:

trans = 'N' or 'n': Solve $AX = B$.

trans = 'T' or 't': Solve $A^T X = B$.

trans = 'C' or 'c': Solve $A * X = B$.

n The number of linear equations, that is, the order of the matrix A . $n \geq 0$.

kl The number of subdiagonals within the band of A . $kl \geq 0$.

ku The number of superdiagonals within the band of A . $ku \geq 0$.

nrhs The number of right hand sides, that is, the number of columns of the matrix B . $nrhs \geq 0$.

ab The matrix A in band storage, in the first $kl+ku+1$ rows. The j -th column of A is stored in the j -th column of the array **ab** as follows:

$$ab(ku+1+i-j, j) = A(i, j) \text{ for } \max(1, j-ku) \leq i \leq \min(n, j+kl)$$

ldab The leading dimension of array **ab** in the calling program unit. $ldab \geq kl+ku+1$.

afb If **fact = 'F' or 'f'**, the details of the LU factorization of the band matrix A , as computed by a previous call. U , an upper triangular band matrix with $kl+ku$ superdiagonals, is stored in the first $kl+ku+1$ rows. The multipliers, L , used during the factorization, are stored in rows $kl+ku+2$ to $2kl+ku+1$.

Not used as input if **fact = 'N' or 'n' or 'E' or 'e'**.

	ldafb	The leading dimension of array afb in the calling program unit. $\text{ldafb} \geq 2kl+ku+1$.
	ipiv	If fact = 'F' or 'f', the pivot indices from the factorization $A = PLU$ as computed by a previous call; row i of the matrix was interchanged with row ipiv (i). Not used as input if fact = 'N' or 'n' or 'E' or 'e'.
	equed	If fact = 'F' or 'f', the type of equilibration, if any, that was done before factoring A on a previous call, as follows: equed = 'N' or 'n': No equilibration was done. equed = 'R' or 'r': Only row equilibration was done. equed = 'C' or 'c': Only column equilibration was done. equed = 'B' or 'b': Both row and column equilibration were done. Not used as input if fact = 'N' or 'n' or 'E' or 'e'.
	r	The diagonal elements of the diagonal row scaling matrix R if the matrix A was factored during a previous call (fact = 'F' or 'f'), and if row equilibration was done during that call (equed = 'R' or 'r' or 'B' or 'b'). $r(i) > 0, i = 1, 2, \dots, n$. Otherwise, not used as input.
	c	The diagonal elements of the diagonal column scaling matrix C if the matrix A was factored during a previous call (fact = 'F' or 'f'), and if column equilibration was done during that call (equed = 'C' or 'c' or 'B' or 'b'). $c(i) > 0, i = 1, 2, \dots, n$. Otherwise, not used as input.
	b	The n -by- nrhs matrix of right hand side vectors for the system of linear equations $AX = B$.
	ldb	The leading dimension of array b in the calling program unit. $\text{ldb} \geq \max(1, n)$.
	ldx	The leading dimension of array x in the calling program unit. $\text{ldx} \geq \max(1, n)$.
Working Storage	work,	Arrays used for work space.
	iwork,	On successful exit from SGBSVX or DGBSVX, work (1) contains the reciprocal growth factor, $\ A\ _{\Delta} / \ U\ _{\Delta} = (\max_{ij} a_{ij}) / (\max_{ij} u_{ij})$.
	rwork	On successful exit from CGBSVX or ZGBSVX, rwork (1) contains the reciprocal growth factor, $\ A\ _{\Delta} / \ U\ _{\Delta} = (\max_{ij} a_{ij}) / (\max_{ij} u_{ij})$.
		If $\ A\ _{\Delta} / \ U\ _{\Delta} \ll 1$, then the stability of the LU factorization of the (equilibrated) A matrix could be poor. This also means that the solution vector X , reciprocal condition number estimate rcond , and forward error bound ferr could be unreliable. If the factorization fails with $0 < \text{info} \leq n$, then work (1) or rwork (1) contains the reciprocal growth factor for the leading info columns of A .

Output	ab	<p>If fact = 'E' or 'e' and equed = 'R', then <i>A</i> was overwritten by <i>RA</i>.</p> <p>If fact = 'E' or 'e' and equed = 'C', then <i>A</i> was overwritten by <i>AC</i>.</p> <p>If fact = 'E' or 'e' and equed = 'B', then <i>A</i> was overwritten by <i>RAC</i>.</p> <p>Not used as output if fact = 'F' or 'f' or 'N' or 'n', or if fact = 'E' or 'e' and equed = 'N' on exit.</p>
	afb	<p>If fact = 'N' or 'n' or 'E' or 'e', then afb returns details of the <i>LU</i> factorization of <i>A</i>, after equilibration, if performed. <i>U</i>, an upper triangular band matrix with kl+ku superdiagonals, is stored in the first kl+ku+1 rows. The multipliers, <i>L</i>, used during the factorization, are stored in rows kl+ku+2 to 2kl+ku+1.</p> <p>Not used as output if fact = 'F' or 'f'.</p>
	ipiv	<p>If fact = 'N' or 'n', then ipiv contains the pivot indices from the factorization $A = PLU$ of the original matrix <i>A</i>.</p> <p>If fact = 'E' or 'e', then ipiv contains the pivot indices from the factorization $A = PLU$ of the equilibrated matrix <i>A</i>.</p> <p>Not used as output if fact = 'F' or 'f'.</p>
	equed	<p>If fact = 'E' or 'e', specifies the form of equilibration that was done, as follows:</p> <p>equed = 'N': No equilibration.</p> <p>equed = 'R': Row equilibration; <i>A</i> was premultiplied by <i>R</i>.</p> <p>equed = 'C': Column equilibration; <i>A</i> was postmultiplied by <i>C</i>.</p> <p>equed = 'B': Both row and column equilibration; <i>A</i> was replaced with <i>RAC</i>.</p> <p>Not used as output if fact = 'F' or 'f' or 'N' or 'n'.</p>
	r	<p>If fact = 'E' or 'e' and equed = 'R' or 'B' on exit, the diagonal elements of the diagonal row scaling matrix <i>R</i>.</p> <p>Destroyed if fact = 'E' or 'e' and equed = 'N' or 'C' on exit.</p> <p>Not used as output if fact = 'F' or 'f' or 'N' or 'n'.</p>
	c	<p>If fact = 'E' or 'e' and equed = 'C' or 'B' on exit, the diagonal elements of the diagonal column scaling matrix <i>C</i>.</p> <p>Destroyed if fact = 'E' or 'e' and equed = 'N' or 'R' on exit.</p> <p>Not used as output if fact = 'F' or 'f' or 'N' or 'n'.</p>
	b	<p>If equed = 'N' or 'n', b is not modified.</p> <p>If trans = 'N' or 'n' and equed = 'R' or 'r' or 'B' or 'b', <i>B</i> is overwritten by <i>RB</i>.</p> <p>If trans = 'T' or 't' or 'C' or 'c' and equed = 'C' or 'c' or 'B' or 'b', <i>B</i> was overwritten by <i>CB</i>.</p>

x On successful exit, the solution vectors X to the original system of equations $AX = B$.

rcond On successful exit, the estimate of the reciprocal condition number of the matrix A , after equilibration, if performed. If **rcond** is small enough so that the logical expression

$$1.0 + \text{rcond} \cdot \text{EQ. 1.0}$$

is true, then A can be regarded as singular to working precision. This condition is indicated by a return code of **info** > 0 , and the solution and error bounds are not computed.

ferr On successful exit, estimated forward error bounds for each solution vector. Let x_{true} and x represent column j of the true and computed solutions, respectively. Then **ferr**(j) is intended to bound $\|x - x_{true}\|_{\infty} / \|x\|_{\infty}$. The quality of the error bound depends on the quality of the computed estimate of $\|A^{-1}\|$ computed in the code; if the estimate is accurate, the error bound is valid.

berr On successful exit, **berr**(j) is the componentwise relative backward error of solution vector j (that is, the smallest relative change in any entry of A or column j of B that makes column j of X an exact solution).

info Status response:

info = 0: Successful exit.

info < 0: If **info** = $-k$, the k -th argument had an invalid value.

info > 0: If **info** = $k \leq n$, $U(k, k)$ is zero. If **info** = $n+1$, the factor U is nonsingular, but **rcond** is less than machine epsilon. The factorization has been completed, but the matrix A is singular to working precision, and the solution and error bounds have not been computed.

Notes If you provide a previously factored matrix, the following output arguments from the **fact** = 'N' or 'n' or 'E' or 'e' call are input arguments to the **fact** = 'F' or 'f' call: **afb**, **ipiv**, **equed**, and possibly **r** and/or **c**, depending on the value of **equed**.

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

fact \neq 'E' or 'e' or 'F' or 'f' or 'N' or 'n',
trans \neq 'N' or 'n' or 'T' or 't' or 'C' or 'c',
n < 0,
kl < 0,
ku < 0,
nrhs < 0,
ldab < **kl**+**ku**+1,
ldafb < 2**kl**+**ku**+1,
fact = 'F' or 'f' and **equed** \neq 'N', 'n', 'B', 'b', 'R', 'r', 'C' or 'c',
r is an input argument but contains a non-positive value,
c is an input argument but contains a non-positive value,
ldb < **max**(1,**n**), and
ldx < **max**(1,**n**).

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding, for example, the **fact** argument as 'Factored' for 'F', 'NoEquilibration' for 'N', or 'Equilibration' for 'E'.

Purpose These subprograms solve a system of linear equations $AX = B$, where A is an n -by- n matrix and X and B are n -by- $nrhs$ matrices. You may supply either a new matrix A or an A that was used in a previous call, together with its previously computed scaling matrices, if any, and its L and U factors.

These subroutines perform the following:

1. If you provide a new A matrix and request equilibration (**fact** = 'E' or 'e'), A is analyzed to determine whether or not to do row equilibration, and, independently, whether or not to do column equilibration. If both equilibrations are done, real diagonal scaling matrices, R and C , are computed to equilibrate the system. If row equilibration is not done, R is the identity; if column equilibration is not done, C is the identity. Output argument **equed** indicates which equilibrations were done.

If you supply a previously factored A matrix (**fact** = 'F' or 'f') and it was equilibrated (**equed** = 'R' or 'r' or 'C' or 'c' or 'B' or 'b'), then the previously computed scaling matrices will be used in the solution phase.

In either case, if equilibration is done, the system actually solved depends upon the **trans** argument as follows:

If **trans** = 'N' or 'n', equilibrate the system as $(RAC)(C^{-1}X) = RB$.

If **trans** = 'T' or 't', equilibrate the system as $(RAC)^T(R^{-1}X) = CB$.

If **trans** = 'C' or 'c', equilibrate the system as $(RAC)*(R^{-1}X) = CB$.

If equilibration is done, A is overwritten by RAC .

If equilibration is done, or if A was factored on a previous call where equilibration was done, then B may be overwritten. Specifically, if **trans** = 'N' or 'n' and **equed** is 'R' or 'r' or 'B' or 'b', then B is overwritten by RB . If **trans** = 'T' or 't' or 'C' or 'c' and **equed** is 'C' or 'c' or 'B' or 'b', then B is overwritten by CB .

2. If **fact** = 'N' or 'n' or 'E' or 'e', A is copied from array **a** to array **af** (after equilibration, if performed), where it is factored as $A = PLU$, where P is a permutation matrix, L is a unit lower triangular matrix, and U is upper triangular.
3. 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 epsilon, steps 4 through 6 are skipped.
4. The system of equations $AX = B$ is solved for X using the factored form of A .
5. Iterative refinement is applied to improve the computed solution vectors and calculate error bounds and backward error estimates.
6. If equilibration was used before factoring the A matrix, X is scaled, if necessary, so as to solve the original system. Specifically, if **trans** = 'N' or 'n' and the final value of **equed** is 'C' or 'c' or 'B' or 'b', then X is premultiplied by C . If **trans** = 'T' or 't' or 'C' or 'c' and the final value of **equed** is 'R' or 'r' or 'B' or 'b', then X is premultiplied by R .

Usage

LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

CHARACTER*1 equed, fact, trans
INTEGER*4 info, lda, ldaf, ldb, ldx, n, nrhs
REAL*4 rcond
INTEGER*4 ipiv(n), iwork(n)
REAL*4 a(lda, n), af(ldaf, n), b(ldb, nrhs),
berr(nrhs), c(n), ferr(nrhs), r(n),
work(3*n), x(ldx, n)
CALL SGESVX (fact, trans, n, nrhs, a, lda, af, ldaf, ipiv, equed,
r, c, b, ldb, x, ldx, rcond, ferr, berr, work,
iwork, info)

CHARACTER*1 equed, fact, trans
INTEGER*4 info, lda, ldaf, ldb, ldx, n, nrhs
REAL*8 rcond
INTEGER*4 ipiv(n), iwork(n)
REAL*8 a(lda, n), af(ldaf, n), b(ldb, nrhs),
berr(nrhs), c(n), ferr(nrhs), r(n),
work(3*n), x(ldx, n)
CALL DGESVX (fact, trans, n, nrhs, a, lda, af, ldaf, ipiv, equed,
r, c, b, ldb, x, ldx, rcond, ferr, berr, work,
iwork, info)

CHARACTER*1 equed, fact, trans
INTEGER*4 info, lda, ldaf, ldb, ldx, n, nrhs
REAL*4 rcond
INTEGER*4 ipiv(n)
REAL*4 berr(nrhs), c(n), ferr(nrhs), r(n), rwork(n)
COMPLEX*8 a(lda, n), af(ldaf, n), b(ldb, nrhs),
work(2*n), x(ldx, nrhs)
CALL CGESVX (fact, trans, n, nrhs, a, lda, af, ldaf, ipiv, equed,
r, c, b, ldb, x, ldx, rcond, ferr, berr, work,
rwork, info)

CHARACTER*1 equed, fact, trans
INTEGER*4 info, lda, ldaf, ldb, ldx, n, nrhs
REAL*8 rcond
INTEGER*4 ipiv(n)
REAL*8 berr(nrhs), c(n), ferr(nrhs), r(n), rwork(n)
COMPLEX*16 a(lda, n), af(ldaf, n), b(ldb, nrhs),
work(2*n), x(ldx, nrhs)
CALL ZGESVX (fact, trans, n, nrhs, a, lda, af, ldaf, ipiv, equed,
r, c, b, ldb, x, ldx, rcond, ferr, berr, work,
rwork, info)

LAPACK8, available on C Series and Exemplar architectures:

CHARACTER*1 equed, fact, trans
INTEGER*8 info, lda, ldaf, ldb, ldx, n, nrhs
REAL*8 rcond
INTEGER*8 ipiv(n), iwork(n)
REAL*8 a(lda, n), af(ldaf, n), b(ldb, nrhs),
 berr(nrhs), c(n), ferr(nrhs), r(n),
 work(3*n), x(ldx, n)
CALL SGEVX (fact, trans, n, nrhs, a, lda, af, ldaf, ipiv, equed,
 r, c, b, ldb, x, ldx, rcond, ferr, berr, work,
 iwork, info)

CHARACTER*1 equed, fact, trans
INTEGER*8 info, lda, ldaf, ldb, ldx, n, nrhs
REAL*8 rcond
INTEGER*8 ipiv(n)
REAL*8 berr(nrhs), c(n), ferr(nrhs), r(n), rwork(n)
COMPLEX*16 a(lda, n), af(ldaf, n), b(ldb, nrhs),
 work(2*n), x(ldx, nrhs)
CALL CGESVX (fact, trans, n, nrhs, a, lda, af, ldaf, ipiv, equed,
 r, c, b, ldb, x, ldx, rcond, ferr, berr, work,
 rwork, info)

Input **fact** 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, as follows:

fact = 'F' or 'f': **af** and **ipiv** contain the factored form of A . If **equed = 'R'** or **'r'** or **'C'** or **'c'** or **'B'** or **'b'**, A was equilibrated before factoring and the scaling matrices are provided in one or both of **r** and **c**.
fact = 'N' or 'n': The matrix A is copied to **af** and factored.
fact = 'E' or 'e': The matrix A is equilibrated, if necessary, then copied to **af** and factored.

trans Specifies the form of the system of equations, as follows:

trans = 'N' or 'n': Solve $AX = B$.
trans = 'T' or 't': Solve $A^T X = B$.
trans = 'C' or 'c': Solve $A^* X = B$.

n The number of linear equations, that is, the order of the matrix A . $n \geq 0$.

nrhs The number of right hand sides, that is, the number of columns of the matrix B . $nrhs \geq 0$.

a The n -by- n matrix A .

lda The leading dimension of array **a** in the calling program unit. $lda \geq \max(1, n)$.

af If **fact = 'F'** or **'f'**, the factors L and U from the factorization $A = PLU$ as computed by a previous call.

Not used as input if **fact = 'N'** or **'n'** or **'E'** or **'e'**.

- ldaf** The leading dimension of array **af** in the calling program unit. $\text{ldaf} \geq \max(1, n)$.
- ipiv** If **fact** = 'F' or 'f', the pivot indices from the factorization $A = PLU$ as computed by a previous call; row i of the matrix was interchanged with row **ipiv**(i).
- Not used as input if **fact** = 'N' or 'n' or 'E' or 'e'.
- equed** If **fact** = 'F' or 'f', the type of equilibration, if any, that was done before factoring A on a previous call, as follows:
- equed** = 'N' or 'n': No equilibration was done.
equed = 'R' or 'r': Only row equilibration was done.
equed = 'C' or 'c': Only column equilibration was done.
equed = 'B' or 'b': Both row and column equilibration were done.
- Not used as input if **fact** = 'N' or 'n' or 'E' or 'e'.
- r** The diagonal elements of the diagonal row scaling matrix R if the matrix A was factored during a previous call (**fact** = 'F' or 'f'), and if row equilibration was done during that call (**equed** = 'R' or 'r' or 'B' or 'b'). $r(i) > 0, i = 1, 2, \dots, n$.
- Otherwise, not used as input.
- c** The diagonal elements of the diagonal column scaling matrix C if the matrix A was factored during a previous call (**fact** = 'F' or 'f'), and if column equilibration was done during that call (**equed** = 'C' or 'c' or 'B' or 'b'). $c(i) > 0, i = 1, 2, \dots, n$.
- Otherwise, not used as input.
- b** The n -by-**nrhs** matrix of right hand side vectors for the system of linear equations $AX = B$.
- ldb** The leading dimension of array **b** in the calling program unit. $\text{ldb} \geq \max(1, n)$.
- ldx** The leading dimension of array **x** in the calling program unit. $\text{ldx} \geq \max(1, n)$.
- Working Storage**
- work, iwork, rwork** Arrays used for work space.
- On successful exit from SGESVX or DGESVX, **work**(1) contains the reciprocal growth factor, $\|A\|_{\Delta} / \|U\|_{\Delta} = (\max_{ij} |a_{ij}|) / (\max_{ij} |u_{ij}|)$.
- On successful exit from CGESVX or ZGESVX, **rwork**(1) contains the reciprocal growth factor, $\|A\|_{\Delta} / \|U\|_{\Delta} = (\max_{ij} |a_{ij}|) / (\max_{ij} |u_{ij}|)$.
- If $\|A\|_{\Delta} / \|U\|_{\Delta} \ll 1$, then the stability of the LU factorization of the (equilibrated) A matrix could be poor. This also means that the solution vector X , reciprocal condition number estimate **rcond**, and forward error bound **ferr** could be unreliable.
- If the factorization fails with $0 < \text{info} \leq n$, then **work**(1) or **rwork**(1) contains the reciprocal growth factor for the leading **info** columns of A .

Output	a	<p>If fact = 'E' or 'e' and equed = 'R', then A was overwritten by RA.</p> <p>If fact = 'E' or 'e' and equed = 'C', then A was overwritten by AC.</p> <p>If fact = 'E' or 'e' and equed = 'B', then A was overwritten by RAC.</p> <p>Not used as output if fact = 'F' or 'f' or 'N' or 'n', or if fact = 'E' or 'e' and equed = 'N' on exit.</p>
	af	<p>If fact = 'N' or 'n' or 'E' or 'e', then af returns the factors L and U from the factorization $A = PLU$ of the matrix A, after equilibration, if performed.</p> <p>Not used as output if fact = 'F' or 'f'.</p>
	ipiv	<p>If fact = 'N' or 'n', then ipiv contains the pivot indices from the factorization $A = PLU$ of the original matrix A.</p> <p>If fact = 'E' or 'e', then ipiv contains the pivot indices from the factorization $A = PLU$ of the equilibrated matrix A.</p> <p>Not used as output if fact = 'F' or 'f'.</p>
	equed	<p>If fact = 'E' or 'e', specifies the form of equilibration that was done, as follows:</p> <p>equed = 'N': No equilibration.</p> <p>equed = 'R': Row equilibration; A was premultiplied by R.</p> <p>equed = 'C': Column equilibration; A was postmultiplied by C.</p> <p>equed = 'B': Both row and column equilibration; A was replaced with RAC.</p> <p>Not used as output if fact = 'F' or 'f' or 'N' or 'n'.</p>
	r	<p>If fact = 'E' or 'e' and equed = 'R' or 'B' on exit, the diagonal elements of the diagonal row scaling matrix R.</p> <p>Destroyed if fact = 'E' or 'e' and equed = 'N' or 'C' on exit.</p> <p>Not used as output if fact = 'F' or 'f' or 'N' or 'n'.</p>
	c	<p>If fact = 'E' or 'e' and equed = 'C' or 'B' on exit, the diagonal elements of the diagonal column scaling matrix C.</p> <p>Destroyed if fact = 'E' or 'e' and equed = 'N' or 'R' on exit.</p> <p>Not used as output if fact = 'F' or 'f' or 'N' or 'n'.</p>
	b	<p>If equed = 'N' or 'n', b is not modified.</p> <p>If trans = 'N' or 'n' and equed = 'R' or 'r' or 'B' or 'b', B is overwritten by RB.</p> <p>If trans = 'T' or 't' or 'C' or 'c' and equed = 'C' or 'c' or 'B' or 'b', B was overwritten by CB.</p>

x On successful exit, the solution vectors X to the original system of equations $AX = B$.

rcond On successful exit, the estimate of the reciprocal condition number of the matrix A , after equilibration, if performed. If **rcond** is small enough so that the logical expression

$$1.0 + \text{rcond} \cdot \text{EQ} \cdot 1.0$$

is true, then A can be regarded as singular to working precision. This condition is indicated by a return code of **info** > 0, and the solution and error bounds are not computed.

ferr On successful exit, estimated forward error bounds for each solution vector. Let x_{true} and x represent column j of the true and computed solutions, respectively. Then **ferr**(j) is intended to bound $\|x - x_{true}\|_{\infty} / \|x\|_{\infty}$. The quality of the error bound depends on the quality of the computed estimate of $\|A^{-1}\|$ computed in the code; if the estimate is accurate, the error bound is valid.

berr On successful exit, **berr**(j) is the componentwise relative backward error of solution vector j (that is, the smallest relative change in any entry of A or column j of B that makes column j of X an exact solution).

info Status response:

info = 0: Successful exit.

info < 0: If **info** = $-k$, the k -th argument had an invalid value.

info > 0: If **info** = $k \leq n$, $U(k,k)$ is zero. If **info** = $n+1$, the factor U is nonsingular, but **rcond** is less than machine epsilon. The factorization has been completed, but the matrix A is singular to working precision, and the solution and error bounds have not been computed.

Notes

If you provide a previously factored matrix, the following output arguments from the **fact** = 'N' or 'n' or 'E' or 'e' call are input arguments to the **fact** = 'F' or 'f' call: **af**, **ipiv**, **equed**, and possibly **r** and/or **c**, depending on the value of **equed**.

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

fact ≠ 'E' or 'e' or 'F' or 'f' or 'N' or 'n',
trans ≠ 'N' or 'n' or 'T' or 't' or 'C' or 'c',
n < 0,
nrhs < 0,
lda < max(1,n),
ldaf < max(1,n),
fact = 'F' or 'f' and **equed** ≠ 'N', 'n', 'B', 'b', 'R', 'r', 'C' or 'c',
r is an input argument but contains a non-positive value,
c is an input argument but contains a non-positive value,
ldb < max(1,n), and
ldx < max(1,n).

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding, for example, the **fact** argument as 'Factored' for 'F', 'NoEquilibration' for 'N', or 'Equilibration' for 'E'.

Solve General Tridiagonal Linear System**SGTSVX/DGTSVX/.../ZGTSVX****Purpose**

These subprograms solve a system of linear equations $AX = B$, where A is a tridiagonal matrix of order n and X and B are n -by- n matrices. A tridiagonal matrix $A = (a_{ij})$ is a matrix whose nonzero elements lie only on the principal diagonal ($i = j$), the subdiagonal ($i = j+1$), and the superdiagonal ($i = j-1$) of the matrix.

These subroutines perform the following:

1. If **fact** = 'N' or 'n', the matrix A is copied from arrays **dl**, **d**, and **du** to arrays **dlf**, **df**, and **duf**, where it is factored as $A = LU$, 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. 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 epsilon, steps 3 and 4 are skipped.
3. The system of equations $AX = B$ is solved for X using the factored form of A .
4. Iterative refinement is applied to improve the computed solution vectors and calculate error bounds and backward error estimates.

Matrix Storage The following example illustrates the storage of general tridiagonal matrices. Consider the following tridiagonal matrix of order $n = 7$:

11	12	0	0	0	0	0
21	22	23	0	0	0	0
0	32	33	34	0	0	0
0	0	43	44	45	0	0
0	0	0	54	55	56	0
0	0	0	0	65	66	67
0	0	0	0	0	76	77

The subdiagonal is stored in array **dl**, the principal diagonal is stored in array **d**, and the superdiagonal is stored in array **du**, as follows:

i	dl (i)	d (i)	du (i)
1	21	11	12
2	32	22	23
3	43	33	34
4	54	44	45
5	65	55	56
6	76	66	67
7		77	

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

CHARACTER*1 fact, trans
INTEGER*4 info, ldb, ldx, n, nrhs
REAL*4 rcond
INTEGER*4 ipiv(n), iwork(n)
REAL*4 b(ldb, nrhs), berr(nrhs), d(n), df(n),
dl(n-1), dlf(n-1), du(n-1), du2(n-2), duf(n-1),
ferr(nrhs), work(3*n), x(ldx, nrhs)
CALL SGTSVX (fact, trans, n, nrhs, dl, d, du, dlf, df, duf,
du2, ipiv, b, ldb, x, ldx, rcond, ferr, berr,
work, iwork, info)

CHARACTER*1 fact, trans
INTEGER*4 info, ldb, ldx, n, nrhs
REAL*8 rcond
INTEGER*4 ipiv(n), iwork(n)
REAL*8 b(ldb, nrhs), berr(nrhs), d(n), df(n),
dl(n-1), dlf(n-1), du(n-1), du2(n-2), duf(n-1),
ferr(nrhs), work(3*n), x(ldx, nrhs)
CALL DGTSVX (fact, trans, n, nrhs, dl, d, du, dlf, df, duf,
du2, ipiv, b, ldb, x, ldx, rcond, ferr, berr,
work, iwork, info)

CHARACTER*1 fact, trans
INTEGER*4 info, ldb, ldx, n, nrhs
REAL*4 rcond
INTEGER*4 ipiv(n)
REAL*4 berr(nrhs), ferr(nrhs), rwork(n)
COMPLEX*8 b(ldb, nrhs), d(n), df(n),
dl(n-1), dlf(n-1), du(n-1), du2(n-2), duf(n-1),
work(2*n), x(ldx, nrhs)
CALL CGTSVX (fact, trans, n, nrhs, dl, d, du, dlf, df, duf,
du2, ipiv, b, ldb, x, ldx, rcond, ferr, berr,
work, rwork, info)

CHARACTER*1 fact, trans
INTEGER*4 info, ldb, ldx, n, nrhs
REAL*8 rcond
INTEGER*4 ipiv(n)
REAL*8 berr(nrhs), ferr(nrhs), rwork(n)
COMPLEX*16 b(ldb, nrhs), d(n), df(n),
dl(n-1), dlf(n-1), du(n-1), du2(n-2), duf(n-1),
work(2*n), x(ldx, nrhs)
CALL ZGTSVX (fact, trans, n, nrhs, dl, d, du, dlf, df, duf,
du2, ipiv, b, ldb, x, ldx, rcond, ferr, berr,
work, rwork, info)

LAPACK8, available on C Series and Exemplar architectures:

```

CHARACTER*1 fact, trans
INTEGER*8 info, ldb, ldx, n, nrhs
REAL*8 rcond
INTEGER*8 ipiv(n), iwork(n)
REAL*8 b(ldb, nrhs), berr(nrhs), d(n), df(n),
dl(n-1), dlf(n-1), du(n-1), du2(n-2), duf(n-1),
ferr(nrhs), work(3*n), x(ldx, nrhs)
CALL SGTSVX (fact, trans, n, nrhs, dl, d, du, dlf, df, duf,
du2, ipiv, b, ldb, x, ldx, rcond, ferr, berr,
work, iwork, info)

```

```

CHARACTER*1 fact, trans
INTEGER*8 info, ldb, ldx, n, nrhs
REAL*8 rcond
INTEGER*8 ipiv(n)
REAL*8 berr(nrhs), ferr(nrhs), rwork(n)
COMPLEX*16 b(ldb, nrhs), d(n), df(n),
dl(n-1), dlf(n-1), du(n-1), du2(n-2), duf(n-1),
work(2*n), x(ldx, nrhs)
CALL CGTSVX (fact, trans, n, nrhs, dl, d, du, dlf, df, duf,
du2, ipiv, b, ldb, x, ldx, rcond, ferr, berr,
work, rwork, info)

```

Input	fact	Specifies whether or not the factored form of A has been supplied on entry, as follows: fact = 'F' or 'f': dlf, df, duf, du2, and ipiv2 contain the factored form of A . fact = 'N' or 'n': The matrix is copied to dlf, df, duf, and du2 and factored.
	trans	Specifies the form of the system of equations, as follows: trans = 'N' or 'n': Solve $AX = B$. trans = 'T' or 't': Solve $A^T = B$. trans = 'C' or 'c': Solve $A * X = B$.
	n	The order of the matrix A . $n \geq 0$.
	nrhs	The number of right hand sides, that is, the number of columns of the matrix B . $nrhs \geq 0$.
	dl	The $n-1$ subdiagonal elements of A .
	d	The n diagonal elements of A .
	du	The $n-1$ superdiagonal elements of A .
	dlf	If fact = 'F' or 'f' , the $n-1$ multipliers that define the matrix L from the LU factorization of A as computed by a previous call. Not used as input if fact = 'N' or 'n' or 'E' or 'e' .

	df	If fact = 'F' or 'f', the n diagonal elements of the upper triangular matrix <i>U</i> from the <i>LU</i> factorization of <i>A</i> . Not used as input if fact = 'N' or 'n' or 'E' or 'e'.
	duf	If fact = 'F' or 'f', the n-1 elements of the first superdiagonal of <i>U</i> . Not used as input if fact = 'N' or 'n' or 'E' or 'e'.
	du2	If fact = 'F' or 'f', the n-2 elements of the second superdiagonal of <i>U</i> . Not used as input if fact = 'N' or 'n' or 'E' or 'e'.
	ipiv	If fact = 'F' or 'f', the pivot indices from the <i>LU</i> factorization of <i>A</i> as computed by a previous call.
	b	The n-by-nrhs matrix of right hand side vectors for the system of equations $AX = B$.
	ldb	The leading dimension of array b in the calling program unit. $ldb \geq \max(1,n)$.
	ldx	The leading dimension of array x in the calling program unit. $ldx \geq \max(1,n)$.
Working Storage	work, iwork, rwork	Arrays used for work space.
Output	dlf	If fact = 'N' or 'n', the n-1 multipliers that define the matrix <i>L</i> from the <i>LU</i> factorization of <i>A</i> . Not used as output if fact = 'F' or 'f'.
	df	If fact = 'N' or 'n', the n diagonal elements of the upper triangular matrix <i>U</i> from the <i>LU</i> factorization of <i>A</i> . Not used as output if fact = 'F' or 'f'.
	duf	If fact = 'N' or 'n', the n-1 elements of the first superdiagonal of <i>U</i> . Not used as output if fact = 'F' or 'f'.
	du2	If fact = 'N' or 'n', the n-2 elements of the second superdiagonal of <i>U</i> . Not used as output if fact = 'F' or 'f'.
	ipiv	If fact = 'N' or 'n', the pivot indices from the <i>LU</i> factorization of <i>A</i> ; row <i>i</i> of the matrix was interchanged with row ipiv(i) . ipiv(i) will always be either <i>i</i> or <i>i+1</i> ; ipiv(i) = i indicates a row interchange was not required.
	x	On successful exit, the n-by-nrhs matrix of solution vectors <i>X</i> for the system of equations $AX = B$.

rcond On successful exit, the estimate of the reciprocal condition number of the matrix A . If **rcond** is small enough so that the logical expression

$$1.0 + \text{rcond} \text{ .EQ. } 1.0$$

is true, then A can be regarded as singular to working precision. This condition is indicated by a return code of **info** > 0, and the solution and error bounds are not computed.

ferr On successful exit, estimated forward error bounds for each solution vector. Let x_{true} and x represent column j of the true and computed solutions, respectively. Then **ferr**(j) is intended to bound $\|x - x_{true}\|_{\infty} / \|x\|_{\infty}$. The quality of the error bound depends on the quality of the computed estimate of $\|A^{-1}\|$ computed in the code; if the estimate is accurate, the error bound is valid.

berr On successful exit, **berr**(j) is the componentwise relative backward error of solution vector j (that is, the smallest relative change in any entry of A or column j of B that makes column j of X an exact solution).

info Status response:

info = 0: Successful exit.

info < 0: If **info** = $-k$, the k -th argument had an invalid value.

info > 0: If **info** = $k \leq n$, $U(k,k)$ is zero. The factorization has not been completed unless $k = n$, but the factor U is exactly singular, so the solution and error bounds could not be computed.

If **info** = $n+1$, **rcond** is less than machine epsilon. The factorization has been completed, but the matrix A is singular to working precision, and the solution and error bounds have not been computed.

Notes

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

fact \neq 'F' or 'f' or 'N' or 'n',
trans \neq 'T' or 't' or 'C' or 'c',
n < 0,
nrhs < 0,
ldb < max(1,n), and
ldx < max(1,n).

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the CALL statement may be improved by coding, for example, the **fact** argument as 'Factored' for 'F', 'NoEquilibration' for 'N', or 'Equilibration' for 'E'.

Purpose These subprograms solve a system of linear equations $AX = B$, where A is an n -by- n real symmetric or complex Hermitian positive definite band matrix and X and B are n -by- $nrhs$ matrices. You may supply either a new matrix A or an A that was used in a previous call, together with its previously computed scaling matrix, if any, and its L or U factor:

A real matrix is symmetric if $A = A^T$, its transpose, and a real symmetric matrix A is positive definite if the quadratic form $x^T Ax$ is positive for all nonzero real vectors x . A complex matrix is Hermitian if $A = A^*$, its conjugate transpose, and a complex Hermitian matrix A is positive definite if the quadratic form $x^* Ax$ is positive for all nonzero complex vectors x .

A positive definite band matrix is a positive definite matrix whose nonzero elements all lie fairly near the principal diagonal. Specifically, $a_{ij} = 0$ if $|i-j| > kd$ for some integer kd . The smallest such kd for a given matrix is called the half bandwidth, and $2kd+1$ is called the total bandwidth.

Tridiagonal matrices are the special case $kd = 1$. They can be handled more efficiently by the LAPACK subprograms SPTSVX, DPTSVX, CPTSVX, and ZPTSVX.

These subroutines perform the following:

1. If you provide a new A matrix and request equilibration (**fact** = 'E' or 'e'), A is analyzed to determine whether or not to do equilibration. If equilibration is done, a real diagonal scaling matrix, S , is computed to equilibrate the system. If equilibration is not done, S is the identity.

If you supply a previously factored A matrix (**fact** = 'F' or 'f') and it was equilibrated (**equed** = 'Y' or 'y'), then the previously computed scaling matrix will be used in the solution phase.

In either case, if equilibration is done, the system actually solved is

$$(SAS)(S^{-1}X) = SB.$$

If equilibration is done, A is overwritten by SAS .

If equilibration is done, or if A was factored in a previous call where equilibration was done, then B is overwritten by SB .

2. If **fact** = 'N' or 'n' or 'E' or 'e', the matrix A is copied from array **ab** to array **afb** (after equilibration if **fact** = 'E' or 'e') where the Cholesky decomposition is used to factor A as

$$A = U^*U, \text{ if } \text{uplo} = \text{'U'} \text{ or 'u'}, \text{ or}$$

$$A = LL^*, \text{ if } \text{uplo} = \text{'L'} \text{ or 'l'},$$

where U is an upper triangular matrix, L is a lower triangular matrix, and $*$ indicates conjugate transpose.

3. 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 epsilon, steps 4 through 6 are skipped.
4. The system of equations $AX = B$ is solved for X using the factored form of A .
5. Iterative refinement is applied to improve the computed solution vectors and calculate error bounds and backward error estimates.
6. If equilibration was used before factoring the A matrix, X is scaled so as to solve the original system.

Matrix Storage Because it is not necessary to store or operate on the zeros outside the band of A , and since either triangle of A may be obtained from the other, you need only provide the band within one triangle of A . Compared to storing the entire matrix, this can save memory in two ways: only the elements within the band are stored, and of them, only the upper or the lower triangle.

The following examples illustrate the storage of positive definite band matrices. Consider the following matrix A of order $n = 7$ and half bandwidth $kd = 2$:

11	12	13	0	0	0	0
12	22	23	24	0	0	0
13	23	33	34	35	0	0
0	24	34	44	45	46	0
0	0	35	45	55	56	57
0	0	0	46	56	66	67
0	0	0	0	57	67	77

Upper triangular storage. The upper triangle of A is stored in an array \mathbf{ab} with at least $kd+1 = 3$ rows and 7 columns as follows:

*	*	13	24	35	46	57
*	12	23	34	45	56	67
11	22	33	44	55	66	77

The asterisks represent elements in the kd -by- kd triangle at the upper left corner of \mathbf{ab} that are not referenced. Thus, if a_{ij} is an element within the band of the upper triangle of A , it is stored in $\mathbf{ab}(kd+1+i-j, j)$. Therefore, the columns of the upper triangle of A are stored in the columns of \mathbf{ab} , and the diagonals of the upper triangle of A are stored in the rows of \mathbf{ab} .

Lower triangular storage. The lower triangle of A is stored in the array \mathbf{ab} as follows:

11	22	33	44	55	66	77
12	23	34	45	56	67	*
13	24	35	46	57	*	*

The asterisks represent elements in the kd -by- kd triangle at the lower right corner of \mathbf{ab} that are not referenced. Thus, if a_{ij} is an element within the band of the lower triangle of A , it is stored in $\mathbf{ab}(1+i-j, j)$. Therefore, the columns of the lower triangle of A are stored in the columns of \mathbf{ab} , and the diagonals of the lower triangle of A are stored in the rows of \mathbf{ab} .

Usage

LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

CHARACTER*1 equed, fact, uplo
INTEGER*4 info, kd, ldab, ldafb, ldb, ldx, n, nrhs
REAL*4 rcond
INTEGER*4 iwork(n)
REAL*4 ab(ldab, n), afb(ldafb, n), b(ldb, nrhs),
 berr(nrhs), ferr(nrhs), s(n), work(3*n),
 x(ldx, nrhs)
CALL SPBSVX (fact, uplo, n, kd, nrhs, ab, ldab, afb, ldafb,
 equed, s, b, ldb, x, ldx, rcond, ferr, berr, work,
 iwork, info)

CHARACTER*1 equed, fact, uplo
INTEGER*4 info, kd, ldab, ldafb, ldb, ldx, n, nrhs
REAL*8 rcond
INTEGER*4 iwork(n)
REAL*8 ab(ldab, n), afb(ldafb, n), b(ldb, nrhs),
 berr(nrhs), ferr(nrhs), s(n), work(3*n),
 x(ldx, nrhs)
CALL DPBSVX (fact, uplo, n, kd, nrhs, ab, ldab, afb, ldafb,
 equed, s, b, ldb, x, ldx, rcond, ferr, berr, work,
 iwork, info)

CHARACTER*1 equed, fact, uplo
INTEGER*4 info, kd, ldab, ldafb, ldb, ldx, n, nrhs
REAL*4 rcond
REAL*4 berr(nrhs), ferr(nrhs), s(n), rwork(n)
COMPLEX*8 ab(ldab, n), afb(ldafb, n), b(ldb, nrhs),
 work(2*n), x(ldx, nrhs)
CALL CPBSVX (fact, uplo, n, kd, nrhs, ab, ldab, afb, ldafb,
 equed, s, b, ldb, x, ldx, rcond, ferr, berr, work,
 rwork, info)

CHARACTER*1 equed, fact, uplo
INTEGER*4 info, kd, ldab, ldafb, ldb, ldx, n, nrhs
REAL*8 rcond
REAL*8 berr(nrhs), ferr(nrhs), s(n), rwork(n)
COMPLEX*16 ab(ldab, n), afb(ldafb, n), b(ldb, nrhs),
 work(2*n), x(ldx, nrhs)
CALL ZPBSVX (fact, uplo, n, kd, nrhs, ab, ldab, afb, ldafb,
 equed, s, b, ldb, x, ldx, rcond, ferr, berr, work,
 rwork, info)

LAPACK8, available on C Series and Exemplar architectures:

CHARACTER*1 equed, fact, uplo
INTEGER*8 info, kd, ldab, ldafb, ldb, ldx, n, nrhs
REAL*8 rcond
INTEGER*8 iwork(n)
REAL*8 ab(ldab, n), afb(ldafb, n), b(ldb, nrhs),
 berr(nrhs), ferr(nrhs), s(n), work(3*n),
 x(ldx, nrhs)
CALL SPBSVX (fact, uplo, n, kd, nrhs, ab, ldab, afb, ldafb,
 equed, s, b, ldb, x, ldx, rcond, ferr, berr, work,
 iwork, info)

CHARACTER*1 equed, fact, uplo
INTEGER*8 info, kd, ldab, ldafb, ldb, ldx, n, nrhs
REAL*8 rcond
REAL*8 berr(nrhs), ferr(nrhs), s(n), rwork(n)
COMPLEX*16 ab(ldab, n), afb(ldafb, n), b(ldb, nrhs),
work(2*n), x(ldx, nrhs)
CALL CPBSVX (fact, uplo, n, kd, nrhs, ab, ldab, afb, ldafb,
equed, s, b, ldb, x, ldx, rcond, ferr, berr, work,
rwork, info)

Input

fact 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, as follows:

fact = 'F' or 'f': **afb** contains the factored form of A . If **equed = 'Y' or 'y'**, A was equilibrated before factoring and the scaling matrix is provided in **s**.

fact = 'N' or 'n': The matrix A is copied to **afb** and factored.

fact = 'E' or 'e': The matrix A is equilibrated, if necessary, then copied to **afb** and factored.

uplo Specifies whether the upper or lower triangular part of the symmetric or Hermitian matrix A is stored, as follows:

uplo = 'U' or 'u': The upper triangular part of A is stored.

uplo = 'L' or 'l': The lower triangular part of A is stored.

n The number of linear equations, that is, the order of the matrix A . $n \geq 0$.

kd The number of super-diagonals of the matrix A if **uplo = 'U' or 'u'**, or the number of sub-diagonals if **uplo = 'L' or 'l'**. $kd \geq 0$.

nrhs The number of right hand sides, that is, the number of columns of the matrix B . $nrhs \geq 0$.

ab The upper or lower triangle of the real symmetric or complex Hermitian band matrix A , stored in the first $kd+1$ rows of the array. The j -th column of A is stored in the j -th column of the array **ab** as follows:

If **uplo = 'U' or 'u'**, $ab(kd+1+i-j, j) = A(i, j)$ for $\max(1, j-kd) \leq i \leq j$.

If **uplo = 'L' or 'l'**, $ab(1+i-j, j) = A(i, j)$ for $j \leq i \leq \min(n, j+kd)$.

ldab The leading dimension of array **ab** in the calling program unit. **ldab** $\geq kd+1$.

afb If **fact = 'F' or 'f'**, the triangular factor U or L from the Cholesky factorization $A = U*U$ or $A = LL^*$ of the band matrix A , in the same storage format as A (see **ab**).

ldafb The leading dimension of array **afb** in the calling program unit. **ldafb** $\geq kd+1$.

	equed	If fact = 'F' or 'f', the type of equilibration, if any, that was done before factoring A on a previous call, as follows: equed = 'N' or 'n': No equilibration was done. equed = 'Y' or 'y': Equilibration was done. Not used as input if fact = 'N' or 'n' or 'E' or 'e'.
	s	The diagonal elements of the diagonal scaling matrix S if the matrix A was factored during a previous call (fact = 'F' or 'f'), and if equilibration was done during that call (equed = 'Y' or 'y'). $s(i) > 0, i = 1, 2, \dots, n$. Otherwise, not used as input.
	b	The n -by- $nrhs$ matrix of right hand side vectors for the system of linear equations $AX = B$.
	ldb	The leading dimension of array b in the calling program unit. $ldb \geq \max(1, n)$.
	ldx	The leading dimension of array x in the calling program unit. $ldx \geq \max(1, n)$.
Working Storage	work, iwork, rwork	Arrays used for work space.
Output	ab	If fact = 'E' or 'e' and equed = 'Y', then A was overwritten by SAS . Not used as output if fact = 'F' or 'f' or 'N' or 'n', or if fact = 'E' or 'e' and equed = 'N' on exit.
	afb	If fact = 'N' or 'n' or 'E' or 'e', the triangular factor U or L from the Cholesky factorization $A = U*U$ or $A = LL^*$ of the matrix A , after equilibration, if performed. Not used as output if fact = 'F' or 'f'.
	equed	If fact = 'E' or 'e', specifies the form of equilibration that was done, as follows: equed = 'N': No equilibration. equed = 'Y': Equilibration was done; A was replaced with SAS . Not used as output if fact = 'F' or 'f' or 'N' or 'n'.
	s	If fact = 'E' or 'e' and equed = 'Y' on exit, the diagonal elements of the diagonal scaling matrix S . Destroyed if fact = 'E' or 'e' and equed = 'N' on exit. Not used as output if fact = 'F' or 'f' or 'N' or 'n'.
	b	If equed = 'N' or 'n', b is not modified. If equed = 'Y' or 'y', B was overwritten by SB .

x On successful exit, the solution vectors X to the original system of equations $AX = B$.

rcond On successful exit, the estimate of the reciprocal condition number of the matrix A , after equilibration, if performed. If **rcond** is small enough so that the logical expression

$$1.0 + \text{rcond} \text{ .EQ. } 1.0$$

is true, then A can be regarded as singular to working precision. This condition is indicated by a return code of **info** > 0, and the solution and error bounds are not computed.

ferr On successful exit, estimated forward error bounds for each solution vector. Let x_{true} and x represent column j of the true and computed solutions, respectively. Then **ferr**(j) is intended to bound $\|x - x_{true}\|_{\infty} / \|x\|_{\infty}$. The quality of the error bound depends on the quality of the computed estimate of $\|A^{-1}\|$ computed in the code; if the estimate is accurate, the error bound is valid.

berr On successful exit, **berr**(j) is the componentwise relative backward error of solution vector j (that is, the smallest relative change in any entry of A or column j of B that makes column j of X an exact solution).

info Status response:

info = 0: Successful exit.

info < 0: If **info** = $-k$, the k -th argument had an invalid value.

info > 0: If **info** = $k \leq n$, the leading minor of order k of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

If **info** = $n+1$, **rcond** is less than machine epsilon. The factorization has been completed, but the matrix A is singular to working precision, and the solution and error bounds have not been computed.

Notes

If you provide a previously factored matrix, the following output arguments from the **fact** = 'N' or 'n' or 'E' or 'e' call are input arguments to the **fact** = 'F' or 'f' call: **afb**, **equed**, and possibly **s**, depending on the value of **equed**.

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

fact ≠ 'E' or 'e' or 'F' or 'f' or 'N' or 'n',
uplo ≠ 'U' or 'u' or 'L' or 'l',
n < 0,
kd < 0,
nrhs < 0,
ldab < **kd**+1,
ldafb < **kd**+1,
fact = 'F' or 'f' and **equed** ≠ 'N' or 'n' or 'Y' or 'y',
s is an input argument but contains a non-positive value,
ldb < max(1,**n**), and
ldx < max(1,**n**).

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding, for example, the **uplo** argument as 'Lower' for 'L' or 'Upper' for 'U'.

Solve Positive Definite Linear System**SPOSVX/DPOSVX/.../ZPOSVX****Purpose**

These subprograms solve a system of linear equations $AX = B$, where A is an n -by- n real symmetric or complex Hermitian positive definite matrix and X and B are n -by- $nrhs$ matrices. You may supply either a new matrix A or an A that was used in a previous call, together with its previously computed scaling matrix, if any, and its L or U factor.

A real matrix is symmetric if $A = A^T$, its transpose. A real symmetric matrix A is positive definite if the quadratic form $x^T Ax$ is positive for all nonzero real vectors x . A complex matrix is Hermitian if $A = A^*$, its conjugate transpose, and a complex Hermitian matrix A is positive definite if the quadratic form $x^* Ax$ is positive for all nonzero complex vectors x .

These subroutines perform the following:

1. If you provide a new A matrix and request equilibration (**fact** = 'E' or 'e'), A is analyzed to determine whether or not to do equilibration. If equilibration is done, a real diagonal scaling matrix, S , is computed to equilibrate the system. If equilibration is not done, S is the identity.

If you supply a previously factored A matrix (**fact** = 'F' or 'f') and it was equilibrated (**equed** = 'Y' or 'y'), then the previously computed scaling matrix will be used in the solution phase.

In either case, if equilibration is done, the system actually solved is

$$(SAS)(S^{-1}X) = SB.$$

If equilibration is done, A is overwritten by SAS .

If equilibration is done, or if A was factored in a previous call where equilibration was done, then B is overwritten by SB .

2. If **fact** = 'N' or 'n' or 'E' or 'e', the matrix A is copied from array **a** to array **af** (after equilibration if **fact** = 'E' or 'e') where the Cholesky decomposition is used to factor A as

$$A = U^*U, \text{ if } \mathbf{uplo} = \text{'U' or 'u'}, \text{ or}$$

$$A = LL^*, \text{ if } \mathbf{uplo} = \text{'L' or 'l'},$$

where U is an upper triangular matrix, L is a lower triangular matrix, and $*$ indicates conjugate transpose.

3. 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 epsilon, steps 4 through 6 are skipped.
4. The system of equations $AX = B$ is solved for X using the factored form of A .
5. Iterative refinement is applied to improve the computed solution vectors and calculate error bounds and backward error estimates.
6. If equilibration was used before factoring the A matrix, X is scaled so as to solve the original system.

Matrix Storage Because either triangle of A may be obtained from the other, you need only provide one triangle of A . You may supply either the upper or the lower triangle of A , in a two-dimensional array large enough to hold the entire matrix. The other triangle of the array is not referenced.

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

CHARACTER*1 equed, fact, uplo
INTEGER*4 info, lda, ldaf, ldb, ldx, n, nrhs
REAL*4 rcond
INTEGER*4 iwork(n)
REAL*4 a(lda, n), af(ldaf, n), b(ldb, nrhs), berr(nrhs),
 ferr(nrhs), s(n), work(3*n), x(ldx, nrhs)
CALL SPOSVX (fact, uplo, n, nrhs, a, lda, af, ldaf, equed, s, b,
 ldb, x, ldx, rcond, ferr, berr, work, iwork, info)

CHARACTER*1 equed, fact, uplo
INTEGER*4 info, lda, ldaf, ldb, ldx, n, nrhs
REAL*8 rcond
INTEGER*4 iwork(n)
REAL*8 a(lda, n), af(ldaf, n), b(ldb, nrhs), berr(nrhs),
 ferr(nrhs), s(n), work(3*n), x(ldx, nrhs)
CALL DPOSVX (fact, uplo, n, nrhs, a, lda, af, ldaf, equed, s, b,
 ldb, x, ldx, rcond, ferr, berr, work, iwork, info)

CHARACTER*1 equed, fact, uplo
INTEGER*4 info, lda, ldaf, ldb, ldx, n, nrhs
REAL*4 rcond
REAL*4 berr(nrhs), ferr(nrhs), rwork(n), s(n)
COMPLEX*8 a(lda, n), af(ldaf, n), b(ldb, nrhs),
 x(ldx, nrhs), work(2*n)
CALL CPOSVX (fact, uplo, n, nrhs, a, lda, af, ldaf, equed, s, b,
 ldb, x, ldx, rcond, ferr, berr, work, rwork, info)

CHARACTER*1 equed, fact, uplo
INTEGER*4 info, lda, ldaf, ldb, ldx, n, nrhs
REAL*8 rcond
REAL*8 berr(nrhs), ferr(nrhs), rwork(n), s(n)
COMPLEX*16 a(lda, n), af(ldaf, n), b(ldb, nrhs),
 x(ldx, nrhs), work(2*n)
CALL ZPOSVX (fact, uplo, n, nrhs, a, lda, af, ldaf, equed, s, b,
 ldb, x, ldx, rcond, ferr, berr, work, rwork, info)

LAPACK8, available on C Series and Exemplar architectures:

CHARACTER*1 equed, fact, uplo
INTEGER*8 info, lda, ldaf, ldb, ldx, n, nrhs
REAL*8 rcond
INTEGER*8 iwork(n)
REAL*8 a(lda, n), af(ldaf, n), b(ldb, nrhs), berr(nrhs),
 ferr(nrhs), s(n), work(3*n), x(ldx, nrhs)
CALL SPOSVX (fact, uplo, n, nrhs, a, lda, af, ldaf, equed, s, b,
 ldb, x, ldx, rcond, ferr, berr, work, iwork, info)

CHARACTER*1 equed, fact, uplo
INTEGER*4 info, lda, ldaf, ldb, ldx, n, nrhs
REAL*8 rcond
REAL*8 berr(nrhs), ferr(nrhs), rwork(n), s(n)
COMPLEX*16 a(lda, n), af(ldaf, n), b(ldb, nrhs),
x(ldx, nrhs), work(2*n)
CALL CPOSVX (fact, uplo, n, nrhs, a, lda, af, ldaf, equed, s, b,
ldb, x, ldx, rcond, ferr, berr, work, rwork, info)

Input

fact 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, as follows:

fact = 'F' or 'f': **af** contains the factored form of A . If **equed = 'Y' or 'y'**, A was equilibrated before factoring and the scaling matrix is provided in **s**.

fact = 'N' or 'n': The matrix A is copied to **af** and factored.

fact = 'E' or 'e': The matrix A is equilibrated, if necessary, then copied to **af** and factored.

uplo Specifies whether the upper or lower triangular part of the symmetric or Hermitian matrix A is stored, as follows:

uplo = 'U' or 'u': The upper triangular part of A is stored.

uplo = 'L' or 'l': The lower triangular part of A is stored.

n The number of linear equations, that is, the order of the matrix A . $n \geq 0$.

nrhs The number of right hand sides, that is, the number of columns of the matrix B . $nrhs \geq 0$.

a The symmetric matrix A .

If **uplo = 'U' or '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' or '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 The leading dimension of array **a** in the calling program unit. $lda \geq \max(1, n)$.

af If **fact = 'F' or 'f'**, the triangular factor U or L from the Cholesky factorization $A = U^*U$ or $A = LL^*$, in the same storage format as A .

Not used as input if **fact = 'N' or 'n' or 'E' or 'e'**.

ldaf The leading dimension of array **af** in the calling program unit. $ldaf \geq \max(1, n)$.

	equed	If fact = 'F' or 'f', the type of equilibration, if any, that was done before factoring <i>A</i> on a previous call, as follows: equed = 'N' or 'n': No equilibration was done. equed = 'Y' or 'y': Equilibration was done. Not used as input if fact = 'N' or 'n' or 'E' or 'e'.
	s	The diagonal elements of the diagonal scaling matrix <i>S</i> if the matrix <i>A</i> was factored during a previous call (fact = 'F' or 'f'), and if equilibration was done during that call (equed = 'Y' or 'y'). $s(i) > 0, i = 1, 2, \dots, n$. Otherwise, not used as input.
	b	The <i>n</i> -by- nrhs matrix of right hand side vectors for the system of linear equations $AX = B$.
	ldb	The leading dimension of array b in the calling program unit. $ldb \geq \max(1, n)$.
	ldx	The leading dimension of array x in the calling program unit. $ldx \geq \max(1, n)$.
Working Storage	work, iwork, rwork	Arrays used for work space.
Output	a	If fact = 'E' or 'e' and equed = 'Y', then <i>A</i> was overwritten by <i>SAS</i> . Not used as output if fact = 'F' or 'f' or 'N' or 'n', or if fact = 'E' or 'e' and equed = 'N' on exit.
	af	If fact = 'N' or 'n' or 'E' or 'e', then af returns the triangular factor <i>U</i> or <i>L</i> from the Cholesky factorization $A = U*U$ or $A = LL^*$ of the matrix <i>A</i> , after equilibration, if performed. Not used as output if fact = 'F' or 'f'.
	equed	If fact = 'E' or 'e', specifies the form of equilibration that was done, as follows: equed = 'N': No equilibration. equed = 'Y': Equilibration was done; <i>A</i> was replaced with <i>SAS</i> . Not used as output if fact = 'F' or 'f' or 'N' or 'n'.
	s	If fact = 'E' or 'e' and equed = 'Y' on exit, the diagonal elements of the diagonal scaling matrix <i>S</i> . Destroyed if fact = 'E' or 'e' and equed = 'N' on exit. Not used as output if fact = 'F' or 'f' or 'N' or 'n'.
	b	If equed = 'N' or 'n', b is not modified. If equed = 'Y' or 'y', <i>B</i> was overwritten by <i>SB</i> .

x On successful exit, the solution vectors X to the original system of equations $AX = B$.

rcond On successful exit, the estimate of the reciprocal condition number of the matrix A , after equilibration, if performed. If **rcond** is small enough so that the logical expression

$$1.0 + \text{rcond} \cdot \text{EQ} \cdot 1.0$$

is true, then A can be regarded as singular to working precision. This condition is indicated by a return code of **info** > 0 , and the solution and error bounds are not computed.

ferr On successful exit, estimated forward error bounds for each solution vector. Let x_{true} and x represent column j of the true and computed solutions, respectively. Then **ferr**(j) is intended to bound $\|x - x_{true}\|_{\infty} / \|x\|_{\infty}$. The quality of the error bound depends on the quality of the computed estimate of $\|A^{-1}\|$ computed in the code; if the estimate is accurate, the error bound is valid.

berr On successful exit, **berr**(j) is the componentwise relative backward error of solution vector j (that is, the smallest relative change in any entry of A or column j of B that makes column j of X an exact solution).

info Status response:

info = 0: Successful exit.

info < 0: If **info** = $-k$, the k -th argument had an invalid value.

info > 0: If **info** = $k \leq n$, the leading minor of order k of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

If **info** = $n+1$, **rcond** is less than machine epsilon. The factorization has been completed, but the matrix A is singular to working precision, and the solution and error bounds have not been computed.

Notes

If you provide a previously factored matrix, the following output arguments from the **fact** = 'N' or 'n' or 'E' or 'e' call are input arguments to the **fact** = 'F' or 'f' call: **af**, **equed**, and possibly **s**, depending on the value of **equed**.

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

fact \neq 'E' or 'e' or 'F' or 'f' or 'N' or 'n',
uplo \neq 'U' or 'u' or 'L' or 'l',
n < 0,
nrhs < 0,
lda < max(1,n),
ldaf < max(1,n),
fact = 'F' or 'f' and **equed** \neq 'N' or 'n' or 'Y' or 'y',
s is an input argument but contains a non-positive value,
ldb < max(1,n), and
ldx < max(1,n).

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding, for example, the **uplo** argument as 'Lower' for 'L' or 'Upper' for 'U'.

Solve Positive Definite Packed Linear System**SPPSVX/.../ZPPSVX****Purpose**

These subprograms to solve a system of linear equations $AX = B$, where A is an n -by- n real symmetric or complex Hermitian positive definite matrix stored in packed form and X and B are n -by- n matrices. You may supply either a new matrix A or an A that was used in a previous call, together with its previously computed scaling matrix, if any, and its L or U factor.

A real matrix is symmetric if $A = A^T$, its transpose, and a real symmetric matrix A is positive definite if the quadratic form $x^T A x$ is positive for all nonzero real vectors x . a complex matrix is Hermitian if $A = A^*$, its conjugate transpose. a complex Hermitian matrix A is positive definite if the quadratic form $x^* A x$ is positive for all nonzero complex vectors x .

These subroutines perform the following:

1. If you provide a new A matrix and request equilibration (**fact** = 'E' or 'e'), A is analyzed to determine whether or not to do equilibration. If equilibration is done, a real diagonal scaling matrix, S , is computed to equilibrate the system. If equilibration is not done, S is the identity.

If you supply a previously factored A matrix (**fact** = 'F' or 'f') and it was equilibrated (**equed** = 'Y' or 'y'), then the previously computed scaling matrix will be used in the solution phase.

In either case, if equilibration is done, the system actually solved is

$$(SAS)(S^{-1}X) = SB.$$

If equilibration is done, A is overwritten by SAS .

If equilibration is done, or if A was factored in a previous call where equilibration was done, then B is overwritten by SB .

2. If **fact** = 'N' or 'n' or 'E' or 'e', the matrix A is copied from array **ap** to array **afp** (after equilibration if **fact** = 'E' or 'e') where the Cholesky decomposition is used to factor A as

$$A = U^* U, \text{ if } \text{uplo} = \text{'U'} \text{ or 'u'}, \text{ or}$$

$$A = LL^*, \text{ if } \text{uplo} = \text{'L'} \text{ or 'l'},$$

where U is an upper triangular matrix, L is a lower triangular matrix, and $*$ indicates conjugate transpose.

3. 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 epsilon, steps 4 through 6 are skipped.
4. The system of equations $AX = B$ is solved for X using the factored form of A .
5. Iterative refinement is applied to improve the computed solution vectors and calculate error bounds and backward error estimates.
6. If equilibration was used before factoring the A matrix, X is scaled so as to solve the original system.

Matrix Storage Because either triangle of A may be obtained from the other, you need only provide one triangle of A , either the upper or the lower triangle. Compared to storing the entire matrix, you save memory by supplying that triangle stored column-by-column in packed form in a 1-dimensional array.

The following examples illustrate the packed storage of symmetric or Hermitian matrices.

Upper triangular storage. If the upper triangle of A is

11	12	13	14
	22	23	24
		33	34
			44

then A is packed column-by-column into an array **ap** as follows:

k	1	2	3	4	5	6	7	8	9	10
ap (k)	11	12	22	13	23	33	14	24	34	44

Upper triangular matrix element a_{ij} is stored in array element **ap**($i+j \times (j-1)/2$).

Lower triangular storage. If the lower triangle of A is

11										
21	22									
31	32	33								
41	42	43	44							

then A is packed column-by-column into an array **ap** as follows:

k	1	2	3	4	5	6	7	8	9	10
ap (k)	11	21	31	41	22	32	42	33	43	44

Lower triangular matrix element a_{ij} is stored in array element **ap**($i+(j-1) \times (2n-j)/2$).

Usage

LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

CHARACTER*1 equed, fact, uplo
INTEGER*4 info, ldb, ldx, n, nrhs
REAL*4 rcond
INTEGER*4 iwork(n)
REAL*4 afp((n*(n+1))/2), ap((n*(n+1))/2), b(ldb, nrhs),
 berr(nrhs), ferr(nrhs), s(n), work(3*n), x(ldx, nrhs)
CALL SPPSVX (fact, uplo, n, nrhs, ap, afp, equed, s, b, ldb, x,
 ldx, rcond, ferr, berr, work, iwork, info)

CHARACTER*1 equed, fact, uplo
INTEGER*4 info, ldb, ldx, n, nrhs
REAL*8 rcond
INTEGER*4 iwork(n)
REAL*8 afp((n*(n+1))/2), ap((n*(n+1))/2), b(ldb, nrhs),
 berr(nrhs), ferr(nrhs), s(n), work(3*n), x(ldx, nrhs)
CALL DPPSVX (fact, uplo, n, nrhs, ap, afp, equed, s, b, ldb, x,
 ldx, rcond, ferr, berr, work, iwork, info)

CHARACTER*1 equed, fact, uplo
INTEGER*4 info, ldb, ldx, n, nrhs
REAL*4 rcond
REAL*4 berr(nrhs), ferr(nrhs), rwork(n), s(n)
COMPLEX*8 afp((n*(n+1))/2), ap((n*(n+1))/2), b(ldb, nrhs),
work(2*n), x(ldx, nrhs)
CALL CPPSVX (fact, uplo, n, nrhs, ap, afp, equed, s, b, ldb, x,
ldx, rcond, ferr, berr, work, rwork, info)

CHARACTER*1 equed, fact, uplo
INTEGER*4 info, ldb, ldx, n, nrhs
REAL*8 rcond
REAL*8 berr(nrhs), ferr(nrhs), rwork(n), s(n)
COMPLEX*16 afp((n*(n+1))/2), ap((n*(n+1))/2), b(ldb, nrhs),
work(2*n), x(ldx, nrhs)
CALL ZPPSVX (fact, uplo, n, nrhs, ap, afp, equed, s, b, ldb, x,
ldx, rcond, ferr, berr, work, rwork, info)

LAPACK8, available on C Series and Exemplar architectures:

CHARACTER*1 equed, fact, uplo
INTEGER*8 info, ldb, ldx, n, nrhs
REAL*8 rcond
INTEGER*8 iwork(n)
REAL*8 afp((n*(n+1))/2), ap((n*(n+1))/2), b(ldb, nrhs),
berr(nrhs), ferr(nrhs), s(n), work(3*n), x(ldx, nrhs)
CALL SPPSVX (fact, uplo, n, nrhs, ap, afp, equed, s, b, ldb, x,
ldx, rcond, ferr, berr, work, iwork,
info)

CHARACTER*1 equed, fact, uplo
INTEGER*4 info, ldb, ldx, n, nrhs
REAL*8 rcond
REAL*8 berr(nrhs), ferr(nrhs), rwork(n), s(n)
COMPLEX*16 afp((n*(n+1))/2), ap((n*(n+1))/2), b(ldb, nrhs),
work(2*n), x(ldx, nrhs)
CALL CPPSVX (fact, uplo, n, nrhs, ap, afp, equed, s, b, ldb, x,
ldx, rcond, ferr, berr, work, rwork, info)

Input	fact	<p>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, as follows:</p> <p>fact = 'F' or 'f': afp contains the factored form of A. If equed = 'Y' or 'y', A was equilibrated before factoring and the scaling matrix is provided in s.</p> <p>fact = 'N' or 'n': The matrix A is copied to afp and factored.</p> <p>fact = 'E' or 'e': The matrix A is equilibrated, if necessary, then copied to afp and factored.</p>
	uplo	<p>Specifies whether the upper or lower triangular part of the symmetric or Hermitian matrix A is stored, as follows:</p> <p>uplo = 'U' or 'u': The upper triangular part of A is stored.</p> <p>uplo = 'L' or 'l': The lower triangular part of A is stored.</p>

	n	The number of linear equations, that is, the order of the matrix A . $n \geq 0$.
	nrhs	The number of right hand sides, that is, the number of columns of the matrix B . $nrhs \geq 0$.
	ap	The upper or lower triangle of the symmetric matrix A , packed columnwise in a linear array. The j -th column of A is stored in the array ap as follows: If uplo = 'U' or 'u', $ap(i + (j-1) \times j/2) = A(i, j)$ for $1 \leq i \leq j$. If uplo = 'L' or 'l', $ap(i + (j-1) \times (2n-j)/2) = A(i, j)$ for $j \leq i \leq n$.
	afp	If fact = 'F' or 'f', the triangular factor U or L from the Cholesky factorization $A = U*U$ or $A = LL^*$, in the same storage format as A .
	equed	If fact = 'F' or 'f', the type of equilibration, if any, that was done before factoring A on a previous call, as follows: equed = 'N' or 'n': No equilibration was done. equed = 'Y' or 'y': Equilibration was done. Not used as input if fact = 'N' or 'n' or 'E' or 'e'.
	s	The diagonal elements of the diagonal scaling matrix S if the matrix A was factored during a previous call (fact = 'F' or 'f'), and if equilibration was done during that call (equed = 'Y' or 'y'). $s(i) > 0, i = 1, 2, \dots, n$. Otherwise, not used as input.
	b	The right hand side vectors b for the system of linear equations.
	ldb	The leading dimension of array b in the calling program unit. $ldb \geq \max(1, n)$.
	ldx	The leading dimension of array x in the calling program unit. $ldx \geq \max(1, n)$.
Working Storage	work, iwork, rwork	Arrays used for work space.
Output	ap	If equed = 'Y', then A was overwritten by SAS . Not modified if fact = 'F' or 'f' or 'N' or 'n', or if fact = 'E' or 'e' and equed = 'N' or 'n' on exit.
	afp	If fact = 'N' or 'n' or 'E' or 'e', the triangular factor U or L from the Cholesky factorization $A = U*U$ or $A = LL^*$ of the matrix A , after equilibration, if performed.
	equed	If fact = 'E' or 'e', specifies the form of equilibration that was done, as follows: equed = 'N': No equilibration. equed = 'Y': Equilibration was done; A was replaced with SAS . Not used as output if fact = 'F' or 'f' or 'N' or 'n'.

s If **fact** = 'F' or 'f' and **equed** = 'Y' on exit, the diagonal elements of the diagonal scaling matrix S .

Destroyed if **fact** = 'F' or 'f' and **equed** = 'N' on exit.

Not used as output if **fact** = 'F' or 'f' or 'N' or 'n'.

b If **equed** = 'N', **b** is not modified; if **equed** = 'Y', B was overwritten by SB .

x On successful exit, the solution vectors X to the system of equations $AX = B$.

rcond On successful exit, the estimate of the reciprocal condition number of the matrix A , after equilibration, if performed. If **rcond** is small enough so that the logical expression

$$1.0 + \text{rcond} \text{ .EQ. } 1.0$$

is true, then A can be regarded as singular to working precision. This condition is indicated by a return code of **info** > 0, and the solution and error bounds are not computed.

ferr On successful exit, estimated forward error bounds for each solution vector. Let x_{true} and x represent column j of the true and computed solutions, respectively. Then **ferr**(j) is intended to bound $\|x - x_{true}\|_{\infty} / \|x\|_{\infty}$. The quality of the error bound depends on the quality of the computed estimate of $\|A^{-1}\|$ computed in the code; if the estimate is accurate, the error bound is valid.

berr On successful exit, **berr**(j) is the componentwise relative backward error of solution vector j (that is, the smallest relative change in any entry of A or column j of B that makes column j of X an exact solution).

info Status response:

info = 0: Successful exit.

info < 0: If **info** = $-k$, the k -th argument had an invalid value.

info > 0: If **info** = $k \leq n$, the leading minor of order k of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

If **info** = $n+1$, **rcond** is less than machine epsilon. The factorization has been completed, but the matrix A is singular to working precision, and the solution and error bounds have not been computed.

Notes

If you provide a previously factored matrix, the following output arguments from the **fact = 'N' or 'n' or 'E' or 'e'** call are input arguments to the **fact = 'F' or 'f'** call: **afp**, **equed**, and possibly **s**, depending on the value of **equed**.

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

fact \neq 'E' or 'e' or 'F' or 'f' or 'N' or 'n',
uplo \neq 'U' or 'u' or 'L' or 'l',
n < 0 ,
nrhs < 0 ,
fact = 'F' or 'f' and **equed** \neq 'N' or 'n' or 'Y' or 'y',
s is an input argument but contains a non-positive value,
ldb $< \max(1, n)$, and
ldx $< \max(1, n)$.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding, for example, the **uplo** argument as 'Lower' for 'L' or 'Upper' for 'U'.

Solve Positive Definite Tridiagonal Linear System**SPTSVX/.../ZPTSVX****Purpose**

These subprograms solve a system of linear equations $AX = B$, where A is an n -by- n real symmetric or complex Hermitian positive definite tridiagonal matrix and X and B are n -by- n matrices. A tridiagonal matrix $A = (a_{ij})$ is a matrix whose nonzero elements lie only on the principal diagonal ($i = j$), the subdiagonal ($i = j+1$), and the superdiagonal ($i = j-1$) of the matrix.

A real matrix is symmetric if $A = A^T$, its transpose; a complex matrix is Hermitian if $A = A^*$, its conjugate transpose.

A real symmetric matrix A is positive definite if the quadratic form $x^T Ax$ is positive for all nonzero real vectors x ; a complex Hermitian matrix A is positive definite if the quadratic form $x^* Ax$ is positive for all nonzero complex vectors x .

These subroutines perform the following:

1. If **fact** = 'N' or 'n', the matrix A is copied from arrays **d** and **e** to arrays **df** and **ef**, where it is factored as $A = LDL^*$, where L is a unit lower bidiagonal matrix and D is diagonal. The factorization can also be regarded as having the form $A = U^*DU$.
2. 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 epsilon, steps 3 and 4 are skipped.
3. The system of equations $AX = B$ is solved for X using the factored form of A .
4. Iterative refinement is applied to improve the computed solution vectors and calculate error bounds and backward error estimates.

Matrix Storage The following example illustrates the storage of symmetric or Hermitian tridiagonal matrices. Consider the following symmetric tridiagonal matrix of order $n = 7$:

11	21	0	0	0	0	0
21	22	32	0	0	0	0
0	32	33	43	0	0	0
0	0	43	44	54	0	0
0	0	0	54	55	65	0
0	0	0	0	65	66	76
0	0	0	0	0	76	77

The subdiagonal is stored in array **e** and the principal diagonal is stored in array **d**, as follows:

i	e (i)	d (i)
1	21	11
2	32	22
3	43	33
4	54	44
5	65	55
6	76	66
7		77

Usage

LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

CHARACTER*1 fact
INTEGER*4 info, ldb, ldx, n, nrhs
REAL*4 rcond
REAL*4 b(ldb, nrhs), berr(nrhs), d(n), df(n),
 e(n-1), ef(n-1), ferr(nrhs), work(2*n),
 x(ldx, nrhs)
CALL SPTSVX (fact, n, nrhs, d, e, df, ef, b, ldb, x, ldx,
 rcond, ferr, berr, work, info)

CHARACTER*1 fact
INTEGER*4 info, ldb, ldx, n, nrhs
REAL*8 rcond
REAL*8 b(ldb, nrhs), berr(nrhs), d(n), df(n),
 e(n-1), ef(n-1), ferr(nrhs), work(2*n),
 x(ldx, nrhs)
CALL DPTSVX (fact, n, nrhs, d, e, df, ef, b, ldb, x, ldx,
 rcond, ferr, berr, work, info)

CHARACTER*1 fact
INTEGER*4 info, ldb, ldx, n, nrhs
REAL*4 rcond
REAL*4 berr(nrhs), d(n), df(n), ferr(nrhs), rwork(n)
COMPLEX*8 b(ldb, nrhs), e(n-1), ef(n-1), work(n),
 x(ldx, nrhs)
CALL CPTSVX (fact, n, nrhs, d, e, df, ef, b, ldb, x, ldx,
 rcond, ferr, berr, work, rwork, info)

CHARACTER*1 fact
INTEGER*4 info, ldb, ldx, n, nrhs
REAL*8 rcond
REAL*8 berr(nrhs), d(n), df(n), ferr(nrhs), rwork(n)
COMPLEX*16 b(ldb, nrhs), e(n-1), ef(n-1), work(n),
 x(ldx, nrhs)
CALL ZPTSVX (fact, n, nrhs, d, e, df, ef, b, ldb, x, ldx,
 rcond, ferr, berr, work, rwork, info)

LAPACK8, available on C Series and Exemplar architectures:

CHARACTER*1 fact
INTEGER*8 info, ldb, ldx, n, nrhs
REAL*8 rcond
REAL*8 b(ldb, nrhs), berr(nrhs), d(n), df(n),
 e(n-1), ef(n-1), ferr(nrhs), work(2*n),
 x(ldx, nrhs)
CALL SPTSVX (fact, n, nrhs, d, e, df, ef, b, ldb, x, ldx,
 rcond, ferr, berr, work, info)

CHARACTER*1 fact
INTEGER*4 info, ldb, ldx, n, nrhs
REAL*8 rcond
REAL*8 berr(nrhs), d(n), df(n), ferr(nrhs), rwork(n)
COMPLEX*16 b(ldb, nrhs), e(n-1), ef(n-1), work(n),
 x(ldx, nrhs)
CALL CPTSVX (fact, n, nrhs, d, e, df, ef, b, ldb, x, ldx,
 rcond, ferr, berr, work, rwork, info)

Input	fact	Specifies whether or not the factored form of A has been supplied on entry, as follows: fact = 'F' or 'f': ef and df contain the factored form of A . fact = 'N' or 'n': The matrix is copied to ef and df and factored.
	n	The order of the matrix A . $n \geq 0$.
	nrhs	The number of right hand sides, that is, the number of columns of the matrix B . $nrhs \geq 0$.
	d	The n diagonal elements of the tridiagonal matrix A .
	e	The $n-1$ subdiagonal elements of the tridiagonal matrix A .
	df	If fact = 'F' or 'f', the n diagonal elements of the diagonal matrix D from the LDL^* factorization of A . Not used as input if fact = 'N' or 'n'.
	ef	If fact = 'F' or 'f', the $n-1$ subdiagonal elements of the unit bidiagonal factor L from the LDL^* factorization of A . Not used as input if fact = 'N' or 'n'.
	b	The n -by- $nrhs$ matrix of right hand side vectors for the system of equations $AX = B$.
	ldb	The leading dimension of array b in the calling program unit. $ldb \geq \max(1,n)$.
	ldx	The leading dimension of array x in the calling program unit. $ldx \geq \max(1,n)$.
Working Storage	work, rwork	Arrays used for work space.
Output	df	If fact = 'N' or 'n', the n diagonal elements of the diagonal matrix D from the LDL^* factorization of A . Not modified if fact = 'F' or 'f'.
	ef	If fact = 'N' or 'n', the $n-1$ subdiagonal elements of the unit bidiagonal factor L from the LDL^* factorization of A . Not modified if fact = 'F' or 'f'.
	x	On successful exit, the n -by- $nrhs$ matrix of solution vectors for the system of equations $AX = B$.
	rcond	On successful exit, the estimate of the reciprocal condition number of the matrix A . If rcond is small enough so that the logical expression

$$1.0 + rcond .EQ. 1.0$$

is true, then A can be regarded as singular to working precision. This condition is indicated by a return code of **info** > 0, and the solution and error bounds are not computed.

- ferr** On successful exit, estimated forward error bounds for each solution vector. Let x_{true} and x represent column j of the true and computed solutions, respectively. Then **ferr**(j) is intended to bound $\|x - x_{true}\|_{\infty} / \|x\|_{\infty}$. The quality of the error bound depends on the quality of the computed estimate of $\|A^{-1}\|$ computed in the code; if the estimate is accurate, the error bound is valid.
- berr** On successful exit, **berr**(j) is the componentwise relative backward error of solution vector j (that is, the smallest relative change in any entry of A or column j of B that makes column j of X an exact solution).
- info** Status response:
- info** = 0: Successful exit.
info < 0: If **info** = $-k$, the k -th argument had an invalid value
info > 0: If **info** = $k \leq n$, the leading minor of order k of A is not positive definite, so the factorization could not be completed unless **info** = n , and the solution has not been computed.
- If **info** = $n+1$, **rcond** is less than machine epsilon. The factorization has been completed, but the matrix A is singular to working precision, and the solution and error bounds have not been computed.

Notes If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

fact \neq 'F' or 'f' or 'N' or 'n',
n < 0,
nrhs < 0,
ldb < max(1,**n**), and
ldx < max(1,**n**).

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding the **fact** argument as 'Factored' for 'F' or 'Not Factored' for 'N'.

Solve Symmetric or Hermitian Packed Linear System**SSPSVX/...**

Purpose These subprograms solve a system of linear equations $AX = B$, where A is an n -by- n real or complex symmetric or complex Hermitian matrix stored in packed form and X and B are n -by- $nrhs$ matrices. If A is known to be positive definite, other subprograms in this chapter, specifically designed for positive definite matrices, will be more efficient than these subprograms.

A matrix is symmetric if $A = A^T$, its transpose; a matrix is Hermitian if $A = A^*$, its conjugate transpose.

The structure of A is indicated by the name of the subprogram used:

SSPSVX or DSPSVX A is a real symmetric matrix.
 CSPSVX or ZSPSVX A is a complex symmetric matrix.
 CHPSVX or ZHPSVX A is a complex Hermitian matrix.

These subroutines perform the following:

1. If **fact** = 'N' or 'n', the matrix A is copied from array **ap** to array **afp**, where the diagonal pivoting method is used to factor A as

$$A = UDU^T, \text{ if } \mathbf{uplo} = \text{'U' or 'u'}, \text{ or}$$

$$A = LDL^T, \text{ if } \mathbf{uplo} = \text{'L' or 'l'},$$

where U or L is a product of permutation and unit upper triangular or unit lower triangular matrices. D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks, and T indicates transpose.

2. 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 epsilon, steps 3 and 4 are skipped.
3. The system of equations $AX = B$ is solved for X using the factored form of A .
4. Iterative refinement is applied to improve the computed solution vectors and calculate error bounds and backward error estimates.

Matrix Storage Because either triangle of A may be obtained from the other, you need only provide one triangle of A , either the upper or the lower triangle. Compared to storing the entire matrix, you save memory by supplying that triangle stored column-by-column in packed form in a 1-dimensional array.

The following examples illustrate the packed storage of symmetric or Hermitian matrices.

Upper triangular storage. If the upper triangle of A is

11	12	13	14
	22	23	24
		33	34
			44

then A is packed column-by-column into an array \mathbf{ap} as follows:

k	1	2	3	4	5	6	7	8	9	10
$\mathbf{ap}(k)$	11	12	22	13	23	33	14	24	34	44

Upper triangular matrix element a_{ij} is stored in array element $\mathbf{ap}(i+j \times (j-1)/2)$.

Lower triangular storage. If the lower triangle of A is

11									
21	22								
31	32	33							
41	42	43	44						

then A is packed column-by-column into an array \mathbf{ap} as follows:

k	1	2	3	4	5	6	7	8	9	10
$\mathbf{ap}(k)$	11	21	31	41	22	32	42	33	43	44

Lower triangular matrix element a_{ij} is stored in array element $\mathbf{ap}(i+(j-1) \times (2n-j)/2)$.

Usage

LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

CHARACTER*1 fact, uplo
INTEGER*4 info, ldb, ldx, n, nrhs
REAL*4 rcond
INTEGER*4 ipiv(n), iwork(n)
REAL*4 afp((n*(n+1))/2), ap((n*(n+1))/2), b(ldb, nrhs),
 berr(nrhs), ferr(nrhs), work(3*n), x(ldx, nrhs)
CALL SSPSVX (fact, uplo, n, nrhs, ap, afp, ipiv, b, ldb, x,
 ldx, rcond, ferr, berr, work, iwork, info)

CHARACTER*1 fact, uplo
INTEGER*4 info, ldb, ldx, n, nrhs
REAL*8 rcond
INTEGER*4 ipiv(n), iwork(n)
REAL*8 afp((n*(n+1))/2), ap((n*(n+1))/2), b(ldb, nrhs),
 berr(nrhs), ferr(nrhs), work(3*n), x(ldx, nrhs)
CALL DSPSVX (fact, uplo, n, nrhs, ap, afp, ipiv, b, ldb, x,
 ldx, rcond, ferr, berr, work, iwork, info)

CHARACTER*1 fact, uplo
INTEGER*4 info, ldb, ldx, n, nrhs
REAL*4 rcond
INTEGER*4 ipiv(n)
REAL*4 berr(nrhs), ferr(nrhs), rwork(n)
COMPLEX*8 afp((n*(n+1))/2), ap((n*(n+1))/2), b(ldb, nrhs),
 work(2*n), x(ldx, nrhs)
CALL CHPSVX (fact, uplo, n, nrhs, ap, afp, ipiv, b, ldb, x,
 ldx, rcond, ferr, berr, work, rwork, info)

CHARACTER*1 fact, uplo
INTEGER*4 info, ldb, ldx, n, nrhs
REAL*4 rcond
INTEGER*4 ipiv(n)
REAL*4 berr(nrhs), ferr(nrhs), rwork(n)
COMPLEX*8 afp((n*(n+1))/2), ap((n*(n+1))/2), b(ldb, nrhs),
work(2*n), x(ldx, nrhs)
CALL CSPSVX (fact, uplo, n, nrhs, ap, afp, ipiv, b, ldb, x,
ldx, rcond, ferr, berr, work, rwork, info)

CHARACTER*1 fact, uplo
INTEGER*4 info, ldb, ldx, n, nrhs
REAL*8 rcond
INTEGER*4 ipiv(n)
REAL*8 berr(nrhs), ferr(nrhs), rwork(n)
COMPLEX*16 afp((n*(n+1))/2), ap((n*(n+1))/2), b(ldb, nrhs),
work(2*n), x(ldx, nrhs)
CALL ZHPSVX (fact, uplo, n, nrhs, ap, afp, ipiv, b, ldb, x,
ldx, rcond, ferr, berr, work, rwork, info)

CHARACTER*1 fact, uplo
INTEGER*4 info, ldb, ldx, n, nrhs
REAL*8 rcond
INTEGER*4 ipiv(n)
REAL*8 berr(nrhs), ferr(nrhs), rwork(n)
COMPLEX*16 afp((n*(n+1))/2), ap((n*(n+1))/2), b(ldb, nrhs),
work(2*n), x(ldx, nrhs)
CALL ZSPSVX (fact, uplo, n, nrhs, ap, afp, ipiv, b, ldb, x,
ldx, rcond, ferr, berr, work, rwork, info)

LAPACK8, available on C Series and Exemplar architectures:

CHARACTER*1 fact, uplo
INTEGER*8 info, ldb, ldx, n, nrhs
REAL*8 rcond
INTEGER*8 ipiv(n), iwork(n)
REAL*8 afp((n*(n+1))/2), ap((n*(n+1))/2), b(ldb, nrhs),
berr(nrhs), ferr(nrhs), work(3*n), x(ldx, nrhs)
CALL SSPSVX (fact, uplo, n, nrhs, ap, afp, ipiv, b, ldb, x, ldx,
rcond, ferr, berr, work, iwork, info)

CHARACTER*1 fact, uplo
INTEGER*8 info, ldb, ldx, n, nrhs
REAL*8 rcond
INTEGER*8 ipiv(n)
REAL*8 berr(nrhs), ferr(nrhs), rwork(n)
COMPLEX*16 afp((n*(n+1))/2), ap((n*(n+1))/2), b(ldb, nrhs),
work(2*n), x(ldx, nrhs)
CALL CHPSVX (fact, uplo, n, nrhs, ap, afp, ipiv, b, ldb, x,
ldx, rcond, ferr, berr, work, rwork, info)

CHARACTER*1 fact, uplo
INTEGER*8 info, ldb, ldx, n, nrhs
REAL*8 rcond
INTEGER*8 ipiv(n)
REAL*8 berr(nrhs), ferr(nrhs), rwork(n)
COMPLEX*16 apf((n*(n+1))/2), ap((n*(n+1))/2), b(ldb, nrhs),
work(2*n), x(ldx, nrhs)
CALL CSPSVX (fact, uplo, n, nrhs, ap, apf, ipiv, b, ldb, x,
ldx, rcond, ferr, berr, work, rwork, info)

Input

fact Specifies whether or not the factored form of A has been supplied on entry, as follows:

fact = 'F' or 'f': **apf** and **ipiv** contain the factored form of A .
fact = 'N' or 'n': The matrix A is copied to **apf** and factored.

uplo Specifies whether the upper or lower triangular part of the symmetric or Hermitian matrix A is stored, as follows:

uplo = 'U' or 'u': The upper triangular part of A is stored.
uplo = 'L' or 'l': The lower triangular part of A is stored.

n The number of linear equations, that is, the order of the matrix A . $n \geq 0$.

nrhs The number of right hand sides, that is, the number of columns of the matrix B . $nrhs \geq 0$.

ap The upper or lower triangle of the symmetric matrix A , packed columnwise in a linear array. The j -th column of A is stored in the array **ap** as follows:

If **uplo** = 'U' or 'u', $ap(i + (j-1) \times j/2) = A(i, j)$ for $1 \leq i \leq j$.
If **uplo** = 'L' or 'l', $ap(i + (j-1) \times (2n-j)/2) = A(i, j)$ for $j \leq i \leq n$.

apf If **fact** = 'F' or 'f', the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization $A = UDU^T$ or $A = LDL^T$ as computed by a previous call, stored as a packed triangular matrix in the same storage format as A .

ipiv If **fact** = 'F' or 'f', the details of the interchanges and the block structure of D , as computed by a previous call.

b The n -by- $nrhs$ matrix of right hand side vectors **b** for the system of equations $AX = B$.

ldb The leading dimension of array **b** in the calling program unit. $ldb \geq \max(1, n)$.

ldx The leading dimension of array **x** in the calling program unit. $ldx \geq \max(1, n)$.

Continued

SSPSVX/DSPSVX/CHPSVX/CSPSVX/ZHPSVX/ZSPSVX

Working Storage	work, iwork, rwork	Arrays used for work space.
Output	afp	If fact = 'N' or 'n', the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization $A = UDU^T$ or $A = LDL^T$, stored as a packed triangular matrix in the same storage format as A .
	ipiv	If fact = 'N' or 'n', the 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' or 'u' and ipiv (k) = ipiv ($k-1$) < 0, then rows and columns $k-1$ and $-\text{ipiv}(k)$ were interchanged, and $D(k-1:k, k-1:k)$ is a 2-by-2 diagonal block. If uplo = 'L' or 'l' and ipiv (k) = ipiv ($k+1$) < 0, then rows and columns $k+1$ and $-\text{ipiv}(k)$ were interchanged, and $D(k:k+1, k:k+1)$ is a 2-by-2 diagonal block.
	x	On successful exit, the n -by- nrhs matrix of solution vectors for the system of equations $AX = B$.
	rcond	On successful exit, the estimate of the reciprocal condition number of the matrix A . If rcond is small enough so that the logical expression $1.0 + \text{rcond} \text{ .EQ. } 1.0$ is true, then A can be regarded as singular to working precision. This condition is indicated by a return code of info > 0, and the solution and error bounds are not computed.
	ferr	On successful exit, estimated forward error bounds for each solution vector. Let x_{true} and x represent column j of the true and computed solutions, respectively. Then ferr (j) is intended to bound $\ x - x_{true}\ _{\infty} / \ x\ _{\infty}$. The quality of the error bound depends on the quality of the computed estimate of $\ A^{-1}\ $ computed in the code; if the estimate is accurate, the error bound is valid.
	berr	On successful exit, berr (j) is the componentwise relative backward error of solution vector j (that is, the smallest relative change in any entry of A or column j of B that makes column j of X an exact solution).
	info	Status response: info = 0: Successful exit. info < 0: If info = $-k$, the k -th argument had an invalid value. info > 0: If info = k , $D(k, k)$ is zero. The factorization has been completed, but the block diagonal matrix D is singular, so the solution and error bounds could not be computed. If info = $n+1$, The block diagonal matrix D is nonsingular, but rcond is less than machine epsilon. The factorization has been completed, but the matrix is singular to working precision, so the solution and error bounds have not been computed.

Notes

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

fact \neq 'F' or 'f' or 'N' or 'n',
uplo \neq 'U' or 'u' or 'L' or 'l',
n < 0 ,
nrhs < 0 ,
ldb $< \max(1, n)$, and
ldx $< \max(1, n)$.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding, for example, the **uplo** argument as 'Lower' for 'L' or 'Upper' for 'U'.

Solve Symmetric Linear System**SSYSVX/DSYSVX/.../ZSYSVX****Purpose**

These subprograms solve a system of linear equations $AX = B$, where A is an n -by- n real or complex symmetric or complex Hermitian matrix stored in packed form and X and B are n -by- $nrhs$ matrices. If A is known to be positive definite, other subprograms in this chapter, specifically designed for positive definite matrices, will be more efficient than these subprograms.

A matrix is symmetric if $A = A^T$, its transpose; a matrix is Hermitian if $A = A^*$, its conjugate transpose.

The structure of A is indicated by the name of the subprogram used:

SSYSVX or DSYSVX A is a real symmetric matrix.
 CSYSVX or ZSYSVX A is a complex symmetric matrix.
 CHESVX or ZHESVX A is a complex Hermitian matrix.

These subprograms perform the following:

1. If **fact** = 'N' or 'n', the matrix A is copied from array **a** to array **af**, where the diagonal pivoting method is used to factor A as

$$A = UDU^*, \text{ if } \mathbf{uplo} = \text{'U'} \text{ or 'u'}, \text{ or}$$

$$A = LDL^*, \text{ if } \mathbf{uplo} = \text{'L'} \text{ or 'l'},$$

where U or L is a product of permutation and unit upper triangular or unit lower triangular matrices, D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks, and $*$ indicates transpose in the symmetric cases and conjugate transpose in the Hermitian cases.

2. 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 epsilon, steps 3 and 4 are skipped.
3. The system of equations $AX = B$ is solved for X using the factored form of A .
4. Iterative refinement is applied to improve the computed solution vectors and calculate error bounds and backward error estimates.

Matrix Storage Because either triangle of A may be obtained from the other, you need only provide one triangle of A . You may supply either the upper or the lower triangle of A , in a two-dimensional array large enough to hold the entire matrix. The other triangle of the array is not referenced.

Usage

LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```

CHARACTER*1 fact, uplo
INTEGER*4 info, lda, ldaf, ldb, ldx, lwork, n, nrhs
REAL*4 rcond
INTEGER*4 ipiv(n), iwork(n)
REAL*4 a(lda, n), af(ldaf, n), b(ldb, nrhs),
  berr(nrhs), ferr(nrhs), work(lwork), x(ldx, nrhs)
CALL SSYSVX (fact, uplo, n, nrhs, a, lda, af, ldaf, ipiv, b,
  ldb, x, ldx, rcond, ferr, berr, work, lwork,
  iwork, info)

```

CHARACTER*1 fact, uplo
INTEGER*4 info, lda, ldaf, ldb, ldx, lwork, n, nrhs
REAL*8 rcond
INTEGER*4 ipiv(n), iwork(n)
REAL*8 a(lda, n), af(ldaf, n), b(ldb, nrhs),
 berr(nrhs), ferr(nrhs), work(lwork), x(ldx, nrhs)
CALL DSYVX (fact, uplo, n, nrhs, a, lda, af, ldaf, ipiv, b,
 ldb, x, ldx, rcond, ferr, berr, work, lwork,
 iwork, info)

CHARACTER*1 fact, uplo
INTEGER*4 info, lda, ldaf, ldb, ldx, lwork, n, nrhs
REAL*4 rcond
INTEGER*4 ipiv(n)
REAL*4 berr(nrhs), ferr(nrhs), rwork(n)
COMPLEX*8 a(lda, n), af(ldaf, n), b(ldb, nrhs),
 work(lwork), x(ldx, nrhs)
CALL CHESVX (fact, uplo, n, nrhs, a, lda, af, ldaf, ipiv, b,
 ldb, x, ldx, rcond, ferr, berr, work, lwork,
 rwork, info)

CHARACTER*1 fact, uplo
INTEGER*4 info, lda, ldaf, ldb, ldx, lwork, n, nrhs
REAL*4 rcond
INTEGER*4 ipiv(n)
REAL*4 berr(nrhs), ferr(nrhs), rwork(n)
COMPLEX*8 a(lda, n), af(ldaf, n), b(ldb, nrhs),
 work(lwork), x(ldx, nrhs)
CALL CSYSVX (fact, uplo, n, nrhs, a, lda, af, ldaf, ipiv, b,
 ldb, x, ldx, rcond, ferr, berr, work, lwork,
 rwork, info)

CHARACTER*1 fact, uplo
INTEGER*4 info, lda, ldaf, ldb, ldx, lwork, n, nrhs
REAL*8 rcond
INTEGER*4 ipiv(n)
REAL*8 berr(nrhs), ferr(nrhs), rwork(n)
COMPLEX*16 a(lda, n), af(ldaf, n), b(ldb, nrhs),
 work(lwork), x(ldx, nrhs)
CALL ZHESVX (fact, uplo, n, nrhs, a, lda, af, ldaf, ipiv, b,
 ldb, x, ldx, rcond, ferr, berr, work, lwork,
 rwork, info)

CHARACTER*1 fact, uplo
INTEGER*4 info, lda, ldaf, ldb, ldx, lwork, n, nrhs
REAL*8 rcond
INTEGER*4 ipiv(n)
REAL*8 berr(nrhs), ferr(nrhs), rwork(n)
COMPLEX*16 a(lda, n), af(ldaf, n), b(ldb, nrhs),
 work(lwork), x(ldx, nrhs)
CALL ZSYSVX (fact, uplo, n, nrhs, a, lda, af, ldaf, ipiv, b,
 ldb, x, ldx, rcond, ferr, berr, work, lwork,
 rwork, info)

LAPACK8, available on C Series and Exemplar architectures:

```

CHARACTER*1 fact, uplo
INTEGER*8    info, lda, ldaf, ldb, ldx, lwork, n, nrhs
REAL*8      rcond
INTEGER*8    ipiv(n), iwork(n)
REAL*8      a(lda, n), af(ldaf, n), b(ldb, nrhs),
             berr(nrhs), ferr(nrhs), work(lwork), x(ldx, nrhs)
CALL SSYSVX (fact, uplo, n, nrhs, a, lda, af, ldaf, ipiv, b,
             ldb, x, ldx, rcond, ferr, berr, work, lwork,
             iwork, info)

```

```

CHARACTER*1 fact, uplo
INTEGER*8    info, lda, ldaf, ldb, ldx, lwork, n, nrhs
REAL*8      rcond
INTEGER*8    ipiv(n)
REAL*8      berr(nrhs), ferr(nrhs), rwork(n)
COMPLEX*16  a(lda, n), af(ldaf, n), b(ldb, nrhs),
             work(lwork), x(ldx, nrhs)
CALL CHESVX (fact, uplo, n, nrhs, a, lda, af, ldaf, ipiv, b,
             ldb, x, ldx, rcond, ferr, berr, work, lwork,
             rwork, info)

```

```

CHARACTER*1 fact, uplo
INTEGER*8    info, lda, ldaf, ldb, ldx, lwork, n, nrhs
REAL*8      rcond
INTEGER*8    ipiv(n)
REAL*8      berr(nrhs), ferr(nrhs), rwork(n)
COMPLEX*16  a(lda, n), af(ldaf, n), b(ldb, nrhs),
             work(lwork), x(ldx, nrhs)
CALL CSYSVX (fact, uplo, n, nrhs, a, lda, af, ldaf, ipiv, b,
             ldb, x, ldx, rcond, ferr, berr, work, lwork,
             rwork, info)

```

Input	fact	Specifies whether or not the factored form of A has been supplied on entry, as follows: fact = 'F' or 'f': af and ipiv contain the factored form of A . fact = 'N' or 'n': The matrix A will be copied to af and factored.
	uplo	Specifies whether the upper or lower triangular part of the symmetric or Hermitian matrix A is stored, as follows: uplo = 'U' or 'u': The upper triangular part of A is stored. uplo = 'L' or 'l': The lower triangular part of A is stored.
	n	The number of linear equations, that is, the order of the matrix A . $n \geq 0$.
	nrhs	The number of right hand sides, that is, the number of columns of the matrix B . $nrhs \geq 0$.

	a	The symmetric or Hermitian matrix a . If uplo = 'U' or 'u', the leading n -by- n upper triangular part of a contains the upper triangular part of the matrix <i>A</i> , and the strictly lower triangular part of a is not referenced. If uplo = 'L' or 'l', the leading n -by- n lower triangular part of a contains the lower triangular part of the matrix <i>A</i> , and the strictly upper triangular part of a is not referenced.
	lda	The leading dimension of array a in the calling program unit. $lda \geq \max(1,n)$.
	af	If fact = 'F' or 'f', the block diagonal matrix <i>D</i> and the multipliers used to obtain the factor <i>U</i> or <i>L</i> from the factorization $A=UDU^*$ or $A=LDL^*$. Not used as input if fact = 'N' or 'n'.
	ldaf	The leading dimension of array af in the calling program unit. $ldaf \geq \max(1,n)$.
	ipiv	If fact = 'F' or 'f', the details of the interchanges and the block structure of <i>D</i> . If $ipiv(k) > 0$, then rows and columns <i>k</i> and $ipiv(k)$ were interchanged and $D(k,k)$ is a 1-by-1 diagonal block. If $uplo = 'U'$ or 'u' and $ipiv(k) = ipiv(k-1) < 0$, then rows and columns <i>k</i> -1 and $-ipiv(k)$ were interchanged and $D(k-1:k, k-1:k)$ is a 2-by-2 diagonal block. If $uplo = 'L'$ or 'l' and $ipiv(k) = ipiv(k+1) < 0$, then rows and columns <i>k</i> +1 and $-ipiv(k)$ were interchanged and $D(k:k+1, k:k+1)$ is a 2-by-2 diagonal block. Not used as input if fact = 'N' or 'n'.
	b	The n -by- nrhs matrix of right hand side vectors b for the system of equations $AX = B$.
	ldb	The leading dimension of array b in the calling program unit. $ldb \geq \max(1,n)$.
	ldx	The leading dimension of array x in the calling program unit. $ldx \geq \max(1,n)$.
	lwork	The length of array work . $lwork \geq \max(1,3n)$ for SSYSVX and DSYSVX; $lwork \geq \max(1,2n)$ for CHESVX, CSYSVX, ZHESVX, and DSYSVX. For good performance, lwork must generally be larger. The optimum value of lwork for high performance is returned in work(1) .
Working Storage	work, iwork, rwork	Arrays used for work space. On successful exit, work(1) contains the optimal work space length lwork for high performance.
Output	af	If fact = 'N' or 'n', the block diagonal matrix <i>D</i> and the multipliers used to obtain the factor <i>U</i> or <i>L</i> from the factorization $A = UDU^*$ or $A = LDL^*$. Not used as output if fact = 'F' or 'f'.

- ipiv** If **fact** = 'N' or 'n', the 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' or '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' or '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.
- Not used as output if **fact** = 'F' or 'f'.
- x** On successful exit, the n -by- $nrhs$ matrix of solution vectors for the system of equations $AX = B$.
- rcond** On successful exit, the estimate of the reciprocal condition number of the matrix A . If **rcond** is small enough so that the logical expression
- $$1.0 + rcond .EQ. 1.0$$
- is true, then A can be regarded as singular to working precision. This condition is indicated by a return code of **info** > 0, and the solution and error bounds are not computed.
- ferr** On successful exit, estimated forward error bounds for each solution vector. Let x_{true} and x represent column j of the true and computed solutions, respectively. Then **ferr**(j) is intended to bound $\|x - x_{true}\|_{\infty} / \|x\|_{\infty}$. The quality of the error bound depends on the quality of the computed estimate of $\|A^{-1}\|$ computed in the code; if the estimate is accurate, the error bound is valid.
- berr** On successful exit, **berr**(j) is the componentwise relative backward error of solution vector j (that is, the smallest relative change in any entry of A or column j of B that makes column j of X an exact solution).
- info** Status response
- info** = 0: Successful exit.
- info** < 0: If **info** = $-k$, the k -th argument had an invalid value.
- info** > 0: If **info** $\leq n$, $D(k, k)$ is zero. The factorization has been completed, but the block diagonal matrix D is singular, so the solution and error bounds could not be computed. If **info** = $n+1$ the block diagonal matrix D is nonsingular, but **rcond** is less than machine epsilon. The factorization has been completed, but the matrix is singular to working precision, so the solution and error bounds have not been computed.

Notes

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

fact \neq 'F' or 'f' or 'N' or 'n',
uplo \neq 'U' or 'u' or 'L' or 'l',
n < 0 ,
nrhs < 0 ,
lda $< \max(1, n)$,
ldaf $< \max(1, n)$,
ldb $< \max(1, n)$,
ldx $< \max(1, n)$, and
lwork too small.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding, for example, the **uplo** argument as 'Lower' for 'L' or 'Upper' for 'U'.

Computational Subprograms for Linear Equations

Overview

This chapter describes some of the computational subprograms to solve systems of linear equations. Although software is included for all LAPACK computational subprograms for linear equations, not all of them are described in this chapter.

This chapter explains how to use LAPACK subprograms to solve systems of linear equations or calculate the inverse of a matrix.

These operations are performed for a variety of types of matrices, including:

- real and complex general full matrices
- real and complex general band matrices
- real symmetric and complex Hermitian positive definite full matrices
- real symmetric and complex Hermitian positive definite band matrices
- real and complex general tridiagonal matrices
- real symmetric and complex Hermitian positive definite tridiagonal matrices
- real and complex symmetric and complex Hermitian indefinite matrices

Chapter Objectives

After you read this chapter you will:

- understand the role of the condition number in solving linear equations
- know how to compute the inverse of a matrix
- know when not to compute the inverse of a matrix
- know how to use the described subprograms

What You Need to Know to Use These Subprograms

The LAPACK subprograms in this chapter are organized so that it is usually necessary to call two or more subprograms to perform the above operations. This division of labor significantly enhances the number of processing options such as matrix factoring, condition number estimation, solving, and computing an inverse matrix that you may apply to a specific problem to obtain a suitable solution. It also allows you the flexibility to choose between subprograms that are fast but use a less reliable, elementary test for singularity and subprograms that are slightly slower but use a significantly more reliable test involving an estimate of the condition number of the coefficient matrix.

Condition Number

The condition number, $\kappa(A)$, of the coefficient matrix A measures the sensitivity of the solution x of the system of linear equations $Ax = b$ to errors in the matrix A and the right hand side b . Under reasonable assumptions, if δA and δb represent the errors in A and b , respectively, and if $\| \cdot \|$ represents any vector norm and its subordinate matrix norm, the error δx in x that results from solving $(A + \delta A)(x + \delta x) = b + \delta b$ instead of $Ax = b$ is bounded by

$$\frac{\|\delta x\|}{\|x\|} \leq \frac{\kappa(A)}{1 - \|A^{-1}\| \|\delta A\|} \left(\frac{\|\delta A\|}{\|A\|} + \frac{\|\delta b\|}{\|b\|} \right).$$

A standard result of numerical analysis shows that the roundoff error introduced by the solution process may be modeled by taking $\|\delta A\|/\|A\|$ and $\|\delta b\|/\|b\|$ to be small multiples of the computer's machine epsilon. Computational singularity of A results in $\kappa(A) = \infty$. A more common situation occurs when A is not numerically singular but is ill-conditioned. When a matrix is ill-conditioned, $\kappa(A)$ is large, so small errors in the matrix and right hand side and small roundoff errors introduced during the solution process itself may be magnified greatly in the solution.

Since $1 < \kappa(A) \leq \infty$, it is more convenient to compute the reciprocal condition number, $1/\kappa(A)$, than $\kappa(A)$ itself. Roughly speaking, the reciprocal condition number has the interpretation that if $1/\kappa(A)$ is about 10^{-d} , elements of x can be expected to have about d fewer significant digits of accuracy than the elements of A or b . Consequently, if errors in the coefficient matrix and right hand side exceed $1/\kappa(A)$, or if $1/\kappa(A)$ is negligible compared to 1.0, then x may have no significant digits at all.

Matrix Inversion

Subprograms for computing the inverse of a matrix are provided, although it is almost never necessary to compute one. While papers and reference books extensively use the notation " $A^{-1}b$ " to mean "the solution x of the system of linear equations $Ax = b$," it is almost always more efficient and more accurate to compute the solution directly than to invert the coefficient matrix and multiply the inverse times the right hand side vector. This is true even if there are many systems of equations, all using the same coefficient matrix; the matrix may be factored once and the systems may be solved from the factors about as efficiently—and more accurately—than by matrix multiplication by the inverse.

Combining Computational Subprograms

When you use the computational subprograms instead of calling the simple or expert drivers, you usually must put two or more of them together to carry out your entire computation. This section shows several ways to assemble an algorithm from several computational subprograms. In these examples, we assume your matrix is a 5-by-5 real general matrix stored in a single precision array big enough to handle a 10-by-10 problem. We do not show how the matrix, or the right hand side, if needed, is generated. The examples generalize for other matrix formats, when the appropriate subprograms exist. However, since the inverse of a band matrix A generally is a full matrix, and therefore would not fit in the band storage for A , no direct provision is made for computing A^{-1} for band matrices. Thus, these examples do not generalize to computing the inverse of a band matrix.

Solving Linear Equations with a Simple Singularity Test

The following code segment shows how to solve a system of linear equations $Ax = b$, basing the test for matrix singularity on the generation of an exact zero pivot during matrix factorization. SGETRF computes the LU factorization of the coefficient matrix A . If no exact zero pivots occurred, then SGETRS computes the solution vector x , overwriting the right hand side vector B with it; otherwise, the matrix is singular.

```

INTEGER*4 INFO, LDA, LDB, N, NMAX, NRHS
PARAMETER ( NMAX = 10 )
PARAMETER ( LDA = NMAX )
PARAMETER ( LDB = NMAX )
INTEGER*4 IPIV(NMAX)
REAL*4 A(LDA,NMAX), B(LDB)

N = 5
NRHS = 1
CALL SGETRF (N, N, A, LDA, IPIV, INFO)
IF ( INFO .EQ. 0 ) THEN
    CALL SGETRS ('NotTransposed', N, NRHS, A, LDA, IPIV, B, LDB, INFO)
ELSE
    handle singular matrix
ENDIF

```

Solving Linear Equations with a Condition Number Singularity Test

The following code segment shows how to solve a system of linear equations, basing the test for matrix singularity on the condition number of the matrix. SLANGE is used to compute $\|A\|_{\infty}$. SGETRF computes the LU factorization of A . If SGETRF indicates that an exact zero pivot occurred, then RCOND is set to zero; otherwise, SGECON is called to estimate the condition number. Finally, if RCOND is significant compared to 1.0, SGETRS is called to compute the solution; otherwise, the matrix is computationally singular.

```

INTEGER*4 INFO, LDA, LDB, N, NMAX, NRHS
REAL*4 ANORM, RCOND, SLANGE
PARAMETER ( NMAX = 10 )
PARAMETER ( LDA = NMAX )
PARAMETER ( LDB = NMAX )
INTEGER*4 IPIV(NMAX), IWORK(NMAX)
REAL*4 A(LDA,NMAX), B(LDB), WORK(4*NMAX)

N = 5
NRHS = 1
ANORM = SLANGE ('InfinityNorm', N, N, A, LDA, WORK)
CALL SGETRF (N, N, A, LDA, IPIV, INFO)
IF ( INFO .NE. 0 ) THEN
    RCOND = 0.0
ELSE
    CALL SGECON ('InfinityNorm', N, A, LDA, ANORM, RCOND, WORK,
&                IWORK, INFO)
ENDIF
IF ( RCOND + 1.0 .NE. 1.0 ) THEN
    CALL SGETRS ('NotTransposed', N, NRHS, A, LDA, IPIV, B, LDB, INFO)
ELSE
    handle singular matrix
ENDIF

```

Inverting a Matrix with a Simple Singularity Test

Notwithstanding the previous advice not to invert your matrix, here is one way to do it in those unusual situations where the inverse really is required. SGETRF computes the *LU* factorization of *A*. If no zeros occurred on the diagonal of *U*, then SGETRI computes the inverse matrix, overwriting the *LU* factorization with it. If either SGETRF or SGETRI returns a nonzero status response, the matrix is singular.

```

INTEGER*4 INFO, LDA, LWORK, N, NMAX
PARAMETER ( NMAX = 10 )
PARAMETER ( LDA = NMAX )
PARAMETER ( LWORK = 5 * NMAX )
INTEGER*4 IPIV(LDA)
REAL*4     A(LDA,NMAX), WORK(LWORK)

N = 5
CALL SGETRF (N, N, A, LDA, IPIV, INFO)
IF ( INFO.EQ. 0 ) THEN
    CALL SGETRI (N, A, LDA, IPIV, WORK, LWORK, INFO)
ENDIF
IF ( INFO.NE. 0 ) THEN
    handle singular matrix
ENDIF

```

Inverting a Matrix with a Condition Number Singularity Test

Here is another way to compute the inverse of a matrix in those unusual situations where the inverse really is required, this time basing the singularity test on an estimate of the condition number. SLANGE is used to compute $\|A\|_{\infty}$. SGETRF computes the *LU* factorization of *A*. If SGETRF indicates that an exact zero pivot occurred, then RCOND is set to zero; otherwise, SGECON is called to estimate the condition number. Finally, if RCOND is significant compared to 1.0, then SGETRI computes the inverse matrix, overwriting the *LU* factorization with it; otherwise, the matrix is computationally singular.

```

INTEGER*4 INFO, LDA, LWORK, N, NMAX
REAL*4     ANORM, RCOND, SLANGE
PARAMETER ( NMAX = 10 )
PARAMETER ( LDA = NMAX )
PARAMETER ( LWORK = 5 * NMAX )
INTEGER*4 IPIV(LDA), IWORK(NMAX)
REAL*4     A(LDA,NMAX), WORK(LWORK)

N = 5
ANORM = SLANGE ('InfinityNorm', N, N, A, LDA, WORK)
CALL SGETRF (N, N, A, LDA, IPIV, INFO)
IF ( INFO.NE. 0 ) THEN
    RCOND = 0.0
ELSE
    CALL SGECON ('InfinityNorm', N, A, LDA, ANORM, RCOND, WORK,
&              IWORK, INFO)
ENDIF
IF ( RCOND + 1.0.NE. 1.0 ) THEN
    CALL SGETRI (N, A, LDA, IPIV, WORK, LWORK, INFO)
ELSE
    handle singular matrix
ENDIF

```

Supplemental Reading

- Anderson, E. *et al.* *LAPACK Users' Guide*, Second Edition. Philadelphia, PA: Society for Industrial and Applied Mathematics. 1995.
- Forsythe, G. and C.B. Moler. *Computer Solution of Linear Algebraic Systems*. Englewood Cliffs, NJ: Prentice-Hall, Inc. 1967.
- Golub, G.H. and C.F. Van Loan. *Matrix Computations*, 2nd Edition. Baltimore, MD: The Johns Hopkins University Press. 1989.

Subprogram Descriptions

Estimate the Condition Number of a General Band Matrix SGBCON, DGBCON, CGBCON, ZGBCON	4-8
Factor a General Band Matrix SGBTRF, DGBTRF, CGBTRF, ZGBTRF	4-11
Solve a General Band Linear System SGBTRS, DGBTRS, CGBTRS, ZGBTRS	4-14
Estimate the Condition Number of a General Full Matrix SGECON, DGECON, CGECON, ZGECON	4-16
Factor a General Full Matrix SGETRF, DGETRF, CGETRF, ZGETRF	4-18
Invert a General Full Matrix SGETRI, DGETRI, CGETRI, ZGETRI	4-20
Solve a General Full Linear System SGETRS, DGETRS, CGETRS, ZGETRS	4-22
Estimate the Condition Number of a General Tridiagonal Matrix SGTCON, DGTCON, CGTCON, ZGTCON	4-24
Factor a General Tridiagonal Matrix SGTTRF, DGTTRF, CGTTRF, ZGTTRF	4-27
Solve a General Tridiagonal Linear System SGTTRS, DGTTRS, CGTTRS, ZGTTRS	4-29
Estimate the Condition Number of a Positive Definite Band Matrix SPBCON, DPBCON, CPBCON, ZPBCON	4-31
Factor a Positive Definite Band Matrix SPBTRF, DPBTRF, CPBTRF, ZPBTRF	4-34
Solve a Positive Definite Band Linear System SPBTRS, DPBTRS, CPBTRS, ZPBTRS	4-37
Estimate the Condition Number of a Positive Definite Full Matrix SPOCON, DPOCON, CPOCON, ZPOCON	4-39
Factor a Positive Definite Full Matrix SPOTRF, DPOTRF, CPOTRF, ZPOTRF	4-41
Invert a Positive Definite Full Matrix SPOTRI, DPOTRI, CPOTRI, ZPOTRI	4-43
Solve a Positive Definite Full Linear System SPOTRS, DPOTRS, CPOTRS, ZPOTRS	4-45

Estimate the Condition Number of a Positive Definite Matrix Stored in Packed Form SPPCON, DPPCON, CPPCON, ZPPCON	4-47
Factor a Positive Definite Matrix Stored in Packed Form SPPTRF, DPPTRF, CPPTRF, ZPPTRF	4-49
Invert a Positive Definite Matrix Stored in Packed Form SPPTRI, DPPTRI, CPPTRI, ZPPTRI	4-52
Solve a Positive Definite Packed Linear System SPPTRS, DPPTRS, CPPTRS, ZPPTRS	4-55
Estimate the Condition Number of a Positive Definite Tridiagonal Matrix SPTCON, DPTCON, CPTCON, ZPTCON	4-57
Factor a Positive Definite Tridiagonal Matrix SPTTRF, DPTTRF, CPTTRF, ZPTTRF	4-59
Solve a Positive Definite Tridiagonal Linear System SPTTRS, DPTTRS, CPTTRS, ZPTTRS	4-61
Estimate the Condition Number of a Symmetric or Hermitian Matrix Stored in Packed Form SSPCON, DSPCON, CHPCON, CSPCON, ZHPCON, ZSPCON	4-63
Factor a Symmetric or Hermitian Matrix Stored in Packed Form SSPTRF, DSPTRF, CHPTRF, CSPTRF, ZHPTRF, ZSPTRF	4-66
Invert a Symmetric or Hermitian Matrix Stored in Packed Form SSPTRI, DSPTRI, CHPTRI, CSPTRI, ZHPTRI, ZSPTRI	4-70
Solve a Symmetric or Hermitian Packed Linear System SSPTRS, DSPTRS, CHPTRS, CSPTRS, ZHPTRS, ZSPTRS	4-73
Estimate the Condition Number of a Symmetric or Hermitian Matrix SSYCON, DSYCON, CHECON, CSYCON, ZHECON, ZSYCON	4-75
Factor a Symmetric or Hermitian Matrix SSYTRF, DSYTRF, CHETRF, CSYTRF, ZHETRF, ZSYTRF	4-78
Invert a Symmetric or Hermitian Matrix SSYTRI, DSYTRI, CHETRI, CSYTRI, ZHETRI, ZSYTRI	4-81
Solve a Symmetric or Hermitian Linear System SSYTRS, DSYTRS, CHETRS, CSYTRS, ZHETRS, ZSYTRS	4-84
Estimate the Condition Number of a Triangular Band Matrix STBCON, DTBCON, CTBCON, ZTBCON	4-87
Solve a Triangular Band Linear System STBTRS, DTBTRS, CTBTRS, ZTBTRS	4-91
Estimate the Condition Number of a Triangular Matrix Stored in Packed Form STPCON, DTPCON, CTPCON, ZTPCON	4-94
Invert a Triangular Matrix Stored in Packed Form STPTRI, DTPTRI, CTPTRI, ZTPTRI	4-97
Solve a Triangular Packed Linear System STPTRS, DTPTRS, CTPTRS, ZTPTRS	4-100
Estimate the Condition Number of a Triangular Full Matrix STRCON, DTRCON, CTRCON, ZTRCON	4-103
Invert a Triangular Full Matrix STRTRI, DTRTRI, CTRTRI, ZTRTRI	4-106
Solve a Triangular Full Linear System STRTRS, DTRTRS, CTRTRS, ZTRTRS	4-108

Subprograms not in this Guide

Table 4-1 4-110

Purpose These subprograms estimate the reciprocal of the condition number of a general band matrix A , in either the 1-norm or the ∞ -norm, using the LU factorization computed by `_GBTRF`.

An estimate is obtained for $\|A^{-1}\|$, and the reciprocal of the condition number is computed as $\mathbf{rcond} = (\|A\| \|A^{-1}\|)^{-1}$.

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```

CHARACTER*1 norm
INTEGER*4 info, kl, ku, ldab, n
REAL*4 anorm, rcond
INTEGER*4 ipiv(n), iwork(n)
REAL*4 ab(ldab, n), work(3*n)
CALL SGBCON (norm, n, kl, ku, ab, ldab, ipiv, anorm, rcond,
work, iwork, info)

```

```

CHARACTER*1 norm
INTEGER*4 info, kl, ku, ldab, n
REAL*8 anorm, rcond
INTEGER*4 ipiv(n), iwork(n)
REAL*8 ab(ldab, n), work(3*n)
CALL DGBCON (norm, n, kl, ku, ab, ldab, ipiv, anorm, rcond,
work, iwork, info)

```

```

CHARACTER*1 norm
INTEGER*4 info, kl, ku, ldab, n
REAL*4 anorm, rcond
INTEGER*4 ipiv(n)
REAL*4 rwork(n)
COMPLEX*8 ab(ldab, n), work(2*n)
CALL CGBCON (norm, n, kl, ku, ab, ldab, ipiv, anorm, rcond,
work, rwork, info)

```

```

CHARACTER*1 norm
INTEGER*4 info, kl, ku, ldab, n
REAL*8 anorm, rcond
INTEGER*4 ipiv(n)
REAL*8 rwork(n)
COMPLEX*16 ab(ldab, n), work(2*n)
CALL ZGBCON (norm, n, kl, ku, ab, ldab, ipiv, anorm, rcond,
work, rwork, info)

```

LAPACK8, available on C Series and Exemplar architectures:

```

CHARACTER*1 norm
INTEGER*8 info, kl, ku, ldab, n
REAL*8 anorm, rcond
INTEGER*8 ipiv(n), iwork(n)
REAL*8 ab(ldab, n), work(3*n)
CALL SGBCON (norm, n, kl, ku, ab, ldab, ipiv, anorm, rcond,
work, iwork, info)

```

CHARACTER*1 norm
INTEGER*8 info, kl, ku, ldab, n
REAL*8 anorm, rcond
INTEGER*8 ipiv(n)
REAL*8 rwork(n)
COMPLEX*16 ab(ldab, n), work(2*n)
CALL CGBCON (norm, n, kl, ku, ab, ldab, ipiv, anorm, rcond,
work, rwork, info)

Input

norm Specifies whether to use the 1-norm or the ∞ -norm to estimate the condition number, as follows, as follows:

norm = '1', 'O', or 'o': Use $\| \|_1$.

norm = 'I' or 'i': Use $\| \|_\infty$.

n The order of the matrix A . $n \geq 0$.

kl The number of subdiagonals within the band of A . $kl \geq 0$.

ku The number of superdiagonals within the band of A . $ku \geq 0$.

ab Details of the LU factorization of the band matrix A , as computed by `_GBTRF`. U is an upper triangular band matrix with $kl+ku$ superdiagonals, stored in the first $kl+ku+1$ rows. The multipliers, L , used during the factorization, are stored in rows $kl+ku+2$ to $2kl+ku+1$.

ldab The leading dimension of array **ab** in the calling program unit. $ldab \geq 2kl+ku+1$.

ipiv The pivot indices; for $1 \leq i \leq n$, row i of the matrix was interchanged with row $ipiv(i)$.

anorm If **norm** = '1' or 'O' or 'o', $\|A\|_1$ of the original matrix A .

If **norm** = 'I' or 'i', $\|A\|_\infty$ of the original matrix A .

Working Storage

work, iwork, rwork Arrays used for work space.

Output

rcond On successful exit, the estimate of the reciprocal condition number of the matrix A , computed as $rcond = (\|A\| \|A^{-1}\|)^{-1}$, using the norm specified by **norm**. If **rcond** is small enough so that the logical expression

$$1.0 + rcond \text{ .EQ. } 1.0$$

is true, then A can be regarded as singular to working precision. If **rcond** is zero, then the companion subprograms for solving and computing the inverse may divide by zero.

info Status response:

info = 0: Successful exit.

info < 0: If **info** = $-k$, the k -th argument had an invalid value.

Notes

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11) may be replaced with a user-supplied version to change the error procedure. Error conditions are

norm ≠ '1' or 'O' or 'o' or 'I' or 'i',
n < 0,
kl < 0,
ku < 0,
ldab < 2**kl**+**ku**+1, and
anorm < 0.0.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding the **norm** argument as 'One-norm' for 'O' or 'Infinity-norm' for 'I'.

Factor General Band Matrix**SGBTRF/DGBTRF/.../ZGBTRF**

Purpose These subprograms compute an LU factorization of an m -by- n general band matrix A using partial pivoting with row interchanges. A band matrix is a matrix whose nonzero elements all lie near the principal diagonal. Specifically, $a_{ij} = 0$ if $i-j > kl$ or $j-i > ku$ for some integers kl and ku . The smallest such kl and ku for a given matrix are called the lower and upper bandwidths, respectively, and $k = kl + ku + 1$ is the total bandwidth.

Matrix Storage Because it is not necessary to store or operate on the zeros outside the band of A , you need only provide the elements within the band of A . Compared to storing the entire matrix, this can save memory if $2kl + ku + 1 < n$.

The following example illustrates the storage of general band matrices. Consider the following matrix A of order $n = 9$ and lower and upper bandwidths $kl = 2$ and $ku = 3$, respectively:

11	12	13	14	0	0	0	0	0
21	22	23	24	25	0	0	0	0
31	32	33	34	35	36	0	0	0
0	42	43	44	45	46	47	0	0
0	0	53	54	55	56	57	58	0
0	0	0	64	65	66	67	68	69
0	0	0	0	75	76	77	78	79
0	0	0	0	0	86	87	88	89
0	0	0	0	0	0	97	98	99

When Gaussian elimination is performed on a general band matrix, pivoting introduces nonzero elements above the band. L can be stored with a lower bandwidth of kl , but U requires an upper bandwidth of $kl + ku$. You must, therefore, provide storage for the extra kl diagonals. This is done by presenting the original matrix to the subprogram in an array large enough to satisfy the additional storage requirements. Thus, for the above matrix, A is given in an array \mathbf{ab} with at least $2kl + ku + 1 = 8$ rows and $n = 9$ columns as follows:

*	*	*	*	*	+	+	+	+
*	*	*	*	+	+	+	+	+
*	*	*	14	25	36	47	58	69
*	*	13	24	35	46	57	68	79
*	12	23	34	45	56	67	78	89
11	22	33	44	55	66	77	88	99
21	32	43	54	65	76	87	98	*
31	42	53	64	75	86	97	*	*

The asterisks in the $(kl + ku)$ -by- $(kl + ku)$ triangle at the upper left corner and in the kl -by- kl triangle at the lower right corner represent elements of \mathbf{ab} that are not referenced, and the plus signs in the first kl rows indicate elements that may be filled in during the factorization. Thus, if a_{ij} is an element within the band of A , then it is stored in $\mathbf{ab}(kl + ku + 1 + i - j, j)$. Therefore, the columns of A are stored in the columns of \mathbf{ab} , and the diagonals of A are stored in the rows of \mathbf{ab} , such that the principal diagonal is stored in row $kl + ku + 1$ of \mathbf{ab} .

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```

INTEGER*4 info, kl, ku, ldab, m, n
INTEGER*4 ipiv(min(m,n))
REAL*4    ab(ldab, n)
CALL SGBTRF (m, n, kl, ku, ab, ldab, ipiv, info)

```

```

INTEGER*4 info, kl, ku, ldab, m, n
INTEGER*4 ipiv(min(m,n))
REAL*8    ab(ldab, n)
CALL DGBTRF (m, n, kl, ku, ab, ldab, ipiv, info)

```

```

INTEGER*4 info, kl, ku, ldab, m, n
INTEGER*4 ipiv(min(m,n))
COMPLEX*8 ab(ldab, n)
CALL CGBTRF (m, n, kl, ku, ab, ldab, ipiv, info)

```

```

INTEGER*4 info, kl, ku, ldab, m, n
INTEGER*4 ipiv(min(m,n))
COMPLEX*16 ab(ldab, n)
CALL ZGBTRF (m, n, kl, ku, ab, ldab, ipiv, info)

```

LAPACK8, available on C Series and Exemplar architectures:

```

INTEGER*8 info, kl, ku, ldab, m, n
INTEGER*8 ipiv(min(m,n))
REAL*8    ab(ldab, n)
CALL SGBTRF (m, n, kl, ku, ab, ldab, ipiv, info)

```

```

INTEGER*8 info, kl, ku, ldab, m, n
INTEGER*8 ipiv(min(m,n))
COMPLEX*16 ab(ldab, n)
CALL CGBTRF (m, n, kl, ku, ab, ldab, ipiv, info)

```

Input	m	The number of rows of the matrix A . $m \geq 0$.
	n	The number of columns of the matrix A . $n \geq 0$.
	kl	The number of subdiagonals within the band of A . $kl \geq 0$.
	ku	The number of superdiagonals within the band of A . $ku \geq 0$.
	ab	The matrix A in band storage, in rows $kl+1$ to $2kl+ku+1$; rows 1 to kl of array ab need not be set. The j -th column of A is stored in the j -th column of array ab as follows: $ab(kl+ku+1+i-j, j) = A(i, j)$ for $\max(1, j-ku) \leq i \leq \min(m, j+kl)$
	ldab	The leading dimension of array ab in the calling program unit. $ldab \geq 2kl+ku+1$.
Output	ab	On successful exit, details of the factorization. U is an upper triangular band matrix with $kl+ku$ superdiagonals, stored in rows 1 to $kl+ku+1$. The multipliers, L , used during the factorization, are stored in rows $kl+ku+2$ to $2kl+ku+1$.
	ipiv	On successful exit, the pivot indices; for $1 \leq i \leq \min(m, n)$, row i of the matrix was interchanged with row ipiv (i).

info Status response:

info = 0: Successful exit.

info < 0: If **info** = $-k$, the k -th argument had an invalid value.

info > 0: If **info** = k , $U(k,k)$ is zero. The factorization has been completed, but the factor U is singular, and division by zero will occur if it is used to solve a system of equations.

Notes

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11) may be replaced with a user-supplied version to change the error procedure. Error conditions are

m < 0,
n < 0,
kl < 0,
ku < 0, and
ldab < $2kl+ku+1$.

Purpose These subprograms solve a system of linear equations $AX = B$, $A^T X = B$, or $A^* X = B$ with a general band matrix A using the LU factorization computed by `_GBTRF`, where A^T is the transpose of A , and A^* is the conjugate transpose.

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```
CHARACTER*1 trans
INTEGER*4 info, kl, ku, ldab, ldb, n, nrhs
INTEGER*4 ipiv(n)
REAL*4 ab(ldab, n), b(ldb, nrhs)
CALL SGBTRS (trans, n, kl, ku, nrhs, ab, ldab, ipiv, b, ldb, info)
```

```
CHARACTER*1 trans
INTEGER*4 info, kl, ku, ldab, ldb, n, nrhs
INTEGER*4 ipiv(n)
REAL*8 ab(ldab, n), b(ldb, nrhs)
CALL DGBTRS (trans, n, kl, ku, nrhs, ab, ldab, ipiv, b, ldb, info)
```

```
CHARACTER*1 trans
INTEGER*4 info, kl, ku, ldab, ldb, n, nrhs
INTEGER*4 ipiv(n)
COMPLEX*8 ab(ldab, n), b(ldb, nrhs)
CALL CGBTRS (trans, n, kl, ku, nrhs, ab, ldab, ipiv, b, ldb, info)
```

```
CHARACTER*1 trans
INTEGER*4 info, kl, ku, ldab, ldb, n, nrhs
INTEGER*4 ipiv(n)
COMPLEX*16 ab(ldab, n), b(ldb, nrhs)
CALL ZGBTRS (trans, n, kl, ku, nrhs, ab, ldab, ipiv, b, ldb, info)
```

LAPACK8, available on C Series and Exemplar architectures:

```
CHARACTER*1 trans
INTEGER*8 info, kl, ku, ldab, ldb, n, nrhs
INTEGER*8 ipiv(n)
REAL*8 ab(ldab, n), b(ldb, nrhs)
CALL SGBTRS (trans, n, kl, ku, nrhs, ab, ldab, ipiv, b, ldb, info)
```

```
CHARACTER*1 trans
INTEGER*8 info, kl, ku, ldab, ldb, n, nrhs
INTEGER*8 ipiv(n)
COMPLEX*16 ab(ldab, n), b(ldb, nrhs)
CALL CGBTRS (trans, n, kl, ku, nrhs, ab, ldab, ipiv, b, ldb, info)
```

Input `trans` Specifies the form of the system of equations, as follows:

```
trans = 'N' or 'n': Solve  $AX = B$ .
trans = 'T' or 't': Solve  $A^T X = B$ .
trans = 'C' or 'c': Solve  $A^* X = B$ .
```

`n` The order of the matrix A . $n \geq 0$.

`kl` The number of subdiagonals within the band of A . $kl \geq 0$.

ku	The number of superdiagonals within the band of A . $ku \geq 0$.
nrhs	The number of right hand sides, that is, the number of columns of the matrix B . $nrhs \geq 0$.
ab	Details of the LU factorization of the band matrix A , as computed by <code>_GBTRF</code> . U is an upper triangular band matrix with $kl+ku$ superdiagonals, stored in the first $kl+ku+1$ rows. The multipliers, L , used during the factorization, are stored in rows $kl+ku+2$ to $2kl+ku+1$.
ldab	The leading dimension of array ab in the calling program unit. $ldab \geq 2kl+ku+1$.
ipiv	The pivot indices; for $1 \leq i \leq n$, row i of the matrix was interchanged with row <code>ipiv(i)</code> .
b	The n -by- $nrhs$ matrix of right hand side vectors for the system of linear equations $AX = B$.
ldb	The leading dimension of array b in the calling program unit. $ldb \geq \max(1,n)$.
Output	
b	On successful exit, the n -by- $nrhs$ matrix of solution vectors X overwrites the input.
info	Status response: info = 0: Successful exit. info < 0: If info = $-k$, the k -th argument had an invalid value.

Notes

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

trans \neq 'N' or 'n' or 'T' or 't' or 'C' or 'c',
n < 0,
kl < 0,
ku < 0,
nrhs < 0,
ldab < $2kl+ku+1$, and
ldb < $\max(1,n)$.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding the **trans** argument as 'NotTransposed' for 'N', 'Transposed' for 'T', or 'ConjugateTransposed' for 'C'.

Purpose These subprograms estimate the reciprocal of the condition number of a general matrix A , in either the 1-norm or the ∞ -norm, using the LU factorization computed by `_GETRF`.

An estimate is obtained for $\|A^{-1}\|$ and the reciprocal of the condition number is computed as $\text{rcond} = (\|A\| \|A^{-1}\|)^{-1}$.

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```

CHARACTER*1 norm
INTEGER*4 info, lda, n
REAL*4 anorm, rcond
INTEGER*4 iwork(n)
REAL*4 a(lda, n), work(4*n)
CALL SGECON (norm, n, a, lda, anorm, rcond, work, iwork, info)

```

```

CHARACTER*1 norm
INTEGER*4 info, lda, n
REAL*8 anorm, rcond
INTEGER*4 iwork(n)
REAL*8 a(lda, n), work(4*n)
CALL DGECON (norm, n, a, lda, anorm, rcond, work, iwork, info)

```

```

CHARACTER*1 norm
INTEGER*4 info, lda, n
REAL*4 anorm, rcond, rwork(2*n)
COMPLEX*8 a(lda, n), work(2*n)
CALL CGECON (norm, n, a, lda, anorm, rcond, work, rwork, info)

```

```

CHARACTER*1 norm
INTEGER*4 info, lda, n
REAL*8 anorm, rcond, rwork(2*n)
COMPLEX*16 a(lda, n), work(2*n)
CALL ZGECON (norm, n, a, lda, anorm, rcond, work, rwork, info)

```

LAPACK8, available on C Series and Exemplar architectures:

```

CHARACTER*1 norm
INTEGER*8 info, lda, n
REAL*8 anorm, rcond
INTEGER*8 iwork(n)
REAL*8 a(lda, n), work(4*n)
CALL SGECON (norm, n, a, lda, anorm, rcond, work, iwork, info)

```

```

CHARACTER*1 norm
INTEGER*8 info, lda, n
REAL*8 anorm, rcond, rwork(2*n)
COMPLEX*16 a(lda, n), work(2*n)
CALL CGECON (norm, n, a, lda, anorm, rcond, work, rwork, info)

```

Input **norm** Specifies whether to use the 1-norm or the ∞ -norm to estimate the condition number, as follows:

norm = '1', 'O', or 'o': Use $\| \cdot \|_1$.

norm = 'I' or 'i': Use $\| \cdot \|_{\infty}$.

- n** The order of the matrix A . $n \geq 0$.
- a** The factors L and U from the factorization $A = PLU$ as computed by `_GETRF`.
- lda** The leading dimension of array **a** in the calling program unit. $\text{lda} \geq \max(1, n)$.
- anorm** If **norm** = '1' or 'O' or 'o', $\|A\|_1$ of the original matrix A .
If **norm** = 'I' or 'i', $\|A\|_\infty$ of the original matrix A .

Working Storage **work,** Arrays used for work space.
iwork,
rwork

Output **rcond** On successful exit, the estimate of the reciprocal condition number of the matrix A , computed as $\text{rcond} = (\|A\| \|A^{-1}\|)^{-1}$, using the norm specified by **norm**. If **rcond** is small enough so that the logical expression

$$1.0 + \text{rcond} \text{ .EQ. } 1.0$$

is true, then A can be regarded as singular to working precision. If **rcond** is zero, then the companion subprograms for solving and computing the inverse may divide by zero.

- info** Status response:
- info** = 0: Successful exit.
info < 0: If **info** = $-k$, the k -th argument had an invalid value.

Notes If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

norm \neq '1' or 'O' or 'o' or 'I' or 'i',
n < 0,
lda < $\max(1, n)$, and
anorm < 0.0.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding the **norm** argument as 'Infinity-Norm' for 'I' or '1-norm' for '1'.

Purpose These subprograms compute the LU factorization of a general m -by- n matrix A using partial pivoting with row interchanges.

The factorization has the form $A = PLU$ 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 $m < n$).

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```
INTEGER*4 info, lda, m, n
INTEGER*4 ipiv(min(m,n))
REAL*4    a(lda, n)
CALL SGETRF (m, n, a, lda, ipiv, info)
```

```
INTEGER*4 info, lda, m, n
INTEGER*4 ipiv(min(m,n))
REAL*8    a(lda, n)
CALL DGETRF (m, n, a, lda, ipiv, info)
```

```
INTEGER*4 info, lda, m, n
INTEGER*4 ipiv(min(m,n))
COMPLEX*8 a(lda, n)
CALL CGETRF (m, n, a, lda, ipiv, info)
```

```
INTEGER*4 info, lda, m, n
INTEGER*4 ipiv(min(m,n))
COMPLEX*16 a(lda, n)
CALL ZGETRF (m, n, a, lda, ipiv, info)
```

LAPACK8, available on C Series and Exemplar architectures:

```
INTEGER*8 info, lda, m, n
INTEGER*8 ipiv(min(m,n))
REAL*8    a(lda, n)
CALL SGETRF (m, n, a, lda, ipiv, info)
```

```
INTEGER*8 info, lda, m, n
INTEGER*8 ipiv(min(m,n))
COMPLEX*16 a(lda, n)
CALL CGETRF (m, n, a, lda, ipiv, info)
```

Input

m The number of rows of the matrix A . $m \geq 0$.

n The number of columns of the matrix A . $n \geq 0$.

a The m -by- n matrix A to be factored.

lda The leading dimension of array a in the calling program unit. $lda \geq \max(1, m)$.

Output

a On successful exit, the factors L and U from the factorization $A = PLU$; the unit diagonal elements of L are not stored.

ipiv On successful exit, the pivot indices; for $1 \leq i \leq \min(m, n)$, row i of the matrix was interchanged with row $ipiv(i)$.

info Status response:

info = 0: Successful exit.

info < 0: If **info** = $-k$, the k -th argument had an invalid value.

info > 0: If **info** = k , $U(k,k)$ is zero. The factorization has been completed, but the factor U is singular, and division by zero will occur if it is used to solve a system of equations.

Notes

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

$m < 0$,
 $n < 0$, and
 $lda < \max(1,m)$.

Purpose	These subprograms compute the inverse of a general matrix using the <i>LU</i> factorization computed by <code>_GETRF</code> .	
Usage	LAPACK, available on C Series, Exemplar, and PA-RISC architectures:	
	<pre> INTEGER*4 info, lda, lwork, n INTEGER*4 ipiv(n) REAL*4 a(lda, n), work(lwork) CALL SGETRI (n, a, lda, ipiv, work, lwork, info) </pre>	
	<pre> INTEGER*4 info, lda, lwork, n INTEGER*4 ipiv(n) REAL*8 a(lda, n), work(lwork) CALL DGETRI (n, a, lda, ipiv, work, lwork, info) </pre>	
	<pre> INTEGER*4 info, lda, lwork, n INTEGER*4 ipiv(n) COMPLEX*8 a(lda, n), work(lwork) CALL CGETRI (n, a, lda, ipiv, work, lwork, info) </pre>	
	<pre> INTEGER*4 info, lda, lwork, n INTEGER*4 ipiv(n) COMPLEX*16 a(lda, n), work(lwork) CALL ZGETRI (n, a, lda, ipiv, work, lwork, info) </pre>	
	LAPACK8, available on C Series and Exemplar architectures:	
	<pre> INTEGER*8 info, lda, lwork, n INTEGER*8 ipiv(n) REAL*8 a(lda, n), work(lwork) CALL SGETRI (n, a, lda, ipiv, work, lwork, info) </pre>	
	<pre> INTEGER*8 info, lda, lwork, n INTEGER*8 ipiv(n) COMPLEX*16 a(lda, n), work(lwork) CALL CGETRI (n, a, lda, ipiv, work, lwork, info) </pre>	
Input	n	The order of the matrix <i>A</i> . $n \geq 0$.
	a	The factors <i>L</i> and <i>U</i> from the factorization $A = PLU$ as computed by <code>_GETRF</code> .
	lda	The leading dimension of array a in the calling program unit. $lda \geq \max(1, n)$.
	ipiv	The pivot indices from <code>_GETRF</code> ; for $1 \leq i \leq n$, row <i>i</i> of the matrix was interchanged with row <code>ipiv(i)</code> .
	lwork	The length of array work . $lwork \geq \max(1, n)$. For good performance, lwork must generally be larger. The optimum value of lwork for high performance is returned in <code>work(1)</code> .
Working Storage	work	An array used for work space. On successful exit, <code>work(1)</code> contains the optimal work space length lwork for high performance.

Output **a** On successful exit, the inverse of the original matrix A overwrites the input.

info Status response:

info = 0: Successful exit.

info < 0: If **info** = $-k$, the k -th argument had an invalid value.

info > 0: If **info** = k , $U(k,k)$ is zero; the matrix is singular and its inverse could not be computed.

Notes

It is almost never necessary to compute the inverse of a matrix. While papers and reference books extensively use the notation " $A^{-1}b$ " to mean "the solution x of the system of linear equations $Ax = b$," it is almost always more efficient and more accurate to compute the solution directly than to invert the coefficient matrix and multiply the inverse times the right hand side vector. This is true even if there are many systems of equations, all using the same coefficient matrix; the matrix may be factored once and the systems may be solved from the factors usually just as efficiently—and more accurately—than by matrix multiplication by the inverse.

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

n < 0,
lda < max(1,**n**), and
lwork < max(1,**n**).

Purpose These subprograms solve a system of linear equations $AX = B$, $A^T X = B$, or $A^* X = B$ with a general matrix A using the LU factorization computed by `_GETRF`, where A^T is the transpose of A , and A^* is the conjugate transpose.

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```

CHARACTER*1 trans
INTEGER*4 info, lda, ldb, n, nrhs
INTEGER*4 ipiv(n)
REAL*4 a(lda, n), b(ldb, nrhs)
CALL SGETRS (trans, n, nrhs, a, lda, ipiv, b, ldb, info)

```

```

CHARACTER*1 trans
INTEGER*4 info, lda, ldb, n, nrhs
INTEGER*4 ipiv(n)
REAL*8 a(lda, n), b(ldb, nrhs)
CALL DGETRS (trans, n, nrhs, a, lda, ipiv, b, ldb, info)

```

```

CHARACTER*1 trans
INTEGER*4 info, lda, ldb, n, nrhs
INTEGER*4 ipiv(n)
COMPLEX*8 a(lda, n), b(ldb, nrhs)
CALL CGETRS (trans, n, nrhs, a, lda, ipiv, b, ldb, info)

```

```

CHARACTER*1 trans
INTEGER*4 info, lda, ldb, n, nrhs
INTEGER*4 ipiv(n)
COMPLEX*16 a(lda, n), b(ldb, nrhs)
CALL ZGETRS (trans, n, nrhs, a, lda, ipiv, b, ldb, info)

```

LAPACK8, available on C Series and Exemplar architectures:

```

CHARACTER*1 trans
INTEGER*8 info, lda, ldb, n, nrhs
INTEGER*8 ipiv(n)
REAL*8 a(lda, n), b(ldb, nrhs)
CALL SGETRS (trans, n, nrhs, a, lda, ipiv, b, ldb, info)

```

```

CHARACTER*1 trans
INTEGER*8 info, lda, ldb, n, nrhs
INTEGER*8 ipiv(n)
COMPLEX*16 a(lda, n), b(ldb, nrhs)
CALL CGETRS (trans, n, nrhs, a, lda, ipiv, b, ldb, info)

```

Input **trans** Specifies the form of the system of equations, as follows:

trans = 'N' or 'n': Solve $AX = B$.

trans = 'T' or 't': Solve $A^T X = B$.

trans = 'C' or 'c': Solve $A^* X = B$.

n The order of the matrix A . $n \geq 0$.

nrhs The number of right hand sides, that is, the number of columns of the matrix B . $nrhs \geq 0$.

	a	The factors L and U from the factorization $A = PLU$ as computed by <code>_GETRF</code> .
	lda	The leading dimension of array a in the calling program unit. $lda \geq \max(1,n)$.
	ipiv	The pivot indices from <code>_GETRF</code> ; for $1 \leq i \leq n$, row i of the matrix was interchanged with row <code>ipiv(i)</code> .
	b	The n -by- nrhs matrix of right hand side vectors for the system of linear equations $AX = B$.
	ldb	The leading dimension of array b in the calling program unit. $ldb \geq \max(1,n)$.
Output	b	On successful exit, the n -by- nrhs matrix of solution vectors X overwrites the input.
	info	Status response: info = 0: Successful exit. info < 0: If info = $-k$, the k -th argument had an invalid value.

Notes

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

trans \neq 'N' or 'n' or 'T' or 't' or 'C' or 'c',
n < 0,
nrhs < 0,
lda < $\max(1,n)$, and
ldb < $\max(1,n)$.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding the **trans** argument as 'NotTransposed' for 'N', 'Transposed' for 'T', or 'ConjugateTransposed' for 'C'.

Purpose These subprograms estimate the reciprocal of the condition number of a general tridiagonal matrix A , in either the 1-norm or the ∞ -norm, using the LU factorization computed by _GTTRF.

An estimate is obtained for $\|A^{-1}\|$, and the reciprocal of the condition number is computed as $\mathbf{rcond} = (\|A\| \|A^{-1}\|)^{-1}$.

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```

CHARACTER*1 norm
INTEGER*4 info, n
REAL*4 anorm, rcond
INTEGER*4 ipiv(n), iwork(n)
REAL*4 d(n), dl(n-1), du(n-1), du2(n-2), work(n)
CALL SGTCON (norm, n, dl, d, du, du2, ipiv, anorm, rcond,
work, iwork, info)

```

```

CHARACTER*1 norm
INTEGER*4 info, n
REAL*8 anorm, rcond
INTEGER*4 ipiv(n), iwork(n)
REAL*8 d(n), dl(n-1), du(n-1), du2(n-2), work(n)
CALL DGTCON (norm, n, dl, d, du, du2, ipiv, anorm, rcond,
work, iwork, info)

```

```

CHARACTER*1 norm
INTEGER*4 info, n
REAL*4 anorm, rcond
INTEGER*4 ipiv(n)
COMPLEX*8 d(n), dl(n-1), du(n-1), du2(n-2), work(n)
CALL CGTCON (norm, n, dl, d, du, du2, ipiv, anorm, rcond,
work, info)

```

```

CHARACTER*1 norm
INTEGER*4 info, n
REAL*8 anorm, rcond
INTEGER*4 ipiv(n)
COMPLEX*16 d(n), dl(n-1), du(n-1), du2(n-2), work(n)
CALL ZGTCON (norm, n, dl, d, du, du2, ipiv, anorm, rcond,
work, info)

```

LAPACK8, available on C Series and Exemplar architectures:

```

CHARACTER*1 norm
INTEGER*8 info, n
REAL*8 anorm, rcond
INTEGER*8 ipiv(n), iwork(n)
REAL*8 d(n), dl(n-1), du(n-1), du2(n-2), work(n)
CALL SGTCON (norm, n, dl, d, du, du2, ipiv, anorm, rcond,
work, iwork, info)

```

CHARACTER*1 norm
INTEGER*8 info, n
REAL*8 anorm, rcond
INTEGER*8 ipiv(n)
COMPLEX*16 d(n), dl(n-1), du(n-1), du2(n-2), work(n)
CALL CGTCON (norm, n, dl, d, du, du2, ipiv, anorm, rcond,
work, info)

Input

norm Specifies whether to use the 1-norm or the ∞ -norm to estimate the condition number, as follows:

norm = '1', 'O', or 'o': Use $\| \cdot \|_1$.

norm = 'I' or 'i': Use $\| \cdot \|_\infty$.

n The order of the matrix A . $n \geq 0$.

dl The $n-1$ multipliers that define the matrix L from the LU factorization of A .

d The n diagonal elements of the upper triangular matrix U from the LU factorization of A .

du The $n-1$ elements of the first superdiagonal of U .

du2 The $n-2$ elements of the second superdiagonal of U .

ipiv The pivot indices; for $1 \leq i \leq n$, row i of the matrix was interchanged with row $ipiv(i)$.

anorm If **norm** = '1' or 'O' or 'o', $\|A\|_1$ of the original matrix A .

If **norm** = 'I' or 'i', $\|A\|_\infty$ of the original matrix A .

Working Storage

work, iwork Arrays used for work space.

Output

rcond On successful exit, the estimate of the reciprocal condition number of the matrix A , computed as $\mathbf{rcond} = (\|A\| \|A^{-1}\|)^{-1}$, using the norm specified by **norm**. If **rcond** is small enough so that the logical expression

$$1.0 + \mathbf{rcond} .EQ. 1.0$$

is true, then A can be regarded as singular to working precision. If **rcond** is zero, then the companion subprograms for solving and computing the inverse may divide by zero.

info Status response:

info = 0: Successful exit.

info < 0: If **info** = $-k$, the k -th argument had an invalid value.

Notes

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11) may be replaced with a user-supplied version to change the error procedure. Error conditions are

norm ≠ '1' or 'O' or 'o' or 'I' or 'i',
n < 0, and
anorm < 0.0.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding the **norm** argument as 'One-norm' for 'O' or 'Infinity-norm' for 'I'.

Factor General Tridiagonal Matrix**SGTTRF/DGTTRF/.../ZGTTRF**

Purpose These subprograms compute an LU factorization of a general tridiagonal matrix A using partial pivoting with row interchanges. A matrix $A = (a_{ij})$ is tridiagonal if its nonzero elements lie only on the principal diagonal ($i = j$), the subdiagonal ($i = j+1$), and the superdiagonal ($i = j-1$) of the matrix.

Given such a matrix A , these subprograms compute the factorization of the form $A = LU$ where L is a product of permutation and unit lower bidiagonal matrices and U is upper triangular with nonzeros on only the principal diagonal and first two superdiagonals.

Matrix Storage The following example illustrates the storage of general tridiagonal matrices. Consider the following tridiagonal matrix of order $n = 7$:

11	12	0	0	0	0	0
21	22	23	0	0	0	0
0	32	33	34	0	0	0
0	0	43	44	45	0	0
0	0	0	54	55	56	0
0	0	0	0	65	66	67
0	0	0	0	0	76	77

The subdiagonal is stored in array **dl**, the principal diagonal is stored in array **d**, and the superdiagonal is stored in array **du**, as follows:

i	dl (i)	d (i)	du (i)
1	21	11	12
2	32	22	23
3	43	33	34
4	54	44	45
5	65	55	56
6	76	66	67
7		77	

Usage

LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```

INTEGER*4 info, n
INTEGER*4 ipiv(n)
REAL*4 d(n), dl(n-1), du(n-1), du2(n-2)
CALL SGTTRF (n, dl, d, du, du2, ipiv, info)

```

```

INTEGER*4 info, n
INTEGER*4 ipiv(n)
REAL*8 d(n), dl(n-1), du(n-1), du2(n-2)
CALL DGTTRF (n, dl, d, du, du2, ipiv, info)

```

```

INTEGER*4 info, n
INTEGER*4 ipiv(n)
COMPLEX*8 d(n), dl(n-1), du(n-1), du2(n-2)
CALL CGTTRF (n, dl, d, du, du2, ipiv, info)

```

```

INTEGER*4 info, n
INTEGER*4 ipiv(n)
COMPLEX*16 d(n), dl(n-1), du(n-1), du2(n-2)
CALL ZGTTRF (n, dl, d, du, du2, ipiv, info)

```

LAPACK8, available on C Series and Exemplar architectures:

```

INTEGER*8 info, n
INTEGER*8 ipiv(n)
REAL*8    d(n), dl(n-1), du(n-1), du2(n-2)
CALL SGTTRF (n, dl, d, du, du2, ipiv, info)

```

```

INTEGER*8 info, n
INTEGER*8 ipiv(n)
COMPLEX*16 d(n), dl(n-1), du(n-1), du2(n-2)
CALL CGTTRF (n, dl, d, du, du2, ipiv, info)

```

Input	n	The order of the matrix A . $n \geq 0$.
	dl	The $n-1$ subdiagonal elements of A .
	d	The diagonal elements of A .
	du	The $n-1$ superdiagonal elements of A .
Output	dl	On successful exit, the $n-1$ multipliers that define the matrix L from the LU factorization of A .
	d	On successful exit, the n diagonal elements of the upper triangular matrix U from the LU factorization of A .
	du	On successful exit, the $n-1$ elements of the first superdiagonal of U .
	du2	On successful exit, the $n-2$ elements of the second superdiagonal of U .
	ipiv	On successful exit, the pivot indices from the LU factorization of A ; row i of the matrix was interchanged with row $\text{ipiv}(i)$. $\text{ipiv}(i)$ will always be either i or $i+1$; $\text{ipiv}(i) = i$ indicates a row interchange was not required.
	info	Status response: info = 0: Successful exit. info < 0: If $\text{info} = -k$, the k -th argument had an invalid value. info > 0: If $\text{info} = k$, $U(k, k)$ is zero. The factorization has been completed, but the factor U is singular, and division by zero will occur if it is used to solve a system of equations.

Notes If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11) may be replaced with a user-supplied version to change the error procedure. Error conditions are

$n < 0$.

Solve General Tridiagonal System**SGTTRS/DGTTRS/.../ZGTTRS**

Purpose These subprograms solve a system of linear equations $AX = B$, $A^T X = B$, or $A^* X = B$ with a general tridiagonal matrix A using the LU factorization computed by `_GTTRF`, where A^T is the transpose of A , and A^* is the conjugate transpose.

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```
CHARACTER*1 trans
INTEGER*4    info, ldb, n, nrhs
INTEGER*4    ipiv(n)
REAL*4       b(ldb, nrhs), d(n), dl(n-1), du(n-1), du2(n-2)
CALL SGTTRS (trans, n, nrhs, dl, d, du, du2, ipiv, b, ldb, info)
```

```
CHARACTER*1 trans
INTEGER*4    info, ldb, n, nrhs
INTEGER*4    ipiv(n)
REAL*8       b(ldb, nrhs), d(n), dl(n-1), du(n-1), du2(n-2)
CALL DGTTRS (trans, n, nrhs, dl, d, du, du2, ipiv, b, ldb, info)
```

```
CHARACTER*1 trans
INTEGER*4    info, ldb, n, nrhs
INTEGER*4    ipiv(n)
COMPLEX*8    b(ldb, nrhs), d(n), dl(n-1), du(n-1), du2(n-2)
CALL CGTTRS (trans, n, nrhs, dl, d, du, du2, ipiv, b, ldb, info)
```

```
CHARACTER*1 trans
INTEGER*4    info, ldb, n, nrhs
INTEGER*4    ipiv(n)
COMPLEX*16   b(ldb, nrhs), d(n), dl(n-1), du(n-1), du2(n-2)
CALL ZGTTRS (trans, n, nrhs, dl, d, du, du2, ipiv, b, ldb, info)
```

LAPACK8, available on C Series and Exemplar architectures:

```
CHARACTER*1 trans
INTEGER*8    info, ldb, n, nrhs
INTEGER*8    ipiv(n)
REAL*8       b(ldb, nrhs), d(n), dl(n-1), du(n-1), du2(n-2)
CALL SGTTRS (trans, n, nrhs, dl, d, du, du2, ipiv, b, ldb, info)
```

```
CHARACTER*1 trans
INTEGER*8    info, ldb, n, nrhs
INTEGER*8    ipiv(n)
COMPLEX*16   b(ldb, nrhs), d(n), dl(n-1), du(n-1), du2(n-2)
CALL CGTTRS (trans, n, nrhs, dl, d, du, du2, ipiv, b, ldb, info)
```

Input **trans** Specifies the form of the system of equations, as follows:

```
trans = 'N' or 'n':  Solve  $AX = B$ .
trans = 'T' or 't':  Solve  $A^T X = B$ .
trans = 'C' or 'c':  Solve  $A^* X = B$ .
```

n The order of the matrix A . $n \geq 0$.

nrhs The number of right hand sides, that is, the number of columns of the matrix B . $nrhs \geq 0$.

	dl	The $n-1$ multipliers that define the matrix L from the LU factorization of A .
	d	The n diagonal elements of the upper triangular matrix U from the LU factorization of A .
	du	The $n-1$ elements of the first superdiagonal of U .
	du2	The $n-2$ elements of the second superdiagonal of U .
	ipiv	The pivot indices; for $1 \leq i \leq n$, row i of the matrix was interchanged with row $ipiv(i)$.
	b	The n -by- $nrhs$ matrix of right hand side vectors for the system of linear equations $AX = B$.
	ldb	The leading dimension of array b in the calling program unit. $ldb \geq \max(1,n)$.
Output	b	On successful exit, the n -by- $nrhs$ matrix of solution vectors X overwrites the input.
	info	Status response: info = 0: Successful exit. info < 0: If info = $-k$, the k -th argument had an invalid value.

Notes If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

trans \neq 'N' or 'n' or 'T' or 't' or 'C' or 'c',
n < 0,
nrhs < 0, and
ldb < $\max(1,n)$.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding the **trans** argument as 'NotTransposed' for 'N', 'Transposed' for 'T', or 'ConjugateTransposed' for 'C'.

Condition Number of Positive Definite Band Matrix**SPBCON/.../ZPBCON**

Purpose These subprograms estimate the reciprocal of the condition number of a real symmetric or complex Hermitian positive definite band matrix A using the Cholesky factorization computed by `_PBTRF`.

An estimate is obtained for $\|A^{-1}\|$, and the reciprocal of the condition number is computed as $\text{rcond} = (\|A\|_1 \|A^{-1}\|_1)^{-1}$.

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```

CHARACTER*1 uplo
INTEGER*4   info, kd, ldab, n
REAL*4     anorm, rcond
INTEGER*4   iwork(n)
REAL*4     ab(ldab, n), work(3*n)
CALL SPBCON (uplo, n, kd, ab, ldab, anorm, rcond, work, iwork,
              info)

```

```

CHARACTER*1 uplo
INTEGER*4   info, kd, ldab, n
REAL*8     anorm, rcond
INTEGER*4   iwork(n)
REAL*8     ab(ldab, n), work(3*n)
CALL DPBCON (uplo, n, kd, ab, ldab, anorm, rcond, work, iwork,
              info)

```

```

CHARACTER*1 uplo
INTEGER*4   info, kd, ldab, n
REAL*4     anorm, rcond
REAL*4     rwork(n)
COMPLEX*8  ab(ldab, n), work(2*n)
CALL CPBCON (uplo, n, kd, ab, ldab, anorm, rcond, work, rwork,
              info)

```

```

CHARACTER*1 uplo
INTEGER*4   info, kd, ldab, n
REAL*8     anorm, rcond
REAL*8     rwork(n)
COMPLEX*16 ab(ldab, n), work(2*n)
CALL ZPBCON (uplo, n, kd, ab, ldab, anorm, rcond, work, rwork,
              info)

```

LAPACK8, available on C Series and Exemplar architectures:

```

CHARACTER*1 uplo
INTEGER*8   info, kd, ldab, n
REAL*8     anorm, rcond
INTEGER*8   iwork(n)
REAL*8     ab(ldab, n), work(3*n)
CALL SPBCON (uplo, n, kd, ab, ldab, anorm, rcond, work, iwork,
              info)

```

CHARACTER*1 uplo
INTEGER*8 info, kd, ldab, n
REAL*8 anorm, rcond
REAL*8 rwork(n)
COMPLEX*16 ab(ldab, n), work(2*n)
CALL CPBCON (uplo, n, kd, ab, ldab, anorm, rcond, work, rwork,
 info)

Input **uplo** Specifies whether the upper or lower triangular factor of the symmetric or Hermitian matrix A is stored, as follows:

 uplo = 'U' or 'u': U , the upper triangular factor of A is stored.
 uplo = 'L' or 'l': L , the lower triangular factor of A is stored.

n The order of the matrix A . $n \geq 0$.

kd The number of super-diagonals of the matrix A if **uplo** = 'U' or 'u', or the number of sub-diagonals if **uplo** = 'L' or 'l'. $kd \geq 0$.

ab The triangular factor U or L from the Cholesky factorization $A = U*U$ or $A = LL^*$ of the band matrix A , stored in the first $kd+1$ rows of the array. The j -th column of U or L is stored in the array **ab** as follows:

 If **uplo** = 'U' or 'u', $ab(kd+1+i-j, j) = U(i, j)$ for $\max(1, j-kd) \leq i \leq j$;
 If **uplo** = 'L' or 'l', $ab(1+i-j, j) = L(i, j)$ for $j \leq i \leq \min(n, j+kd)$.

ldab The leading dimension of array **ab** in the calling program unit. $ldab \geq kd+1$.

anorm $\|A\|_1$ (= $\|A\|_\infty$) of the original symmetric or Hermitian band matrix A . $anorm \geq 0$.

Working Storage **work,** Arrays used for work space.
 iwork,
 rwork

Output **rcond** On successful exit, the estimate of the reciprocal condition number of the matrix A , computed as $rcond = (\|A\|_1 \|A^{-1}\|_1)^{-1}$. If **rcond** is small enough so that the logical expression

 $1.0 + rcond .EQ. 1.0$

 is true, then A can be regarded as singular to working precision. If **rcond** is zero, then the companion subprograms for solving and computing the inverse may divide by zero.

info Status response:

 info = 0: Successful exit.
 info < 0: If **info** = $-k$, the k -th argument had an invalid value.

Notes

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

uplo \neq 'L' or 'l' or 'U' or 'u',
n < 0 ,
kd < 0 ,
ldab $< \mathbf{kd}+1$, and
anorm < 0.0 .

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding the **uplo** argument as 'Lower' for 'L' or 'Upper' for 'U'.

Purpose These subprograms compute the Cholesky factorization of a real symmetric or complex Hermitian positive definite band matrix A . A real matrix is symmetric if $A = A^T$, its transpose; a complex matrix is Hermitian if $A = A^*$, its conjugate transpose.

A real symmetric matrix A is positive definite if the quadratic form $x^T A x$ is positive for all nonzero real vectors x ; a complex Hermitian matrix A is positive definite if the quadratic form $x^* A x$ is positive for all nonzero complex vectors x .

A positive definite band matrix is a positive definite matrix whose nonzero elements all lie fairly near the principal diagonal. Specifically, $a_{ij} = 0$ if $|i-j| > kd$ for some integer kd . The smallest such kd for a given matrix is called the half bandwidth, and $2kd+1$ is called the total bandwidth.

Tridiagonal matrices are the special case $kd = 1$. They can be handled more efficiently by the LAPACK subprograms SPTRF, DPTTRF, CPTTRF, and ZPTTRF.

Given such a matrix A , these subprograms compute the Cholesky factorization of the form $A = U^*U$ or $A = LL^*$ where U is an upper triangular matrix and L is a lower triangular matrix.

Matrix Storage Because it is not necessary to store or operate on the zeros outside the band of A , and since either triangle of A may be obtained from the other, you need only provide the band within one triangle of A . Compared to storing the entire matrix, this can save memory in two ways: only the elements within the band are stored, and of them, only the upper or the lower triangle.

The following examples illustrate the storage of positive definite band matrices. Consider the following matrix A of order $n = 7$ and half bandwidth $kd = 2$:

11	12	13	0	0	0	0
12	22	23	24	0	0	0
13	23	33	34	35	0	0
0	24	34	44	45	46	0
0	0	35	45	55	56	57
0	0	0	46	56	66	67
0	0	0	0	57	67	77

Upper triangular storage. The upper triangle of A is stored in an array **ab** with at least $kd+1 = 3$ rows and 7 columns as follows:

*	*	13	24	35	46	57
*	12	23	34	45	56	67
11	22	33	44	55	66	77

The asterisks represent elements in the kd -by- kd triangle at the upper left corner of **ab** that are not referenced. Thus, if a_{ij} is an element within the band of the upper triangle of A , it is stored in **ab**($kd+1+i-j, j$). Therefore, the columns of the upper triangle of A are stored in the columns of **ab**, and the diagonals of the upper triangle of A are stored in the rows of **ab**.

Lower triangular storage. The lower triangle of A is stored in the array ab as follows:

11	22	33	44	55	66	77
12	23	34	45	56	67	*
13	24	35	46	57	*	*

The asterisks represent elements in the kd -by- kd triangle at the lower right corner of ab that are not referenced. Thus, if a_{ij} is an element within the band of the lower triangle of A , it is stored in $ab(1+i-j, j)$. Therefore, the columns of the lower triangle of A are stored in the columns of ab , and the diagonals of the lower triangle of A are stored in the rows of ab .

Usage

LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```

CHARACTER*1 uplo
INTEGER*4 info, kd, ldab, n
REAL*4 ab(ldab, n)
CALL SPBTRF (uplo, n, kd, ab, ldab, info)

```

```

CHARACTER*1 uplo
INTEGER*4 info, kd, ldab, n
REAL*8 ab(ldab, n)
CALL DPBTRF (uplo, n, kd, ab, ldab, info)

```

```

CHARACTER*1 uplo
INTEGER*4 info, kd, ldab, n
COMPLEX*8 ab(ldab, n)
CALL CPBTRF (uplo, n, kd, ab, ldab, info)

```

```

CHARACTER*1 uplo
INTEGER*4 info, kd, ldab, n
COMPLEX*16 ab(ldab, n)
CALL ZPBTRF (uplo, n, kd, ab, ldab, info)

```

LAPACK8, available on C Series and Exemplar architectures:

```

CHARACTER*1 uplo
INTEGER*8 info, kd, ldab, n
REAL*8 ab(ldab, n)
CALL SPBTRF (uplo, n, kd, ab, ldab, info)

```

```

CHARACTER*1 uplo
INTEGER*8 info, kd, ldab, n
COMPLEX*16 ab(ldab, n)
CALL CPBTRF (uplo, n, kd, ab, ldab, info)

```

Input

uplo Specifies whether the upper or lower triangular part of the symmetric or Hermitian matrix A is stored, as follows:

uplo = 'U' or 'u': The upper triangular part of A is stored.

uplo = 'L' or 'l': The lower triangular part of A is stored.

n The order of the matrix A . $n \geq 0$.

kd The number of super-diagonals of the matrix A if **uplo** = 'U' or 'u', or the number of sub-diagonals if **uplo** = 'L' or 'l'. $kd \geq 0$.

- ab** The upper or lower triangle of the symmetric or Hermitian band matrix A , stored in the first $kd+1$ rows of the array. The j -th column of A is stored in the j -th column of array **ab** as follows:
- If **uplo** = 'U' or 'u', $ab(kd+1+i-j, j) = A(i, j)$ for $\max(1, j-kd) \leq i \leq j$;
- If **uplo** = 'L' or 'l', $ab(1+i-j, j) = A(i, j)$ for $j \leq i \leq \min(n, j+kd)$.
- ldab** The leading dimension of array **ab** in the calling program unit. $ldab \geq kd+1$.
- Output**
- ab** On successful exit, the triangular factor U or L from the Cholesky factorization of A in the same storage format as A overwrites the input.
- info** Status response:
- info** = 0: Successful exit.
- info** < 0: If **info** = $-k$, the k -th argument had an invalid value.
- info** > 0: If **info** = k , the leading minor of order k is not positive definite, and the factorization could not be completed.
- Notes**
- If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are
- uplo** ≠ 'L' or 'l' or 'U' or 'u',
n < 0,
kd < 0, and
ldab < **kd**+1.
- Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding the **uplo** argument as 'Lower' for 'L' or 'Upper' for 'U'.

Solve Positive Definite Band System**SPBTRS/.../ZPBTRS**

Purpose These subprograms solve a system of linear equations $AX = B$ with a real symmetric or complex Hermitian positive definite band matrix A using the Cholesky factorization computed by `_PBTRF`.

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```

CHARACTER*1 uplo
INTEGER*4   info, kd, ldab, ldb, n, nrhs
REAL*4     ab(ldab, n), b(ldb, nrhs)
CALL SPBTRS (uplo, n, kd, nrhs, ab, ldab, b, ldb, info)

```

```

CHARACTER*1 uplo
INTEGER*4   info, kd, ldab, ldb, n, nrhs
REAL*8     ab(ldab, n), b(ldb, nrhs)
CALL DPBTRS (uplo, n, kd, nrhs, ab, ldab, b, ldb, info)

```

```

CHARACTER*1 uplo
INTEGER*4   info, kd, ldab, ldb, n, nrhs
COMPLEX*8  ab(ldab, n), b(ldb, nrhs)
CALL CPBTRS (uplo, n, kd, nrhs, ab, ldab, b, ldb, info)

```

```

CHARACTER*1 uplo
INTEGER*4   info, kd, ldab, ldb, n, nrhs
COMPLEX*16 ab(ldab, n), b(ldb, nrhs)
CALL ZPBTRS (uplo, n, kd, nrhs, ab, ldab, b, ldb, info)

```

LAPACK8, available on C Series and Exemplar architectures:

```

CHARACTER*1 uplo
INTEGER*8   info, kd, ldab, ldb, n, nrhs
REAL*8     ab(ldab, n), b(ldb, nrhs)
CALL SPBTRS (uplo, n, kd, nrhs, ab, ldab, b, ldb, info)

```

```

CHARACTER*1 uplo
INTEGER*8   info, kd, ldab, ldb, n, nrhs
COMPLEX*16 ab(ldab, n), b(ldb, nrhs)
CALL CPBTRS (uplo, n, kd, nrhs, ab, ldab, b, ldb, info)

```

Input **uplo** Specifies whether the upper or lower triangular factor of the symmetric or Hermitian matrix A is stored, as follows:

uplo = 'U' or 'u': U , the upper triangular factor of A is stored.

uplo = 'L' or 'l': L , the lower triangular factor of A is stored.

n The order of the matrix A . $n \geq 0$.

kd The number of super-diagonals of the matrix A if **uplo** = 'U' or 'u', or the number of sub-diagonals if **uplo** = 'L' or 'l'. $kd \geq 0$.

nrhs The number of right hand sides, that is, the number of columns of the matrix B . $nrhs \geq 0$.

- ab** The triangular factor U or L from the Cholesky factorization of the band matrix A , stored in the first $kd+1$ rows of the array. The j -th column of U or L is stored in the array **ab** as follows:
- If **uplo** = 'U' or 'u', $ab(kd+1+i-j, j) = U(i, j)$ for $\max(1, j-kd) \leq i \leq j$;
- If **uplo** = 'L' or 'l', $ab(1+i-j, j) = L(i, j)$ for $j \leq i \leq \min(n, j+kd)$.
- ldab** The leading dimension of array **ab** in the calling program unit. **ldab** $\geq kd+1$.
- b** The n -by-**nrhs** matrix of right hand side vectors for the system of linear equations $AX = B$.
- ldb** The leading dimension of array **b** in the calling program unit. **ldb** $\geq \max(1, n)$.
- Output**
- b** On successful exit, the n -by-**nrhs** matrix of solution vectors X overwrites the input.
- info** Status response:
- info** = 0: Successful exit.
- info** < 0: If **info** = $-k$, the k -th argument had an invalid value.

Notes If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

uplo \neq 'L' or 'l' or 'U' or 'u',
n < 0,
kd < 0,
nrhs < 0,
ldab < **kd**+1, and
ldb < $\max(1, n)$.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding the **uplo** argument as 'Lower' for 'L' or 'Upper' for 'U'.

Condition Number of Positive Definite Matrix**SPOCON/.../ZPOCON**

Purpose These subprograms estimate the reciprocal of the condition number of a real symmetric or complex Hermitian positive definite matrix A using the Cholesky factorization computed by `_POTRF`.

An estimate is obtained for $\|A^{-1}\|_1$, and the reciprocal of the condition number is computed as `rcond` = $(\|A\|_1 \|A^{-1}\|_1)^{-1}$.

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```
CHARACTER*1 uplo
INTEGER*4   info, lda, n
REAL*4      anorm, rcond
INTEGER*4   iwork(n)
REAL*4      a(lda, n), work(3*n)
CALL SPOCON (uplo, n, a, lda, anorm, rcond, work, iwork, info)
```

```
CHARACTER*1 uplo
INTEGER*4   info, lda, n
REAL*8      anorm, rcond
INTEGER*4   iwork(n)
REAL*8      a(lda, n), work(3*n)
CALL DPOCON (uplo, n, a, lda, anorm, rcond, work, iwork, info)
```

```
CHARACTER*1 uplo
INTEGER*4   info, lda, n
REAL*4      anorm, rcond, rwork(n)
COMPLEX*8   a(lda, n), work(2*n)
CALL CPOCON (uplo, n, a, lda, anorm, rcond, work, rwork, info)
```

```
CHARACTER*1 uplo
INTEGER*4   info, lda, n
REAL*8      anorm, rcond, rwork(n)
COMPLEX*16  a(lda, n), work(2*n)
CALL ZPOCON (uplo, n, a, lda, anorm, rcond, work, rwork, info)
```

LAPACK8, available on C Series and Exemplar architectures:

```
CHARACTER*1 uplo
INTEGER*8   info, lda, n
REAL*8      anorm, rcond
INTEGER*8   iwork(n)
REAL*8      a(lda, n), work(3*n)
CALL SPOCON (uplo, n, a, lda, anorm, rcond, work, iwork, info)
```

```
CHARACTER*1 uplo
INTEGER*8   info, lda, n
REAL*8      anorm, rcond, rwork(n)
COMPLEX*16  a(lda, n), work(2*n)
CALL CPOCON (uplo, n, a, lda, anorm, rcond, work, rwork, info)
```

Input `uplo` Specifies whether the upper or lower triangular factor of the symmetric or Hermitian matrix A is stored, as follows:

`uplo` = 'U' or 'u': U , the upper triangular factor of A is stored.
`uplo` = 'L' or 'l': L , the lower triangular factor of A is stored.

	n	The order of the matrix A . $n \geq 0$.
	a	The triangular factor U or L from the Cholesky factorization as computed by <code>_POTRF</code> .
	lda	The leading dimension of array a in the calling program unit. $lda \geq \max(1,n)$.
	anorm	$\ A\ _1$ ($= \ A\ _\infty$) of the original symmetric or Hermitian matrix A . $anorm \geq 0$.
Working Storage	work, iwork, rwork	Arrays used for work space
Output	rcond	On successful exit, the estimate of the reciprocal condition number of the matrix A , computed as $rcond = (\ A\ _1 \ A^{-1}\ _1)^{-1}$. If rcond is small enough so that the logical expression

$$1.0 + rcond .EQ. 1.0$$

is true, then A can be regarded as singular to working precision. If **rcond** is zero, then the companion subprograms for solving and computing the inverse may divide by zero.

info Status response:

info = 0: Successful exit.

info < 0: If **info** = $-k$, the k -th argument had an invalid value.

Notes If an error in the arguments is detected, the subprograms call error handler `XERBLA`, which writes an error message onto the standard error file and terminates execution. The standard version of `XERBLA` (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

uplo \neq 'L' or 'l' or 'U' or 'u',
n < 0,
lda < $\max(1,n)$, and
anorm < 0.0.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the `CALL` statement may be improved by coding the **uplo** argument as 'Lower' for 'L' or 'Upper' for 'U'.

Factor Positive Definite Matrix**SPOTRF/DPOTRF/CPOTRF/ZPOTRF**

Purpose These subprograms compute the Cholesky factorization of a real symmetric or complex Hermitian positive definite matrix A . A real matrix is symmetric if $A = A^T$, its transpose; a complex matrix is Hermitian if $A = A^*$, its conjugate transpose.

A real symmetric matrix A is positive definite if the quadratic form $x^T A x$ is positive for all nonzero real vectors x ; a complex Hermitian matrix A is positive definite if the quadratic form $x^* A x$ is positive for all nonzero complex vectors x .

Given such a matrix A , these subprograms compute the Cholesky factorization of the form $A = U^* U$ or $A = L L^*$ where U is an upper triangular matrix, L is a lower triangular matrix.

Matrix Storage Because either triangle of A may be obtained from the other, you need only provide one triangle of A . You may supply either the upper or the lower triangle of A , in a two-dimensional array large enough to hold the entire matrix. The other triangle of the array is not referenced.

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```
CHARACTER*1 uplo
INTEGER*4   info, lda, n
REAL*4      a(lda, n)
CALL SPOTRF (uplo, n, a, lda, info)
```

```
CHARACTER*1 uplo
INTEGER*4   info, lda, n
REAL*8      a(lda, n)
CALL DPOTRF (uplo, n, a, lda, info)
```

```
CHARACTER*1 uplo
INTEGER*4   info, lda, n
COMPLEX*8   a(lda, n)
CALL CPOTRF (uplo, n, a, lda, info)
```

```
CHARACTER*1 uplo
INTEGER*4   info, lda, n
COMPLEX*16  a(lda, n)
CALL ZPOTRF (uplo, n, a, lda, info)
```

LAPACK8, available on C Series and Exemplar architectures:

```
CHARACTER*1 uplo
INTEGER*8   info, lda, n
REAL*8      a(lda, n)
CALL SPOTRF (uplo, n, a, lda, info)
```

```
CHARACTER*1 uplo
INTEGER*8   info, lda, n
COMPLEX*16  a(lda, n)
CALL CPOTRF (uplo, n, a, lda, info)
```

Input **uplo** Specifies whether the upper or lower triangular part of the symmetric or Hermitian matrix A is stored, as follows:

uplo = 'U' or 'u': The upper triangular part of A is stored.
uplo = 'L' or 'l': The lower triangular part of A is stored.

n The order of the matrix A . $n \geq 0$.

a The symmetric or Hermitian matrix A .

If **uplo** = 'U' or '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' or '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 The leading dimension of array **a** in the calling program unit. $lda \geq \max(1, n)$.

Output

a On successful exit, the Cholesky factor U or L overwrites the input.

info Status response:

info = 0: Successful exit.

info < 0: If **info** = $-k$, the k -th argument had an invalid value.

info > 0: If **info** = k , the leading minor of order k is not positive definite, and the factorization could not be completed.

Notes

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

uplo \neq 'L' or 'l' or 'U' or 'u',

n < 0, and

lda < $\max(1, n)$.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding the **uplo** argument as 'Lower' for 'L' or 'Upper' for 'U'.

Invert Positive Definite Matrix**SPOTRI/DPOTRI/CPOTRI/ZPOTRI**

Purpose These subprograms compute the inverse of a real symmetric or complex Hermitian positive definite matrix A using the Cholesky factorization computed by `_POTRF`.

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```
CHARACTER*1 uplo
INTEGER*4   info, lda, n
REAL*4      a(lda, n)
CALL SPOTRI (uplo, n, a, lda, info)
```

```
CHARACTER*1 uplo
INTEGER*4   info, lda, n
REAL*8      a(lda, n)
CALL DPOTRI (uplo, n, a, lda, info)
```

```
CHARACTER*1 uplo
INTEGER*4   info, lda, n
COMPLEX*8   a(lda, n)
CALL CPOTRI (uplo, n, a, lda, info)
```

```
CHARACTER*1 uplo
INTEGER*4   info, lda, n
COMPLEX*16  a(lda, n)
CALL ZPOTRI (uplo, n, a, lda, info)
```

LAPACK8, available on C Series and Exemplar architectures:

```
CHARACTER*1 uplo
INTEGER*8   info, lda, n
REAL*8      a(lda, n)
CALL SPOTRI (uplo, n, a, lda, info)
```

```
CHARACTER*1 uplo
INTEGER*8   info, lda, n
COMPLEX*16  a(lda, n)
CALL CPOTRI (uplo, n, a, lda, info)
```

Input **uplo** Specifies whether the upper or lower triangular factor of the symmetric or Hermitian matrix A is stored, as follows:

uplo = 'U' or 'u': U , the upper triangular factor of A is stored.

uplo = 'L' or 'l': L , the lower triangular factor of A is stored.

n The order of the matrix A . $n \geq 0$.

a The triangular factor U or L from the Cholesky factorization as computed by `_POTRF`.

lda The leading dimension of array **a** in the calling program unit. $lda \geq \max(1, n)$.

Output **a** On successful exit, the upper or lower triangle of the symmetric or Hermitian inverse of A , overwriting the input factor U or L .

info Status response:

info = 0: Successful exit.

info < 0: If **info** = $-k$, the k -th argument had an invalid value.

info > 0: If **info** = k , the (k, k) element of the factor U or L is zero; the matrix is singular and its inverse could not be computed.

Notes

It is almost never necessary to compute the inverse of a matrix. While papers and reference books extensively use the notation " $A^{-1}b$ " to mean "the solution x of the system of linear equations $Ax = b$," it is almost always more efficient and more accurate to compute the solution directly than to invert the coefficient matrix and multiply the inverse times the right hand side vector. This is true even if there are many systems of equations, all using the same coefficient matrix; the matrix may be factored once and the systems may be solved from the factors usually just as efficiently—and more accurately—than by matrix multiplication by the inverse.

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

uplo ≠ 'L' or 'l' or 'U' or 'u',
n < 0, and
lda < max(1,**n**).

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the CALL statement may be improved by coding the **uplo** argument as 'Lower' for 'L' or 'Upper' for 'U'.

Solve Positive Definite Linear System**SPOTRS/DPOTRS/.../ZPOTRS**

Purpose These subprograms solve a system of linear equations $AX = B$ with a real symmetric or complex Hermitian positive definite matrix A using the Cholesky factorization computed by `_POTRF`.

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```
CHARACTER*1 uplo
INTEGER*4   info, lda, ldb, n, nrhs
REAL*4      a(lda, n), b(ldb, nrhs)
CALL SPOTRS (uplo, n, nrhs, a, lda, b, ldb, info)
```

```
CHARACTER*1 uplo
INTEGER*4   info, lda, ldb, n, nrhs
REAL*8      a(lda, n), b(ldb, nrhs)
CALL DPOTRS (uplo, n, nrhs, a, lda, b, ldb, info)
```

```
CHARACTER*1 uplo
INTEGER*4   info, lda, ldb, n, nrhs
COMPLEX*8   a(lda, n), b(ldb, nrhs)
CALL CPOTRS (uplo, n, nrhs, a, lda, b, ldb, info)
```

```
CHARACTER*1 uplo
INTEGER*4   info, lda, ldb, n, nrhs
COMPLEX*16  a(lda, n), b(ldb, nrhs)
CALL ZPOTRS (uplo, n, nrhs, a, lda, b, ldb, info)
```

LAPACK8, available on C Series and Exemplar architectures:

```
CHARACTER*1 uplo
INTEGER*8   info, lda, ldb, n, nrhs
REAL*8      a(lda, n), b(ldb, nrhs)
CALL SPOTRS (uplo, n, nrhs, a, lda, b, ldb, info)
```

```
CHARACTER*1 uplo
INTEGER*8   info, lda, ldb, n, nrhs
COMPLEX*16  a(lda, n), b(ldb, nrhs)
CALL CPOTRS (uplo, n, nrhs, a, lda, b, ldb, info)
```

Input **uplo** Specifies whether the upper or lower triangular factor of the symmetric or Hermitian matrix A is stored, as follows:

uplo = 'U' or 'u': U , the upper triangular factor of A is stored.

uplo = 'L' or 'l': L , the lower triangular factor of A is stored.

n The order of the matrix A . $n \geq 0$.

nrhs The number of right hand sides, that is, the number of columns of the matrix B . $nrhs \geq 0$.

a The triangular factor U or L from the Cholesky factorization as computed by `_POTRF`.

lda The leading dimension of array **a** in the calling program unit. $lda \geq \max(1, n)$.

- b** The n -by- $nrhs$ matrix of right hand side vectors for the system of linear equations $AX = B$.
- ldb** The leading dimension of array **b** in the calling program unit. $ldb \geq \max(1,n)$.
- Output**
- b** On successful exit, the n -by- $nrhs$ matrix of solution vectors X overwrites the input.
- info** Status response:
- info** = 0: Successful exit.
- info** < 0: If **info** = $-k$, the k -th argument had an invalid value.

Notes If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

uplo \neq 'L' or 'l' or 'U' or 'u',
n < 0,
nrhs < 0,
lda < $\max(1,n)$, and
ldb < $\max(1,n)$.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the CALL statement may be improved by coding the **uplo** argument as 'Lower' for 'L' or 'Upper' for 'U'.

Condition Number of Positive Definite Packed Matrix**SPPCON/...**

Purpose These subprograms estimate the reciprocal of the condition number of a real symmetric or complex Hermitian positive definite matrix A that is stored in an array in packed form, using the Cholesky factorization computed by `_PPTRF`.

An estimate is obtained for $\|A^{-1}\|_1$, and the reciprocal of the condition number is computed as $\text{rcond} = (\|A\|_1 \|A^{-1}\|_1)^{-1}$.

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```

CHARACTER*1 uplo
INTEGER*4   info, n
REAL*4     anorm, rcond
INTEGER*4   iwork(n)
REAL*4     ap((n*(n+1))/2), work(3*n)
CALL SPPCON (uplo, n, ap, anorm, rcond, work, iwork, info)

```

```

CHARACTER*1 uplo
INTEGER*4   info, n
REAL*8     anorm, rcond
INTEGER*4   iwork(n)
REAL*8     ap((n*(n+1))/2), work(3*n)
CALL DPPCON (uplo, n, ap, anorm, rcond, work, iwork, info)

```

```

CHARACTER*1 uplo
INTEGER*4   info, n
REAL*4     anorm, rcond
REAL*4     rwork(n)
COMPLEX*8  ap((n*(n+1))/2), work(2*n)
CALL CPPCON (uplo, n, ap, anorm, rcond, work, rwork, info)

```

```

CHARACTER*1 uplo
INTEGER*4   info, n
REAL*8     anorm, rcond
REAL*8     rwork(n)
COMPLEX*16 ap((n*(n+1))/2), work(2*n)
CALL ZPPCON (uplo, n, ap, anorm, rcond, work, rwork, info)

```

LAPACK8, available on C Series and Exemplar architectures:

```

CHARACTER*1 uplo
INTEGER*8   info, n
REAL*8     anorm, rcond
INTEGER*8   iwork(n)
REAL*8     ap((n*(n+1))/2), work(3*n)
CALL SPPCON (uplo, n, ap, anorm, rcond, work, iwork, info)

```

```

CHARACTER*1 uplo
INTEGER*8   info, n
REAL*8     anorm, rcond
REAL*8     rwork(n)
COMPLEX*16 ap((n*(n+1))/2), work(2*n)
CALL CPPCON (uplo, n, ap, anorm, rcond, work, rwork, info)

```

Input	uplo	Specifies whether the upper or lower triangular factor of the symmetric or Hermitian matrix A is stored, as follows: uplo = 'U' or 'u': U , the upper triangular factor of A is stored. uplo = 'L' or 'l': L , the lower triangular factor of A is stored.
	n	The order of the matrix A . $n \geq 0$.
	ap	The triangular factor U or L from the Cholesky factorization as computed by <code>_PPTRF</code> .
	anorm	$\ A\ _1$ ($= \ A\ _\infty$) of the original symmetric or Hermitian matrix A . anorm ≥ 0 .
Working Storage	work, iwork, rwork	Arrays used for work space
Output	rcond	On successful exit, the estimate of the reciprocal condition number of the matrix A , computed as $\mathbf{rcond} = (\ A\ _1 \ A^{-1}\ _1)^{-1}$. If rcond is small enough so that the logical expression $1.0 + \mathbf{rcond} \text{ .EQ. } 1.0$ is true, then A can be regarded as singular to working precision. If rcond is zero, then the companion subprograms for solving and computing the inverse may divide by zero.
	info	Status response: info = 0: Successful exit. info < 0: If info = $-k$, the k -th argument had an invalid value.
Notes		If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are $\begin{aligned} &\mathbf{uplo} \neq \text{'L' or 'l' or 'U' or 'u'}, \\ &\mathbf{n} < 0, \text{ and} \\ &\mathbf{anorm} < 0.0. \end{aligned}$

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the CALL statement may be improved by coding the **uplo** argument as 'Lower' for 'L' or 'Upper' for 'U'.

Factor Positive Definite Packed Matrix**SPPTRF/DPPTRF/.../ZPPTRF**

Purpose These subprograms compute the Cholesky factorization of a real symmetric or complex Hermitian positive definite matrix A that is stored in an array in packed form. A real matrix is symmetric if $A = A^T$, its transpose; a complex matrix is Hermitian if $A = A^*$, its conjugate transpose.

A real symmetric matrix A is positive definite if the quadratic form $x^T A x$ is positive for all nonzero real vectors x ; a complex Hermitian matrix A is positive definite if the quadratic form $x^* A x$ is positive for all nonzero complex vectors x .

Given such a matrix A , these subprograms compute the Cholesky factorization of the form $A = U^* U$ or $A = L L^*$ where U is an upper triangular matrix, L is a lower triangular matrix.

Matrix Storage Because either triangle of A may be obtained from the other, you need only provide one triangle of A , either the upper or the lower triangle. Compared to storing the entire matrix, you save memory by supplying that triangle stored column-by-column in packed form in a 1-dimensional array.

The following examples illustrate the packed storage of symmetric or Hermitian matrices.

Upper triangular storage. If the upper triangle of A is

$$\begin{array}{cccc} 11 & 12 & 13 & 14 \\ & 22 & 23 & 24 \\ & & 33 & 34 \\ & & & 44 \end{array}$$

then A is packed column-by-column into an array **ap** as follows:

k	1	2	3	4	5	6	7	8	9	10
ap(k)	11	12	22	13	23	33	14	24	34	44

Upper triangular matrix element a_{ij} is stored in array element **ap**($i+j \times (j-1)/2$).

Lower triangular storage. If the lower triangle of A is

$$\begin{array}{cccc} 11 & & & \\ 21 & 22 & & \\ 31 & 32 & 33 & \\ 41 & 42 & 43 & 44 \end{array}$$

then A is packed column-by-column into an array **ap** as follows:

k	1	2	3	4	5	6	7	8	9	10
ap(k)	11	21	31	41	22	32	42	33	43	44

Lower triangular matrix element a_{ij} is stored in array element **ap**($i+(j-1) \times (2n-j)/2$).

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```

CHARACTER*1  uplo
INTEGER*4    info, n
REAL*4       ap((n*(n+1))/2)
CALL SPPTRF (uplo, n, ap, info)

```

```

CHARACTER*1  uplo
INTEGER*4    info, n
REAL*8       ap((n*(n+1))/2)
CALL DPPTRF (uplo, n, ap, info)

```

```

CHARACTER*1  uplo
INTEGER*4    info, n
COMPLEX*8   ap((n*(n+1))/2)
CALL CPPTRF (uplo, n, ap, info)

```

```

CHARACTER*1  uplo
INTEGER*4    info, n
COMPLEX*16  ap((n*(n+1))/2)
CALL ZPPTRF (uplo, n, ap, info)

```

LAPACK8, available on C Series and Exemplar architectures:

```

CHARACTER*1  uplo
INTEGER*8    info, n
REAL*8       ap((n*(n+1))/2)
CALL SPPTRF (uplo, n, ap, info)

```

```

CHARACTER*1  uplo
INTEGER*8    info, n
COMPLEX*16  ap((n*(n+1))/2)
CALL CPPTRF (uplo, n, ap, info)

```

Input **uplo** Specifies whether the upper or lower triangular part of the symmetric or Hermitian matrix A is stored, as follows:

uplo = 'U' or 'u': The upper triangular part of A is stored.
uplo = 'L' or 'l': The lower triangular part of A is stored.

n The order of the matrix A . $n \geq 0$.

ap The upper or lower triangular part of the symmetric or Hermitian matrix A , packed columnwise in a linear array as follows:

If **uplo** = 'U' or 'u', $ap(i + (j-1) \times j/2) = A(i, j)$ for $1 \leq i \leq j$;
If **uplo** = 'L' or 'l', $ap(i + (j-1) \times (2n-j)/2) = A(i, j)$ for $j \leq i \leq n$.

Output **ap** On successful exit, the Cholesky factor U or L overwrites the input.

info Status response:

info = 0: Successful exit.
info < 0: If **info** = $-k$, the k -th argument had an invalid value.
info > 0: If **info** = k , the leading minor of order k is not positive definite, and the factorization could not be completed.

Notes

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

uplo ≠ 'L' or 'l' or 'U' or 'u', and
n < 0.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding the **uplo** argument as 'Lower' for 'L' or 'Upper' for 'U'.

Purpose These subprograms compute the inverse of a real symmetric or complex Hermitian positive definite matrix A that is stored in an array in packed form, using the Cholesky factorization computed by `_PPTRF`.

Matrix Storage Because either triangle of A^{-1} may be obtained from that triangle of the Cholesky factorization of A or from the other triangle of A^{-1} , you need only provide one triangle of the factorization, and only the corresponding triangle of A^{-1} is computed. Compared to storing the entire matrix, you save memory by supplying only either the upper or the lower triangle of the factorization of A , stored column-by-column in packed form in a 1-dimensional array.

The following examples illustrate the packed storage of symmetric or Hermitian matrices.

Upper triangular storage. If the upper triangle of A^{-1} is

$$\begin{array}{cccc} 11 & 12 & 13 & 14 \\ & 22 & 23 & 24 \\ & & 33 & 34 \\ & & & 44 \end{array}$$

then $(\alpha_{ij}) = A^{-1}$ is packed column-by-column into an array `ap` as follows:

k	1	2	3	4	5	6	7	8	9	10
<code>ap(k)</code>	11	12	22	13	23	33	14	24	34	44

Upper triangular matrix element α_{ij} is stored in array element `ap(i+j*(j-1)/2)`.

Lower triangular storage. If the lower triangle of A^{-1} is

$$\begin{array}{cccc} 11 & & & \\ 21 & 22 & & \\ 31 & 32 & 33 & \\ 41 & 42 & 43 & 44 \end{array}$$

then $(\alpha_{ij}) = A^{-1}$ is packed column-by-column into an array `ap` as follows:

k	1	2	3	4	5	6	7	8	9	10
<code>ap(k)</code>	11	21	31	41	22	32	42	33	43	44

Lower triangular matrix element α_{ij} is stored in array element `ap(i+(j-1)*(2n-j)/2)`.

Usage

LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```

CHARACTER*1 uplo
INTEGER*4    info, n
REAL*4       ap((n*(n+1))/2)
CALL SPPTRI (uplo, n, ap, info)

```

```

CHARACTER*1 uplo
INTEGER*4    info, n
REAL*8       ap((n*(n+1))/2)
CALL DPPTRI (uplo, n, ap, info)

```

```

CHARACTER*1 uplo
INTEGER*4    info, n
COMPLEX*8    ap((n*(n+1))/2)
CALL CPPTRI (uplo, n, ap, info)

```

```

CHARACTER*1 uplo
INTEGER*4    info, n
COMPLEX*16   ap((n*(n+1))/2)
CALL ZPPTRI (uplo, n, ap, info)

```

LAPACK8, available on C Series and Exemplar architectures:

```

CHARACTER*1 uplo
INTEGER*8    info, n
REAL*8       ap((n*(n+1))/2)
CALL SPPTRI (uplo, n, ap, info)

```

```

CHARACTER*1 uplo
INTEGER*8    info, n
COMPLEX*16   ap((n*(n+1))/2)
CALL CPPTRI (uplo, n, ap, info)

```

Input

uplo Specifies whether the upper or lower triangular factor of the symmetric or Hermitian matrix A is stored, as follows:

uplo = 'U' or 'u': U , the upper triangular factor of A is stored.

uplo = 'L' or 'l': L , the lower triangular factor of A is stored.

n The order of the matrix A . $n \geq 0$.

ap The triangular factor U or L from the Cholesky factorization as computed by `_PPTRF`.

Output

ap On successful exit, the upper or lower triangle of the symmetric or Hermitian inverse of A overwrites the input, as follows:

If **uplo = 'U' or 'u'**, $ap(i + (j-1) \times j/2) = A^{-1}(i, j)$ for $1 \leq i \leq j$;

If **uplo = 'L' or 'l'**, $ap(i + (j-1) \times (2n-j)/2) = A^{-1}(i, j)$ for $j \leq i \leq n$.

info Status response:

info = 0: Successful exit.

info < 0: If **info** = $-k$, the k -th argument had an invalid value.

info > 0: If **info** = k , the (k, k) element of the factor U or L is zero; the matrix is singular and its inverse could not be computed.

Notes

It is almost never necessary to compute the inverse of a matrix. While papers and reference books extensively use the notation " $A^{-1}b$ " to mean "the solution x of the system of linear equations $Ax = b$," it is almost always more efficient and more accurate to compute the solution directly than to invert the coefficient matrix and multiply the inverse times the right hand side vector. This is true even if there are many systems of equations, all using the same coefficient matrix; the matrix may be factored once and the systems may be solved from the factors usually just as efficiently—and more accurately—than by matrix multiplication by the inverse.

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

uplo ≠ 'L' or 'l' or 'U' or 'u',
n < 0.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding the **uplo** argument as 'Lower' for 'L' or 'Upper' for 'U'.

Solve Positive Definite Packed Linear System**SPPTRS/.../ZPPTRS**

Purpose These subprograms solve a system of linear equations $AX = B$ with a real symmetric or complex Hermitian positive definite matrix A that is stored in an array in packed form, using the Cholesky factorization computed by `_PPTRF`.

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```

CHARACTER*1 uplo
INTEGER*4 info, ldb, n, nrhs
REAL*4 ap((n*(n+1))/2), b(ldb, nrhs)
CALL SPPTRS (uplo, n, nrhs, ap, b, ldb, info)

```

```

CHARACTER*1 uplo
INTEGER*4 info, ldb, n, nrhs
REAL*8 ap((n*(n+1))/2), b(ldb, nrhs)
CALL DPPTRS (uplo, n, nrhs, ap, b, ldb, info)

```

```

CHARACTER*1 uplo
INTEGER*4 info, ldb, n, nrhs
COMPLEX*8 ap((n*(n+1))/2), b(ldb, nrhs)
CALL CPPTRS (uplo, n, nrhs, ap, b, ldb, info)

```

```

CHARACTER*1 uplo
INTEGER*4 info, ldb, n, nrhs
COMPLEX*16 ap((n*(n+1))/2), b(ldb, nrhs)
CALL ZPPTRS (uplo, n, nrhs, ap, b, ldb, info)

```

LAPACK8, available on C Series and Exemplar architectures:

```

CHARACTER*1 uplo
INTEGER*8 info, ldb, n, nrhs
REAL*8 ap((n*(n+1))/2), b(ldb, nrhs)
CALL SPPTRS (uplo, n, nrhs, ap, b, ldb, info)

```

```

CHARACTER*1 uplo
INTEGER*8 info, ldb, n, nrhs
COMPLEX*16 ap((n*(n+1))/2), b(ldb, nrhs)
CALL CPPTRS (uplo, n, nrhs, ap, b, ldb, info)

```

Input **uplo** Specifies whether the upper or lower triangular factor of the symmetric or Hermitian matrix A is stored, as follows:

uplo = 'U' or 'u': U , the upper triangular factor of A is stored.

uplo = 'L' or 'l': L , the lower triangular factor of A is stored.

n The order of the matrix A . $n \geq 0$.

nrhs The number of right hand sides, that is, the number of columns of the matrix B . $nrhs \geq 0$.

ap The triangular factor U or L from the Cholesky factorization as computed by `_PPTRF`.

b The n -by- $nrhs$ matrix of right hand side vectors for the system of linear equations $AX = B$.

ldb The leading dimension of array **b** in the calling program unit. $ldb \geq \max(1,n)$.

Output

b On successful exit, the **n**-by-**nrhs** matrix of solution vectors *X* overwrites the input.

info Status response:

info = 0: Successful exit.

info < 0: If **info** = $-k$, the *k*-th argument had an invalid value.

Notes If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

uplo ≠ 'L' or 'l' or 'U' or 'u',
n < 0,
nrhs < 0, and
ldb < $\max(1,n)$.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding the **uplo** argument as 'Lower' for 'L' or 'Upper' for 'U'.

Condition Number of Positive Definite Tridiagonal Matrix**SPTCON/...**

Purpose These subprograms estimate the reciprocal of the condition number of a real symmetric or complex Hermitian positive definite tridiagonal matrix A using the Cholesky factorization computed by `_PTTRF`.

An estimate is obtained for $\|A^{-1}\|$, and the reciprocal of the condition number is computed as $\text{rcond} = (\|A\|_1 \|A^{-1}\|_1)^{-1}$.

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```
INTEGER*4 info, n
REAL*4    anorm, rcond
REAL*4    d(n), e(n-1), work(n)
CALL SPTCON (n, d, e, anorm, rcond, work, info)
```

```
INTEGER*4 info, n
REAL*8    anorm, rcond
REAL*8    d(n), e(n-1), work(n)
CALL DPTCON (n, d, e, anorm, rcond, work, info)
```

```
INTEGER*4 info, n
REAL*4    anorm, rcond
REAL*4    d(n), rwork(n)
COMPLEX*8 e(n-1)
CALL CPTCON (n, d, e, anorm, rcond, rwork, info)
```

```
INTEGER*4 info, n
REAL*8    anorm, rcond
REAL*8    d(n), rwork(n)
COMPLEX*16 e(n-1)
CALL ZPTCON (n, d, e, anorm, rcond, rwork, info)
```

LAPACK8, available on C Series and Exemplar architectures:

```
INTEGER*8 info, n
REAL*8    anorm, rcond
REAL*8    d(n), e(n-1), work(n)
CALL SPTCON (n, d, e, anorm, rcond, work, info)
```

```
INTEGER*8 info, n
REAL*8    anorm, rcond
REAL*8    d(n), rwork(n)
COMPLEX*16 e(n-1)
CALL CPTCON (n, d, e, anorm, rcond, rwork, info)
```

Input

- n** The order of the matrix A . $n \geq 0$.
- d** The n diagonal elements of the diagonal matrix D from the LDL^* factorization of A .
- e** The $n-1$ subdiagonal elements of the unit bidiagonal factor L from the LDL^* factorization of A . e can also be regarded as the superdiagonal of the unit bidiagonal factor U from the U^*DU factorization of A .

	anorm	$\ A\ _1$ ($= \ A\ _\infty$) of the original symmetric or Hermitian tridiagonal matrix A . $\text{anorm} \geq 0$.
Working Storage	work, rwork	Arrays used for work space.
Output	rcond	On successful exit, the estimate of the reciprocal condition number of the matrix A , computed as $\text{rcond} = (\ A\ _1 \ A^{-1}\ _1)^{-1}$. If rcond is small enough so that the logical expression

$$1.0 + \text{rcond} \text{ .EQ. } 1.0$$

is true, then A can be regarded as singular to working precision. If **rcond** is zero, then the companion subprograms for solving and computing the inverse may divide by zero.

info Status response:

info = 0: Successful exit.

info < 0: If **info** = $-k$, the k -th argument had an invalid value.

Notes If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

$n < 0$, and
 $\text{anorm} < 0.0$.

Factor Positive Definite Tridiagonal Matrix**SPTTRF/.../ZPTTRF**

Purpose These subprograms compute the Cholesky factorization of a real symmetric or complex Hermitian positive definite tridiagonal matrix A . A real matrix is symmetric if $A = A^T$, its transpose; a complex matrix is Hermitian if $A = A^*$, its conjugate transpose.

A real symmetric matrix A is positive definite if the quadratic form $x^T Ax$ is positive for all nonzero real vectors x ; a complex Hermitian matrix A is positive definite if the quadratic form $x^* Ax$ is positive for all nonzero complex vectors x .

A matrix $A = (a_{ij})$ is tridiagonal if its nonzero elements lie only on the principal diagonal ($i = j$), the subdiagonal ($i = j+1$), and the superdiagonal ($i = j-1$) of the matrix.

Given such a matrix A , these subprograms compute the Cholesky factorization of the form $A = LDL^*$ where L is a unit lower bidiagonal matrix and D is a diagonal matrix. Alternatively, the factorization can be viewed as $A = U^*DU$.

Matrix Storage The following example illustrates the storage of symmetric or Hermitian tridiagonal matrices. Consider the following symmetric tridiagonal matrix of order $n = 7$:

11	21	0	0	0	0	0
21	22	32	0	0	0	0
0	32	33	43	0	0	0
0	0	43	44	54	0	0
0	0	0	54	55	65	0
0	0	0	0	65	66	76
0	0	0	0	0	76	77

The subdiagonal is stored in array e and the principal diagonal is stored in array d , as follows:

i	$e(i)$	$d(i)$
1	21	11
2	32	22
3	43	33
4	54	44
5	65	55
6	76	66
7		77

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```

INTEGER*4 info, n
REAL*4 d(n), e(n-1)
CALL SPTTRF (n, d, e, info)

```

```

INTEGER*4 info, n
REAL*8 d(n), e(n-1)
CALL DPTTRF (n, d, e, info)

```

```

INTEGER*4 info, n
REAL*4 d(n)
COMPLEX*8 e(n-1)
CALL CPTTRF (n, d, e, info)

```

```

INTEGER*4  info, n
REAL*8     d(n)
COMPLEX*16 e(n-1)
CALL ZPTTRF (n, d, e, info)

```

LAPACK8, available on C Series and Exemplar architectures:

```

INTEGER*8  info, n
REAL*8     d(n), e(n-1)
CALL SPTTRF (n, d, e, info)

```

```

INTEGER*8  info, n
REAL*8     d(n)
COMPLEX*16 e(n-1)
CALL CPTTRF (n, d, e, info)

```

Input	n	The order of the matrix A . $n \geq 0$.
	d	The n diagonal elements of the tridiagonal matrix A .
	e	The $n-1$ subdiagonal elements of the tridiagonal matrix A .
Output	d	On successful exit, the n diagonal elements of the diagonal matrix D from the LDL^* factorization of A .
	e	On successful exit, the $n-1$ subdiagonal elements of the unit lower bidiagonal factor L from the LDL^* factorization of A . e can also be regarded as the superdiagonal of the unit upper bidiagonal factor U from the U^*DU factorization of A .
	info	Status response: info = 0: Successful exit. info < 0: If info = $-k$, the k -th argument had an invalid value. info > 0: If info = k , the leading minor of order k is not positive definite; if $k < n$, the factorization could not be completed, while if $k = n$, the factorization was completed but $D(n) = 0$.

Notes If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

$n < 0$.

Solve Positive Definite Tridiagonal System**SPTTRS/.../ZPTTRS**

Purpose These subprograms solve a system of linear equations $AX = B$ with a real symmetric or complex Hermitian positive definite tridiagonal matrix A using the Cholesky factorization computed by `_PTTRF`.

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```
INTEGER*4 info, ldb, n, nrhs
REAL*4    b(ldb, nrhs), d(n), e(n-1)
CALL SPTTRS (n, nrhs, d, e, b, ldb, info)
```

```
INTEGER*4 info, ldb, n, nrhs
REAL*8    b(ldb, nrhs), d(n), e(n-1)
CALL DPTTRS (n, nrhs, d, e, b, ldb, info)
```

```
CHARACTER*1 uplo
INTEGER*4    info, ldb, n, nrhs
REAL*4      d(n)
COMPLEX*8   b(ldb, nrhs), e(n-1)
CALL CPTTRS (uplo, n, nrhs, d, e, b, ldb, info)
```

```
CHARACTER*1 uplo
INTEGER*4    info, ldb, n, nrhs
REAL*8      d(n)
COMPLEX*16  b(ldb, nrhs), e(n-1)
CALL ZPTTRS (uplo, n, nrhs, d, e, b, ldb, info)
```

LAPACK8, available on C Series and Exemplar architectures:

```
INTEGER*8 info, ldb, n, nrhs
REAL*8    b(ldb, nrhs), d(n), e(n-1)
CALL SPTTRS (n, nrhs, d, e, b, ldb, info)
```

```
CHARACTER*1 uplo
INTEGER*8    info, ldb, n, nrhs
REAL*8      d(n)
COMPLEX*16  b(ldb, nrhs), e(n-1)
CALL CPTTRS (uplo, n, nrhs, d, e, b, ldb, info)
```

Input **uplo** Specifies the form of the factorization and whether the array **e** contains the superdiagonal of the upper bidiagonal factor U or the subdiagonal of the lower bidiagonal factor L , as follows:

uplo = 'U' or 'u': $A = U^T D U$ and **e** is the superdiagonal of U .

uplo = 'L' or 'l': $A = L D L^T$ and **e** is the superdiagonal of L .

n The order of the matrix A . $n \geq 0$.

nrhs The number of right hand sides, that is, the number of columns of the matrix B . $nrhs \geq 0$.

d The n diagonal elements of the diagonal matrix D from the LDL^* factorization of A .

e The $n-1$ subdiagonal elements of the unit bidiagonal factor L from the LDL^* factorization of A . **e** can also be regarded as the superdiagonal of the unit bidiagonal factor U from the $U^* D U$ factorization of A .

	b	The n -by- nrhs matrix of right hand side vectors for the system of linear equations $AX = B$.
	ldb	The leading dimension of array b in the calling program unit. $ldb \geq \max(1,n)$.
Output	b	On successful exit, the n -by- nrhs matrix of solution vectors X overwrites the input.
	info	Status response: info = 0: Successful exit. info < 0: If info = $-k$, the k -th argument had an invalid value.

Notes If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

uplo ≠ 'L' or 'l' or 'U' or 'u',
n < 0,
nrhs < 0, and
ldb < $\max(1,n)$.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the CALL statement may be improved by coding the **uplo** argument as 'Lower' for 'L' or 'Upper' for 'U'.

Condition Number of Symmetric or Hermitian Packed Matrix**SSPCON/...**

Purpose These subprograms estimate the reciprocal of the condition number of a real or complex symmetric or complex Hermitian matrix A that is stored in an array in packed form, using the Bunch-Kaufman factorization computed by `_SPTRF` or `_HPTRF`.

The structure of A is indicated by the name of the subprogram used:

SSPCON or DSPCON A is a real symmetric packed matrix.
 CSPCON or ZSPCON A is a complex symmetric packed matrix.
 CHPCON or ZHPCON A is a complex Hermitian packed matrix.

An estimate is obtained for $\|A^{-1}\|$, and the reciprocal of the condition number is computed as $\text{rcond} = (\|A\| \|A^{-1}\|)^{-1}$.

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```

CHARACTER*1 uplo
INTEGER*4   info, n
REAL*4     anorm, rcond
INTEGER*4   ipiv(n), iwork(n)
REAL*4     ap((n*(n+1))/2), work(2*n)
CALL SSPCON (uplo, n, ap, ipiv, anorm, rcond, work, iwork, info)

```

```

CHARACTER*1 uplo
INTEGER*4   info, n
REAL*8     anorm, rcond
INTEGER*4   ipiv(n), iwork(n)
REAL*8     ap((n*(n+1))/2), work(2*n)
CALL DSPCON (uplo, n, ap, ipiv, anorm, rcond, work, iwork, info)

```

```

CHARACTER*1 uplo
INTEGER*4   info, n
REAL*4     anorm, rcond
INTEGER*4   ipiv(n)
COMPLEX*8  ap((n*(n+1))/2), work(2*n)
CALL CHPCON (uplo, n, ap, ipiv, anorm, rcond, work, info)

```

```

CHARACTER*1 uplo
INTEGER*4   info, n
REAL*4     anorm, rcond
INTEGER*4   ipiv(n)
COMPLEX*8  ap((n*(n+1))/2), work(2*n)
CALL CSPCON (uplo, n, ap, ipiv, anorm, rcond, work, info)

```

```

CHARACTER*1 uplo
INTEGER*4   info, n
REAL*8     anorm, rcond
INTEGER*4   ipiv(n)
COMPLEX*16 ap((n*(n+1))/2), work(2*n)
CALL ZHPCON (uplo, n, ap, ipiv, anorm, rcond, work, info)

```

```

CHARACTER*1 uplo
INTEGER*4 info, n
REAL*8 anorm, rcond
INTEGER*4 ipiv(n)
COMPLEX*16 ap((n*(n+1))/2), work(2*n)
CALL ZSPCON (uplo, n, ap, ipiv, anorm, rcond, work, info)

```

LAPACK8, available on C Series and Exemplar architectures:

```

CHARACTER*1 uplo
INTEGER*8 info, n
REAL*8 anorm, rcond
INTEGER*8 ipiv(n), iwork(n)
REAL*8 ap((n*(n+1))/2), work(2*n)
CALL SSPCON (uplo, n, ap, ipiv, anorm, rcond, work, iwork, info)

```

```

CHARACTER*1 uplo
INTEGER*8 info, n
REAL*8 anorm, rcond
INTEGER*8 ipiv(n)
COMPLEX*16 ap((n*(n+1))/2), work(2*n)
CALL CHPCON (uplo, n, ap, ipiv, anorm, rcond, work, info)

```

```

CHARACTER*1 uplo
INTEGER*8 info, n
REAL*8 anorm, rcond
INTEGER*8 ipiv(n)
COMPLEX*16 ap((n*(n+1))/2), work(2*n)
CALL CSPCON (uplo, n, ap, ipiv, anorm, rcond, work, info)

```

Input	uplo	Specifies whether the upper or lower triangular factor of the symmetric or Hermitian matrix A is stored, as follows: uplo = 'U' or 'u': The upper triangular factor of A is stored. uplo = 'L' or 'l': The lower triangular factor of A is stored.
	n	The order of the matrix A . $n \geq 0$.
	ap	The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by <code>_SPTRF</code> or <code>_HPTRF</code> .
	ipiv	Details of the interchanges and the block structure of D as determined by <code>_SPTRF</code> or <code>_HPTRF</code> .
	anorm	$\ A\ _1$ ($= \ A\ _\infty$) of the original symmetric or Hermitian matrix A . anorm ≥ 0 .
Working Storage	work, iwork	Arrays used for work space.

Output **rcond** On successful exit, the estimate of the reciprocal condition number of the matrix A , computed as $\mathbf{rcond} = (\|A\|_1 \|A^{-1}\|_1)^{-1}$. If **rcond** is small enough so that the logical expression

$$1.0 + \mathbf{rcond} \text{ .EQ. } 1.0$$

is true, then A can be regarded as singular to working precision. If **rcond** is zero, then the companion subprograms for solving and computing the inverse may divide by zero.

info Status response:

info = 0: Successful exit.

info < 0: If **info** = $-k$, the k -th argument had an invalid value.

Notes If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

uplo \neq 'L' or 'l' or 'U' or 'u',
n < 0, and
anorm < 0.0.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding the **uplo** argument as 'Lower' for 'L' or 'Upper' for 'U'.

Purpose

These subprograms compute the factorization of a real or complex symmetric or complex Hermitian matrix A that is stored in an array in packed form, using the Bunch-Kaufman diagonal pivoting method. If A is known to be positive definite, other subprograms in this chapter, specifically designed for positive definite matrices, will be more efficient than these subprograms.

A matrix is symmetric if $A = A^T$, its transpose; a matrix is Hermitian if $A = A^*$, its conjugate transpose.

The structure of A is indicated by the name of the subprogram used:

SSPTRF or DSPTRF A is a real symmetric packed matrix.
 CSPTRF or ZSPTRF A is a complex symmetric packed matrix.
 CHPTRF or ZHPTRF A is a complex Hermitian packed matrix.

If A is real or complex symmetric, the factorization has the form $A = UDU^T$ or $A = LDL^T$ where U is a product of permutation and unit upper triangular matrices, L is a product of permutation and unit lower triangular matrices, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

If A is complex Hermitian, the factorization has the form $A = UDU^*$ or $A = LDL^*$ where U and L are as above, and D is Hermitian and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

Matrix Storage Because either triangle of A may be obtained from the other, you need only provide one triangle of A , either the upper or the lower triangle. Compared to storing the entire matrix, you save memory by supplying that triangle stored column-by-column in packed form in a 1-dimensional array.

The following examples illustrate the packed storage of symmetric or Hermitian matrices.

Upper triangular storage. If the upper triangle of A is

11	12	13	14
	22	23	24
		33	34
			44

then A is packed column-by-column into an array **ap** as follows:

k	1	2	3	4	5	6	7	8	9	10
ap (k)	11	12	22	13	23	33	14	24	34	44

Upper triangular matrix element a_{ij} is stored in array element **ap**($i+j \times (j-1)/2$).

Lower triangular storage. If the lower triangle of A is

11									
21	22								
31	32	33							
41	42	43	44						

then A is packed column-by-column into an array ap as follows:

k	1	2	3	4	5	6	7	8	9	10
$ap(k)$	11	21	31	41	22	32	42	33	43	44

Lower triangular matrix element a_{ij} is stored in array element $ap(i+(j-1) \times (2n-j)/2)$.

Usage

LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```

CHARACTER*1 uplo
INTEGER*4   info, n
INTEGER*4   ipiv(n)
REAL*4      ap((n*(n+1))/2)
CALL SSPTRF (uplo, n, ap, ipiv, info)

```

```

CHARACTER*1 uplo
INTEGER*4   info, n
INTEGER*4   ipiv(n)
REAL*8      ap((n*(n+1))/2)
CALL DSPTRF (uplo, n, ap, ipiv, info)

```

```

CHARACTER*1 uplo
INTEGER*4   info, n
INTEGER*4   ipiv(n)
COMPLEX*8   ap((n*(n+1))/2)
CALL CHPTRF (uplo, n, ap, ipiv, info)

```

```

CHARACTER*1 uplo
INTEGER*4   info, n
INTEGER*4   ipiv(n)
COMPLEX*8   ap((n*(n+1))/2)
CALL CSPTRF (uplo, n, ap, ipiv, info)

```

```

CHARACTER*1 uplo
INTEGER*4   info, n
INTEGER*4   ipiv(n)
COMPLEX*16  ap((n*(n+1))/2)
CALL ZHPTRF (uplo, n, ap, ipiv, info)

```

```

CHARACTER*1 uplo
INTEGER*4   info, n
INTEGER*4   ipiv(n)
COMPLEX*16  ap((n*(n+1))/2)
CALL ZSPTRF (uplo, n, ap, ipiv, info)

```

LAPACK8, available on C Series and Exemplar architectures:

```

CHARACTER*1 uplo
INTEGER*8 info, n
INTEGER*8 ipiv(n)
REAL*8 ap((n*(n+1))/2)
CALL SSPTRF (uplo, n, ap, ipiv, info)

```

```

CHARACTER*1 uplo
INTEGER*8 info, n
INTEGER*8 ipiv(n)
COMPLEX*16 ap((n*(n+1))/2)
CALL CHPTRF (uplo, n, ap, ipiv, info)

```

```

CHARACTER*1 uplo
INTEGER*8 info, n
INTEGER*8 ipiv(n)
COMPLEX*16 ap((n*(n+1))/2)
CALL CSPTRF (uplo, n, ap, ipiv, info)

```

Input	uplo	<p>Specifies whether the upper or lower triangular part of the symmetric or Hermitian matrix A is stored, as follows:</p> <p>uplo = 'U' or 'u': The upper triangular part of A is stored. uplo = 'L' or 'l': The lower triangular part of A is stored.</p>
	n	The order of the matrix A . $n \geq 0$.
	ap	<p>The upper or lower triangular part of the symmetric or Hermitian matrix A, packed columnwise in a linear array as follows:</p> <p>If uplo = 'U' or 'u', $ap(i + (j-1) \times j/2) = A(i, j)$ for $1 \leq i \leq j$;</p> <p>If uplo = 'L' or 'l', $ap(i + (j-1) \times (2n-j)/2) = A(i, j)$ for $j \leq i \leq n$.</p>
Output	ap	On successful exit, the block diagonal matrix D and the multipliers used to obtain the factor U or L overwrites the input.
	ipiv	<p>On successful exit, details of the interchanges and the block structure of D:</p> <p>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.</p> <p>If uplo = 'U' or '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.</p> <p>If uplo = 'L' or '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.</p>

info Status response:

info = 0: Successful exit.

info < 0: If **info** = $-k$, the k -th argument had an invalid value.

info > 0: If **info** = k , $D(k,k)$ is zero. The factorization has been completed, but the block diagonal matrix D is singular, and division by zero will occur if it is used to solve a system of equations.

Notes

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

uplo \neq 'L' or 'l' or 'U' or 'u', and
n < 0.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding the **uplo** argument as 'Lower' for 'L' or 'Upper' for 'U'.

Purpose These subprograms compute the inverse of a real or complex symmetric or complex Hermitian matrix A that is stored in an array in packed form, using the Bunch-Kaufman factorization computed by `_SPTRF` or `_HPTRF`.

The structure of A is indicated by the name of the subprogram used:

SSPTRI or DSPTRI A is a real symmetric matrix.
 CSPTRI or ZSPTRI A is a complex symmetric matrix.
 CHPTRI or ZHPTRI A is a complex Hermitian matrix.

Matrix Storage Because either triangle of A^{-1} may be obtained from that triangle of the Bunch-Kaufman factorization of A or from the other triangle of A^{-1} , you need only provide one triangle of the factorization, and only the corresponding triangle of A^{-1} is computed. Compared to storing the entire matrix, you save memory by supplying only either the upper or the lower triangle of the factorization of A , stored column-by-column in packed form in a 1-dimensional array.

The following examples illustrate the packed storage of symmetric or Hermitian matrices.

Upper triangular storage. If the upper triangle of A^{-1} is

11	12	13	14
	22	23	24
		33	34
			44

then $(\alpha_{ij}) = A^{-1}$ is packed column-by-column into an array `ap` as follows:

k	1	2	3	4	5	6	7	8	9	10
<code>ap(k)</code>	11	12	22	13	23	33	14	24	34	44

Upper triangular matrix element α_{ij} is stored in array element `ap(i+j*(j-1)/2)`.

Lower triangular storage. If the lower triangle of A^{-1} is

11										
21	22									
31	32	33								
41	42	43	44							

then $(\alpha_{ij}) = A^{-1}$ is packed column-by-column into an array `ap` as follows:

k	1	2	3	4	5	6	7	8	9	10
<code>ap(k)</code>	11	21	31	41	22	32	42	33	43	44

Lower triangular matrix element α_{ij} is stored in array element `ap(i+(j-1)*(2n-j)/2)`.

Usage

LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```

CHARACTER*1 uplo
INTEGER*4    info, n
INTEGER*4    ipiv(n)
REAL*4       ap((n*(n+1))/2), work(n)
CALL SSPTRI (uplo, n, ap, ipiv, work, info)

```

```

CHARACTER*1 uplo
INTEGER*4    info, n
INTEGER*4    ipiv(n)
REAL*8       ap((n*(n+1))/2), work(n)
CALL DSPTRI (uplo, n, ap, ipiv, work, info)

```

```

CHARACTER*1 uplo
INTEGER*4    info, n
INTEGER*4    ipiv(n)
COMPLEX*8    ap((n*(n+1))/2), work(n)
CALL CHPTRI (uplo, n, ap, ipiv, work, info)

```

```

CHARACTER*1 uplo
INTEGER*4    info, n
INTEGER*4    ipiv(n)
COMPLEX*8    ap((n*(n+1))/2), work(n)
CALL CSPTRI (uplo, n, ap, ipiv, work, info)

```

```

CHARACTER*1 uplo
INTEGER*4    info, n
INTEGER*4    ipiv(n)
COMPLEX*16   ap((n*(n+1))/2), work(n)
CALL ZHPTRI (uplo, n, ap, ipiv, work, info)

```

```

CHARACTER*1 uplo
INTEGER*4    info, n
INTEGER*4    ipiv(n)
COMPLEX*16   ap((n*(n+1))/2), work(n)
CALL ZSPTRI (uplo, n, ap, ipiv, work, info)

```

LAPACK8, available on C Series and Exemplar architectures:

```

CHARACTER*1 uplo
INTEGER*8    info, n
INTEGER*8    ipiv(n)
REAL*8       ap((n*(n+1))/2), work(n)
CALL SSPTRI (uplo, n, ap, ipiv, work, info)

```

```

CHARACTER*1 uplo
INTEGER*8    info, n
INTEGER*8    ipiv(n)
COMPLEX*16   ap((n*(n+1))/2), work(n)
CALL CHPTRI (uplo, n, ap, ipiv, work, info)

```

```

CHARACTER*1 uplo
INTEGER*8    info, n
INTEGER*8    ipiv(n)
COMPLEX*16   ap((n*(n+1))/2), work(n)
CALL CSPTRI (uplo, n, ap, ipiv, work, info)

```

Input	uplo	Specifies whether the upper or lower triangular factor of the symmetric or Hermitian matrix A is stored, as follows: uplo = 'U' or 'u': The upper triangular factor of A is stored. uplo = 'L' or 'l': The lower triangular factor of A is stored.
	n	The order of the matrix A . $n \geq 0$.
	ap	The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by <code>_SPTRF</code> or <code>_HPTRF</code> .
	ipiv	Details of the interchanges and the block structure of D as determined by <code>_SPTRF</code> or <code>_HPTRF</code> .
Working Storage	work	An array used for work space.
Output	ap	On successful exit, the upper or lower triangle of the symmetric or Hermitian inverse of A overwrites the input, as follows: If uplo = 'U' or 'u', $ap(i + (j-1) \times j/2) = A^{-1}(i, j)$ for $1 \leq i \leq j$; If uplo = 'L' or 'l', $ap(i + (j-1) \times (2n-j)/2) = A^{-1}(i, j)$ for $j \leq i \leq n$.
	info	Status response: info = 0: Successful exit. info < 0: If info = $-k$, the k -th argument had an invalid value. info > 0: If info = k , $D(k, k)$ is zero; the matrix is singular and its inverse could not be computed.

Notes **It is almost never necessary to compute the inverse of a matrix.** While papers and reference books extensively use the notation " $A^{-1}b$ " to mean "the solution x of the system of linear equations $Ax = b$," it is almost always more efficient and more accurate to compute the solution directly than to invert the coefficient matrix and multiply the inverse times the right hand side vector. This is true even if there are many systems of equations, all using the same coefficient matrix; the matrix may be factored once and the systems may be solved from the factors usually just as efficiently—and more accurately—than by matrix multiplication by the inverse.

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

uplo ≠ 'L' or 'l' or 'U' or 'u',
n < 0.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding the **uplo** argument as 'Lower' for 'L' or 'Upper' for 'U'.

Solve Symmetric or Hermitian Packed System**SSPTRS/.../ZSPTRS**

Purpose These subprograms solve a system of linear equations $AX = B$ with a real or complex symmetric or complex Hermitian matrix A that is stored in an array in packed form, using the Bunch-Kaufman factorization computed by `_SPTRF` or `_HPTRF`.

The structure of A is indicated by the name of the subprogram used:

SSPTRS or DSPTRS A is a real symmetric packed matrix.
 CSPTRS or ZSPTRS A is a complex symmetric packed matrix.
 CHETRS or ZHETRS A is a complex Hermitian matrix.

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```
CHARACTER*1 uplo
INTEGER*4    info, ldb, n, nrhs
INTEGER*4    ipiv(n)
REAL*4      ap((n*(n+1))/2), b(ldb, nrhs)
CALL SSPTRS (uplo, n, nrhs, ap, ipiv, b, ldb, info)
```

```
CHARACTER*1 uplo
INTEGER*4    info, ldb, n, nrhs
INTEGER*4    ipiv(n)
REAL*8      ap((n*(n+1))/2), b(ldb, nrhs)
CALL DSPTRS (uplo, n, nrhs, ap, ipiv, b, ldb, info)
```

```
CHARACTER*1 uplo
INTEGER*4    info, ldb, n, nrhs
INTEGER*4    ipiv(n)
COMPLEX*8   ap((n*(n+1))/2), b(ldb, nrhs)
CALL CHPTRS (uplo, n, nrhs, ap, ipiv, b, ldb, info)
```

```
CHARACTER*1 uplo
INTEGER*4    info, ldb, n, nrhs
INTEGER*4    ipiv(n)
COMPLEX*8   ap((n*(n+1))/2), b(ldb, nrhs)
CALL CSPTRS (uplo, n, nrhs, ap, ipiv, b, ldb, info)
```

```
CHARACTER*1 uplo
INTEGER*4    info, ldb, n, nrhs
INTEGER*4    ipiv(n)
COMPLEX*16  ap((n*(n+1))/2), b(ldb, nrhs)
CALL ZHPTRS (uplo, n, nrhs, ap, ipiv, b, ldb, info)
```

```
CHARACTER*1 uplo
INTEGER*4    info, ldb, n, nrhs
INTEGER*4    ipiv(n)
COMPLEX*16  ap((n*(n+1))/2), b(ldb, nrhs)
CALL ZSPTRS (uplo, n, nrhs, ap, ipiv, b, ldb, info)
```

LAPACK8, available on C Series and Exemplar architectures:

```
CHARACTER*1 uplo
INTEGER*8    info, ldb, n, nrhs
INTEGER*8    ipiv(n)
REAL*8      ap((n*(n+1))/2), b(ldb, nrhs)
CALL SSPTRS (uplo, n, nrhs, ap, ipiv, b, ldb, info)
```

```

CHARACTER*1 uplo
INTEGER*8 info, ldb, n, nrhs
INTEGER*8 ipiv(n)
COMPLEX*16 ap((n*(n+1))/2), b(ldb, nrhs)
CALL CHPTRS (uplo, n, nrhs, ap, ipiv, b, ldb, info)

```

```

CHARACTER*1 uplo
INTEGER*8 info, ldb, n, nrhs
INTEGER*8 ipiv(n)
COMPLEX*16 ap((n*(n+1))/2), b(ldb, nrhs)
CALL CSPTRS (uplo, n, nrhs, ap, ipiv, b, ldb, info)

```

Input	uplo	Specifies whether the upper or lower triangular factor of the symmetric or Hermitian matrix A is stored, as follows: uplo = 'U' or 'u': The upper triangular factor of A is stored. uplo = 'L' or 'l': The lower triangular factor of A is stored.
	n	The order of the matrix A . $n \geq 0$.
	nrhs	The number of right hand sides, that is, the number of columns of the matrix B . nrhs ≥ 0 .
	ap	The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by <code>_SPTRF</code> or <code>_HPTRF</code> .
	ipiv	Details of the interchanges and the block structure of D as determined by <code>_SPTRF</code> or <code>_HPTRF</code> .
	b	The n -by- nrhs matrix of right hand side vectors for the system of linear equations $AX = B$.
	ldb	The leading dimension of array b in the calling program unit. ldb $\geq \max(1, n)$.
Output	b	On successful exit, the n -by- nrhs matrix of solution vectors X overwrites the input.
	info	Status response: info = 0: Successful exit. info < 0: If info = $-k$, the k -th argument had an invalid value.

Notes If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

```

uplo  $\neq$  'L' or 'l' or 'U' or 'u',
n < 0,
nrhs < 0, and
ldb < max(1, n).

```

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding the **uplo** argument as 'Lower' for 'L' or 'Upper' for 'U'.

Condition Number of Symmetric or Hermitian Matrix**SSYCON/...**

Purpose These subprograms estimate the reciprocal of the condition number of a real or complex symmetric or complex Hermitian matrix A using the Bunch-Kaufman factorization computed by `_SYTRF` or `_HETRF`.

The structure of A is indicated by the name of the subprogram used:

SSYCON or DSYCON A is a real symmetric matrix.
 CSYCON or ZSYCON A is a complex symmetric matrix.
 CHECON or ZHECON A is a complex Hermitian matrix.

An estimate is obtained for $\|A^{-1}\|$, and the reciprocal of the condition number is computed as $\text{rcond} = (\|A\| \|A^{-1}\|)^{-1}$.

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```

CHARACTER*1 uplo
INTEGER*4   info, lda, n
REAL*4      anorm, rcond
INTEGER*4   ipiv(n), iwork(n)
REAL*4      a(lda, n), work(2*n)
CALL SSYCON (uplo, n, a, lda, ipiv, anorm, rcond, work, iwork,
            info)

```

```

CHARACTER*1 uplo
INTEGER*4   info, lda, n
REAL*8      anorm, rcond
INTEGER*4   ipiv(n), iwork(n)
REAL*8      a(lda, n), work(2*n)
CALL DSYCON (uplo, n, a, lda, ipiv, anorm, rcond, work, iwork,
            info)

```

```

CHARACTER*1 uplo
INTEGER*4   info, lda, n
REAL*4      anorm, rcond
INTEGER*4   ipiv(n)
COMPLEX*8   a(lda, n), work(2*n)
CALL CHECON (uplo, n, a, lda, ipiv, anorm, rcond, work, info)

```

```

CHARACTER*1 uplo
INTEGER*4   info, lda, n
REAL*4      anorm, rcond
INTEGER*4   ipiv(n)
COMPLEX*8   a(lda, n), work(2*n)
CALL CSYCON (uplo, n, a, lda, ipiv, anorm, rcond, work, info)

```

```

CHARACTER*1 uplo
INTEGER*4   info, lda, n
REAL*8      anorm, rcond
INTEGER*4   ipiv(n)
COMPLEX*16  a(lda, n), work(2*n)
CALL ZHECON (uplo, n, a, lda, ipiv, anorm, rcond, work, info)

```

```

CHARACTER*1 uplo
INTEGER*4    info, lda, n
REAL*8      anorm, rcond
INTEGER*4    ipiv(n)
COMPLEX*16  a(lda, n), work(2*n)
CALL ZSYCON (uplo, n, a, lda, ipiv, anorm, rcond, work, info)

```

LAPACK8, available on C Series and Exemplar architectures:

```

CHARACTER*1 uplo
INTEGER*8    info, lda, n
REAL*8      anorm, rcond
INTEGER*8    ipiv(n), iwork(n)
REAL*8      a(lda, n), work(2*n)
CALL SSSYCON (uplo, n, a, lda, ipiv, anorm, rcond, work, iwork,
             info)

```

```

CHARACTER*1 uplo
INTEGER*8    info, lda, n
REAL*8      anorm, rcond
INTEGER*8    ipiv(n)
COMPLEX*16  a(lda, n), work(2*n)
CALL CHECON (uplo, n, a, lda, ipiv, anorm, rcond, work, info)

```

```

CHARACTER*1 uplo
INTEGER*8    info, lda, n
REAL*8      anorm, rcond
INTEGER*8    ipiv(n)
COMPLEX*16  a(lda, n), work(2*n)
CALL CSYCON (uplo, n, a, lda, ipiv, anorm, rcond, work, info)

```

Input	uplo	Specifies whether the upper or lower triangular factor of the symmetric or Hermitian matrix A is stored, as follows: uplo = 'U' or 'u': The upper triangular factor of A is stored. uplo = 'L' or 'l': The lower triangular factor of A is stored.
	n	The order of the matrix A . $n \geq 0$.
	a	The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by <code>_SYTRF</code> or <code>_HETRF</code> .
	lda	The leading dimension of array a in the calling program unit. $lda \geq \max(1, n)$.
	ipiv	Details of the interchanges and the block structure of D as determined by <code>_SYTRF</code> or <code>_HETRF</code> .
	anorm	$\ A\ _1$ ($= \ A\ _1$) of the original symmetric or Hermitian matrix A . $anorm \geq 0$.
Working Storage	work, iwork	Arrays used for work space.

Continued

SSYCON/DSYCON/CHECON/CSPCON/ZHECON/ZSPCON

Output **rcond** On successful exit, the estimate of the reciprocal condition number of the matrix A , computed as $\mathbf{rcond} = (\|A\|_1 \|A^{-1}\|_1)^{-1}$. If \mathbf{rcond} is small enough so that the logical expression

$$1.0 + \mathbf{rcond} \text{ .EQ. } 1.0$$

is true, then A can be regarded as singular to working precision. If \mathbf{rcond} is zero, then the companion subprograms for solving and computing the inverse may divide by zero.

info Status response:

info = 0: Successful exit.

info < 0: If **info** = $-k$, the k -th argument had an invalid value.

Notes If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

uplo \neq 'L' or 'l' or 'U' or 'u',
n < 0,
lda < max(1,n), and
anorm < 0.0.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the CALL statement may be improved by coding the **uplo** argument as 'Lower' for 'L' or 'Upper' for 'U'.

Purpose These subprograms compute the factorization of a real or complex symmetric or complex Hermitian matrix A using the Bunch-Kaufman diagonal pivoting method. If A is known to be positive definite, other subprograms in this chapter, specifically designed for positive definite matrices, will be more efficient than these subprograms.

A matrix is symmetric if $A = A^T$, its transpose; a matrix is Hermitian if $A = A^*$, its conjugate transpose.

The structure of A is indicated by the name of the subprogram used:

SSYTRF or DSYTRF A is a real symmetric matrix.
 CSYTRF or ZSYTRF A is a complex symmetric matrix.
 CHETRF or ZHETRF A is a complex Hermitian matrix.

If A is real or complex symmetric, the factorization has the form $A = UDU^T$ or $A = LDL^T$ where U is a product of permutation and unit upper triangular matrices, L is a product of permutation and unit lower triangular matrices, and D is symmetric and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

If A is complex Hermitian, the factorization has the form $A = UDU^*$ or $A = LDL^*$ where U and L are as above, and D is Hermitian and block diagonal with 1-by-1 and 2-by-2 diagonal blocks.

Matrix Storage Because either triangle of A may be obtained from the other, you need only provide one triangle of A . You may supply either the upper or the lower triangle of A , in a two-dimensional array large enough to hold the entire matrix. The other triangle of the array is not referenced.

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```
CHARACTER*1 uplo
INTEGER*4    info, lda, lwork, n
INTEGER*4    ipiv(n)
REAL*4      a(lda, n), work(lwork)
CALL SSYTRF (uplo, n, a, lda, ipiv, work, lwork, info)
```

```
CHARACTER*1 uplo
INTEGER*4    info, lda, lwork, n
INTEGER*4    ipiv(n)
REAL*8      a(lda, n), work(lwork)
CALL DSYTRF (uplo, n, a, lda, ipiv, work, lwork, info)
```

```
CHARACTER*1 uplo
INTEGER*4    info, lda, lwork, n
INTEGER*4    ipiv(n)
COMPLEX*8   a(lda, n), work(lwork)
CALL CHETRF (uplo, n, a, lda, ipiv, work, lwork, info)
```

```
CHARACTER*1 uplo
INTEGER*4    info, lda, lwork, n
INTEGER*4    ipiv(n)
COMPLEX*8   a(lda, n), work(lwork)
CALL CSYTRF (uplo, n, a, lda, ipiv, work, lwork, info)
```

Continued

SSYTRF/DSYTRF/CHETRF/CSYTRF/ZHETRF/ZSYTRF

CHARACTER*1 uplo
INTEGER*4 info, lda, lwork, n
INTEGER*4 ipiv(n)
COMPLEX*16 a(lda, n), work(lwork)
CALL ZHETRF (uplo, n, a, lda, ipiv, work, lwork, info)

CHARACTER*1 uplo
INTEGER*4 info, lda, lwork, n
INTEGER*4 ipiv(n)
COMPLEX*16 a(lda, n), work(lwork)
CALL ZSYTRF (uplo, n, a, lda, ipiv, work, lwork, info)

LAPACK8, available on C Series and Exemplar architectures:

CHARACTER*1 uplo
INTEGER*8 info, lda, lwork, n
INTEGER*8 ipiv(n)
REAL*8 a(lda, n), work(lwork)
CALL SSYTRF (uplo, n, a, lda, ipiv, work, lwork, info)

CHARACTER*1 uplo
INTEGER*8 info, lda, lwork, n
INTEGER*8 ipiv(n)
COMPLEX*16 a(lda, n), work(lwork)
CALL CHETRF (uplo, n, a, lda, ipiv, work, lwork, info)

CHARACTER*1 uplo
INTEGER*8 info, lda, lwork, n
INTEGER*8 ipiv(n)
COMPLEX*16 a(lda, n), work(lwork)
CALL CSYTRF (uplo, n, a, lda, ipiv, work, lwork, info)

Input **uplo** Specifies whether the upper or lower triangular part of the symmetric or Hermitian matrix A is stored, as follows:

uplo = 'U' or 'u': The upper triangular part of A is stored.
uplo = 'L' or 'l': The lower triangular part of A is stored.

n The order of the matrix A . $n \geq 0$.

a The symmetric or Hermitian matrix A , as follows:

If **uplo** = 'U' or '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' or '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 The leading dimension of array **a** in the calling program unit. $lda \geq \max(1, n)$.

lwork The length of array **work**. $lwork \geq \max(1, n)$. For good performance, **lwork** must generally be larger. The optimum value of **lwork** for high performance is returned in **work(1)**.

Working Storage	work	An array used for work space. On successful exit, work(1) contains the optimal work space length lwork for high performance.
Output	a	On successful exit, the block diagonal matrix D and the multipliers used to obtain the factor U or L overwrites the input.
	ipiv	On successful exit, details of the interchanges and the block structure of D : If $\text{ipiv}(k) > 0$, then rows and columns k and $\text{ipiv}(k)$ were interchanged and $D(k,k)$ is a 1-by-1 diagonal block. If $\text{uplo} = \text{'U'}$ or 'u' and $\text{ipiv}(k) = \text{ipiv}(k-1) < 0$, then rows and columns $k-1$ and $-\text{ipiv}(k)$ were interchanged and $D(k-1:k, k-1:k)$ is a 2-by-2 diagonal block. If $\text{uplo} = \text{'L'}$ or 'l' and $\text{ipiv}(k) = \text{ipiv}(k+1) < 0$, then rows and columns $k+1$ and $-\text{ipiv}(k)$ were interchanged and $D(k:k+1, k:k+1)$ is a 2-by-2 diagonal block.
	info	Status response: info = 0: Successful exit. info < 0: If $\text{info} = -k$, the k -th argument had an invalid value. info > 0: If $\text{info} = k$, $D(k,k)$ is zero. The factorization has been completed, but the block diagonal matrix D is singular, and division by zero will occur if it is used to solve a system of equations.

Notes If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

uplo \neq 'L' or 'l' or 'U' or 'u' ,
n < 0 ,
lda $< \max(1, \mathbf{n})$, and
lwork too small.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding the **uplo** argument as 'Lower' for 'L' or 'Upper' for 'U' .

Invert Symmetric or Hermitian Matrix**SSYTRI/.../ZHETRI/ZSYTRI**

Purpose These subprograms compute the inverse of a real or complex symmetric or complex Hermitian matrix A using the Bunch-Kaufman factorization computed by `_SYTRF` or `_HETRF`.

The structure of A is indicated by the name of the subprogram used:

SSYTRI or DSYTRI A is a real symmetric matrix.
 CSYTRI or ZSYTRI A is a complex symmetric matrix.
 CHETRI or ZHETRI A is a complex Hermitian matrix.

Matrix Storage Because either triangle of A^{-1} may be obtained from that triangle of the Bunch-Kaufman factorization of A or from the other triangle of A^{-1} , you need only provide one triangle of the factorization, and only the corresponding triangle of A^{-1} is computed. You may supply either the upper or the lower triangle of the factorization of A , in a two-dimensional array large enough to hold the entire matrix. The other triangle of the array is not referenced.

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```
CHARACTER*1 uplo
INTEGER*4   info, lda, n
INTEGER*4   ipiv(n)
REAL*4      a(lda, n), work(n)
CALL SSYTRI (uplo, n, a, lda, ipiv, work, info)
```

```
CHARACTER*1 uplo
INTEGER*4   info, lda, n
INTEGER*4   ipiv(n)
REAL*8      a(lda, n), work(n)
CALL DSYTRI (uplo, n, a, lda, ipiv, work, info)
```

```
CHARACTER*1 uplo
INTEGER*4   info, lda, n
INTEGER*4   ipiv(n)
COMPLEX*8   a(lda, n), work(n)
CALL CHETRI (uplo, n, a, lda, ipiv, work, info)
```

```
CHARACTER*1 uplo
INTEGER*4   info, lda, n
INTEGER*4   ipiv(n)
COMPLEX*8   a(lda, n), work(n)
CALL CSYTRI (uplo, n, a, lda, ipiv, work, info)
```

```
CHARACTER*1 uplo
INTEGER*4   info, lda, n
INTEGER*4   ipiv(n)
COMPLEX*16  a(lda, n), work(n)
CALL ZHETRI (uplo, n, a, lda, ipiv, work, info)
```

```
CHARACTER*1 uplo
INTEGER*4   info, lda, n
INTEGER*4   ipiv(n)
COMPLEX*16  a(lda, n), work(n)
CALL ZSYTRI (uplo, n, a, lda, ipiv, work, info)
```

LAPACK8, available on C Series and Exemplar architectures:

```

CHARACTER*1  uplo
INTEGER*8    info, lda, n
INTEGER*8    ipiv(n)
REAL*8       a(lda, n), work(n)
CALL SSYTRI (uplo, n, a, lda, ipiv, work, info)

```

```

CHARACTER*1  uplo
INTEGER*8    info, lda, n
INTEGER*8    ipiv(n)
COMPLEX*16  a(lda, n), work(n)
CALL CHETRI (uplo, n, a, lda, ipiv, work, info)

```

```

CHARACTER*1  uplo
INTEGER*8    info, lda, n
INTEGER*8    ipiv(n)
COMPLEX*16  a(lda, n), work(n)
CALL CSYTRI (uplo, n, a, lda, ipiv, work, info)

```

Input	uplo	<p>Specifies whether the upper or lower triangular factor of the symmetric or Hermitian matrix A is stored, as follows:</p> <p>uplo = 'U' or 'u': The upper triangular factor of A is stored. uplo = 'L' or 'l': The lower triangular factor of A is stored.</p>
	n	The order of the matrix A . $n \geq 0$.
	a	The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by <code>_SYTRF</code> or <code>_HETRF</code> .
	lda	The leading dimension of array a in the calling program unit. $lda \geq \max(1, n)$.
	ipiv	Details of the interchanges and the block structure of D as determined by <code>_SYTRF</code> or <code>_HETRF</code> .
Working Storage	work	An array used for work space.
Output	a	<p>On successful exit, the upper or lower triangle of the symmetric or Hermitian inverse of A overwrites the input, as follows:</p> <p>If uplo = 'U' or 'u', the upper triangular part of A^{-1} overwrites the leading n-by-n upper triangular part of a, and the strictly lower triangular part of a is not referenced.</p> <p>If uplo = 'L' or 'l', the lower triangular part of A^{-1} overwrites the leading n-by-n lower triangular part of a, and the strictly upper triangular part of a is not referenced.</p>
	info	<p>Status response:</p> <p>info = 0: Successful exit. info < 0: If info = $-k$, the k-th argument had an invalid value. info > 0: If info = k, $D(k, k)$ is zero; the matrix is singular and its inverse could not be computed.</p>

Notes

It is almost never necessary to compute the inverse of a matrix. While papers and reference books extensively use the notation " $A^{-1}b$ " to mean "the solution x of the system of linear equations $Ax = b$," it is almost always more efficient and more accurate to compute the solution directly than to invert the coefficient matrix and multiply the inverse times the right hand side vector. This is true even if there are many systems of equations, all using the same coefficient matrix; the matrix may be factored once and the systems may be solved from the factors usually just as efficiently—and more accurately—than by matrix multiplication by the inverse.

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

uplo \neq 'L' or 'l' or 'U' or 'u',
n < 0 , and
lda $< \max(1, n)$.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the CALL statement may be improved by coding the **uplo** argument as 'Lower' for 'L' or 'Upper' for 'U'.

Purpose These subprograms solve a system of linear equations $AX = B$ with a real or complex symmetric or complex Hermitian matrix A using the Bunch-Kaufman factorization computed by `_SYTRF` or `_HETRF`.

The structure of A is indicated by the name of the subprogram used:

SSYTRS or DSYTRS A is a real symmetric matrix.
 CSYTRS or ZSYTRS A is a complex Hermitian matrix.
 CHETRS or ZHETRS A is a complex Hermitian matrix.

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```
CHARACTER*1 uplo
INTEGER*4   info, lda, ldb, n, nrhs
INTEGER*4   ipiv(n)
REAL*4     a(lda, n), b(ldb, nrhs)
CALL SSYTRS (uplo, n, nrhs, a, lda, ipiv, b, ldb, info)
```

```
CHARACTER*1 uplo
INTEGER*4   info, lda, ldb, n, nrhs
INTEGER*4   ipiv(n)
REAL*8     a(lda, n), b(ldb, nrhs)
CALL DSYTRS (uplo, n, nrhs, a, lda, ipiv, b, ldb, info)
```

```
CHARACTER*1 uplo
INTEGER*4   info, lda, ldb, n, nrhs
INTEGER*4   ipiv(n)
COMPLEX*8  a(lda, n), b(ldb, nrhs)
CALL CHETRS (uplo, n, nrhs, a, lda, ipiv, b, ldb, info)
```

```
CHARACTER*1 uplo
INTEGER*4   info, lda, ldb, n, nrhs
INTEGER*4   ipiv(n)
COMPLEX*8  a(lda, n), b(ldb, nrhs)
CALL CSYTRS (uplo, n, nrhs, a, lda, ipiv, b, ldb, info)
```

```
CHARACTER*1 uplo
INTEGER*4   info, lda, ldb, n, nrhs
INTEGER*4   ipiv(n)
COMPLEX*16 a(lda, n), b(ldb, nrhs)
CALL ZHETRS (uplo, n, nrhs, a, lda, ipiv, b, ldb, info)
```

```
CHARACTER*1 uplo
INTEGER*4   info, lda, ldb, n, nrhs
INTEGER*4   ipiv(n)
COMPLEX*16 a(lda, n), b(ldb, nrhs)
CALL ZSYTRS (uplo, n, nrhs, a, lda, ipiv, b, ldb, info)
```

LAPACK8, available on C Series and Exemplar architectures:

```
CHARACTER*1 uplo
INTEGER*8   info, lda, ldb, n, nrhs
INTEGER*8   ipiv(n)
REAL*8     a(lda, n), b(ldb, nrhs)
CALL SSYTRS (uplo, n, nrhs, a, lda, ipiv, b, ldb, info)
```

```

CHARACTER*1 uplo
INTEGER*8   info, lda, ldb, n, nrhs
INTEGER*8   ipiv(n)
COMPLEX*16  a(lda, n), b(ldb, nrhs)
CALL CHETRS (uplo, n, nrhs, a, lda, ipiv, b, ldb, info)

```

```

CHARACTER*1 uplo
INTEGER*8   info, lda, ldb, n, nrhs
INTEGER*8   ipiv(n)
COMPLEX*16  a(lda, n), b(ldb, nrhs)
CALL CSYTRS (uplo, n, nrhs, a, lda, ipiv, b, ldb, info)

```

Input	uplo	Specifies whether the upper or lower triangular factor of the symmetric or Hermitian matrix A is stored, as follows: uplo = 'U' or 'u': The upper triangular factor of A is stored. uplo = 'L' or 'l': The lower triangular factor of A is stored.
	n	The order of the matrix A . $n \geq 0$.
	nrhs	The number of right hand sides, that is, the number of columns of the matrix B . $nrhs \geq 0$.
	a	The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by <code>_SYTRF</code> or <code>_HETRF</code> .
	lda	The leading dimension of array a in the calling program unit. $lda \geq \max(1, n)$.
	ipiv	Details of the interchanges and the block structure of D as determined by <code>_SYTRF</code> or <code>_HETRF</code> .
	b	The n -by- $nrhs$ matrix of right hand side vectors for the system of linear equations $AX = B$.
	ldb	The leading dimension of array b in the calling program unit. $ldb \geq \max(1, n)$.
Output	b	On successful exit, the n -by- $nrhs$ matrix of solution vectors X overwrites the input.
	info	Status response: info = 0: Successful exit. info < 0: If info = $-k$, the k -th argument had an invalid value.

Notes

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

uplo ≠ 'L' or 'l' or 'U' or 'u',
n < 0,
nrhs < 0,
lda < max(1,n), and
ldb < max(1,n).

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding the **uplo** argument as 'Lower' for 'L' or 'Upper' for 'U'.

Condition Number of Triangular Band Matrix**STBCON/.../ZTBCON**

Purpose These subprograms estimate the reciprocal of the condition number of a triangular band matrix A , in either the 1-norm or the ∞ -norm.

$\|A\|$ is computed, an estimate is obtained for $\|A^{-1}\|$, and the reciprocal of the condition number is computed as $\mathbf{rcond} = (\|A\| \|A^{-1}\|)^{-1}$.

Matrix Storage Because it is not necessary to store or operate on the zeros outside the band of A , and since A is triangular, you need only provide the band within the triangle of A . Compared to storing the entire matrix, this can save memory in two ways: only the elements within the band are stored, and of them, only the relevant triangle.

Upper triangular matrix. Consider the following upper triangular matrix A of order $n = 7$ and bandwidth $kd = 2$:

11	12	13	0	0	0	0
0	22	23	24	0	0	0
0	0	33	34	35	0	0
0	0	0	44	45	46	0
0	0	0	0	55	56	57
0	0	0	0	0	66	67
0	0	0	0	0	0	77

A is stored in an array \mathbf{ab} with at least $kd+1 = 3$ rows and 7 columns as follows:

*	*	13	24	35	46	57
*	12	23	34	45	56	67
11	22	33	44	55	66	77

The asterisks represent elements in the kd -by- kd triangle at the upper left corner of \mathbf{ab} that are not referenced. Thus, if a_{ij} is an element within the band of the upper triangle of A , it is stored in $\mathbf{ab}(kd+1+i-j, j)$. Therefore, the columns of the upper triangle of A are stored in the columns of \mathbf{ab} , and the diagonals of the upper triangle of A are stored in the rows of \mathbf{ab} .

Lower triangular matrix. Consider the following lower triangular matrix A of order $n = 7$ and bandwidth $kd = 2$:

11	0	0	0	0	0	0
21	22	0	0	0	0	0
31	32	33	0	0	0	0
0	42	43	44	0	0	0
0	0	53	54	55	0	0
0	0	0	64	65	66	0
0	0	0	0	75	76	77

The lower triangle of A is stored in the array \mathbf{ab} as follows:

11	22	33	44	55	66	77
21	32	43	54	65	76	*
31	42	53	64	75	*	*

The asterisks represent elements in the kd -by- kd triangle at the lower right corner of \mathbf{ab} that are not referenced. Thus, if a_{ij} is an element within the band of the lower triangle of A , it is stored in $\mathbf{ab}(1+i-j, j)$. Therefore, the columns of the lower triangle of A are stored in the columns of \mathbf{ab} , and the diagonals of the lower triangle of A are stored in the rows of \mathbf{ab} .

Usage

LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

CHARACTER*1 diag, norm, uplo
INTEGER*4 info, kd, ldab, n
REAL*4 rcond
INTEGER*4 iwork(n)
REAL*4 ab(ldab, n), work(3*n)
CALL STBCON (norm, uplo, diag, n, kd, ab, ldab, rcond, work,
iwork, info)

CHARACTER*1 diag, norm, uplo
INTEGER*4 info, kd, ldab, n
REAL*8 rcond
INTEGER*4 iwork(n)
REAL*8 ab(ldab, n), work(3*n)
CALL DTBCON (norm, uplo, diag, n, kd, ab, ldab, rcond, work,
iwork, info)

CHARACTER*1 diag, norm, uplo
INTEGER*4 info, kd, ldab, n
REAL*4 rcond, rwork(n)
COMPLEX*8 ab(ldab, n), work(2*n)
CALL CTBCON (norm, uplo, diag, n, kd, ab, ldab, rcond, work,
rwork, info)

CHARACTER*1 diag, norm, uplo
INTEGER*4 info, kd, ldab, n
REAL*8 rcond, rwork(n)
COMPLEX*16 ab(ldab, n), work(2*n)
CALL ZTBCON (norm, uplo, diag, n, kd, ab, ldab, rcond, work,
rwork, info)

LAPACK8, available on C Series and Exemplar architectures:

CHARACTER*1 diag, norm, uplo
INTEGER*8 info, kd, ldab, n
REAL*8 rcond
INTEGER*8 iwork(n)
REAL*8 ab(ldab, n), work(3*n)
CALL STBCON (norm, uplo, diag, n, kd, ab, ldab, rcond, work,
iwork, info)

CHARACTER*1 diag, norm, uplo
INTEGER*8 info, kd, ldab, n
REAL*8 rcond, rwork(n)
COMPLEX*16 ab(ldab, n), work(2*n)
CALL CTBCON (norm, uplo, diag, n, kd, ab, ldab, rcond, work,
rwork, info)

Input

norm Specifies whether to use the 1-norm or the ∞ -norm to estimate the condition number, as follows:

norm = '1', 'O', or 'o': Use $\| \cdot \|_1$.

norm = 'T' or 't': Use $\| \cdot \|_\infty$.

	uplo	Specifies whether the matrix A is an upper or lower triangular band matrix, as follows: uplo = 'U' or 'u': A is an upper triangular band matrix. uplo = 'L' or 'l': A is a lower triangular band matrix.
	diag	Specifies whether the matrix A is unit triangular, that is, $a_{ii} = 1$, or not, as follows: diag = 'N' or 'n': The diagonal of A is stored in the array. diag = 'U' or 'u': The diagonal of A consists of unstored ones.
	n	The order of the matrix A . $n \geq 0$.
	kd	The number of super-diagonals of the matrix A if uplo = 'U' or 'u', or the number of sub-diagonals if uplo = 'L' or 'l'. $kd \geq 0$.
	ab	The upper or lower triangular band matrix A , stored in the first $kd+1$ rows of the array. The j -th column of A is stored in the j -th column of array ab as follows: If uplo = 'U' or 'u', $ab(kd+1+i-j, j) = A(i, j)$ for $\max(1, j-kd) \leq i \leq j$; If uplo = 'L' or 'l', $ab(1+i-j, j) = A(i, j)$ for $j \leq i \leq \min(n, j+kd)$. If diag = 'U' or 'u', the diagonal elements of A are not referenced and are assumed to be 1.
	ldab	The leading dimension of array ab in the calling program unit. $ldab \geq kd+1$.
Working Storage	work, iwork, rwork	Arrays used for work space.
Output	rcond	On successful exit, the estimate of the reciprocal condition number of the matrix A , computed as $rcond = (\ A\ \ A^{-1}\)^{-1}$, using the norm specified by norm . If rcond is small enough so that the logical expression $1.0 + rcond .EQ. 1.0$ is true, then A can be regarded as singular to working precision. If rcond is zero, then the companion subprograms for solving and computing the inverse may divide by zero.
	info	Status response: info = 0: Successful exit. info < 0: If info = $-k$, the k -th argument had an invalid value.

Notes

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

norm \neq '1' or 'O' or 'o' or 'I' or 'i',
uplo \neq 'L' or 'l' or 'U' or 'u',
diag \neq 'N' or 'n' or 'U' or 'u',
n < 0 ,
kd < 0 , and
ldab $< kd+1$.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding, for example, the **norm** argument as 'Infinity-Norm' for 'I' or '1-norm' for '1'.

Solve Triangular Band Linear System**STBTRS/DTBTRS/.../ZTBTRS**

Purpose These subprograms solve a system of linear equations $AX = B$, $A^T X = B$, or $A^* X = B$ with an upper or lower triangular band matrix A , where A^T is the transpose of A , and A^* is the conjugate transpose.

Matrix Storage Because it is not necessary to store or operate on the zeros outside the band of A , and since A is triangular, you need only provide the band within the triangle of A . Compared to storing the entire matrix, this can save memory in two ways: only the elements within the band are stored, and of them, only the relevant triangle.

Upper triangular matrix. Consider the following upper triangular matrix A of order $n = 7$ and bandwidth $kd = 2$:

11	12	13	0	0	0	0
0	22	23	24	0	0	0
0	0	33	34	35	0	0
0	0	0	44	45	46	0
0	0	0	0	55	56	57
0	0	0	0	0	66	67
0	0	0	0	0	0	77

A is stored in an array **ab** with at least $kd+1 = 3$ rows and 7 columns as follows:

*	*	13	24	35	46	57
*	12	23	34	45	56	67
11	22	33	44	55	66	77

The asterisks represent elements in the kd -by- kd triangle at the upper left corner of **ab** that are not referenced. Thus, if a_{ij} is an element within the band of the upper triangle of A , it is stored in $\mathbf{ab}(kd+1+i-j, j)$. Therefore, the columns of the upper triangle of A are stored in the columns of **ab**, and the diagonals of the upper triangle of A are stored in the rows of **ab**.

Lower triangular matrix. Consider the following lower triangular matrix A of order $n = 7$ and bandwidth $kd = 2$:

11	0	0	0	0	0	0
21	22	0	0	0	0	0
31	32	33	0	0	0	0
0	42	43	44	0	0	0
0	0	53	54	55	0	0
0	0	0	64	65	66	0
0	0	0	0	75	76	77

The lower triangle of A is stored in the array **ab** as follows:

11	22	33	44	55	66	77
21	32	43	54	65	76	*
31	42	53	64	75	*	*

The asterisks represent elements in the kd -by- kd triangle at the lower right corner of **ab** that are not referenced. Thus, if a_{ij} is an element within the band of the lower triangle of A , it is stored in $\mathbf{ab}(1+i-j, j)$. Therefore, the columns of the lower triangle of A are stored in the columns of **ab**, and the diagonals of the lower triangle of A are stored in the rows of **ab**.

Usage

LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

CHARACTER*1 **diag**, **trans**, **uplo**
INTEGER*4 **info**, **kd**, **ldab**, **ldb**, **n**, **nrhs**
REAL*4 **ab**(**ldab**, **n**), **b**(**ldb**, **nrhs**)
CALL STBTRS (**uplo**, **trans**, **diag**, **n**, **kd**, **nrhs**, **ab**, **ldab**, **b**, **ldb**,
info)

CHARACTER*1 **diag**, **trans**, **uplo**
INTEGER*4 **info**, **kd**, **ldab**, **ldb**, **n**, **nrhs**
REAL*8 **ab**(**ldab**, **n**), **b**(**ldb**, **nrhs**)
CALL DTBTRS (**uplo**, **trans**, **diag**, **n**, **kd**, **nrhs**, **ab**, **ldab**, **b**, **ldb**,
info)

CHARACTER*1 **diag**, **trans**, **uplo**
INTEGER*4 **info**, **kd**, **ldab**, **ldb**, **n**, **nrhs**
COMPLEX*8 **ab**(**ldab**, **n**), **b**(**ldb**, **nrhs**)
CALL CTBTRS (**uplo**, **trans**, **diag**, **n**, **kd**, **nrhs**, **ab**, **ldab**, **b**, **ldb**,
info)

CHARACTER*1 **diag**, **trans**, **uplo**
INTEGER*4 **info**, **kd**, **ldab**, **ldb**, **n**, **nrhs**
COMPLEX*16 **ab**(**ldab**, **n**), **b**(**ldb**, **nrhs**)
CALL ZTBTRS (**uplo**, **trans**, **diag**, **n**, **kd**, **nrhs**, **ab**, **ldab**, **b**, **ldb**,
info)

LAPACK8, available on C Series and Exemplar architectures:

CHARACTER*1 **diag**, **trans**, **uplo**
INTEGER*8 **info**, **kd**, **ldab**, **ldb**, **n**, **nrhs**
REAL*8 **ab**(**ldab**, **n**), **b**(**ldb**, **nrhs**)
CALL STBTRS (**uplo**, **trans**, **diag**, **n**, **kd**, **nrhs**, **ab**, **ldab**, **b**, **ldb**,
info)

CHARACTER*1 **diag**, **trans**, **uplo**
INTEGER*8 **info**, **kd**, **ldab**, **ldb**, **n**, **nrhs**
COMPLEX*16 **ab**(**ldab**, **n**), **b**(**ldb**, **nrhs**)
CALL CTBTRS (**uplo**, **trans**, **diag**, **n**, **kd**, **nrhs**, **ab**, **ldab**, **b**, **ldb**,
info)

Input

uplo Specifies whether the matrix A is an upper or lower triangular band matrix, as follows:

uplo = 'U' or 'u': A is an upper triangular band matrix.

uplo = 'L' or 'l': A is a lower triangular band matrix.

trans Specifies the form of the system of equations, as follows:

trans = 'N' or 'n': Solve $AX = B$.

trans = 'T' or 't': Solve $A^T X = B$.

trans = 'C' or 'c': Solve $A^* X = B$.

diag Specifies whether the matrix A is unit triangular, that is, $a_{ii} = 1$, or not, as follows:

diag = 'N' or 'n': The diagonal of A is stored in the array.

diag = 'U' or 'u': The diagonal of A consists of unstored ones.

n	The order of the matrix A . $n \geq 0$.
kd	The number of super-diagonals of the matrix A if uplo = 'U' or 'u', or the number of sub-diagonals if uplo = 'L' or 'l'. $kd \geq 0$.
nrhs	The number of right hand sides, that is, the number of columns of the matrix B . $nrhs \geq 0$.
ab	The upper or lower triangular band matrix A , stored in the first $kd+1$ rows of the array. The j -th column of A is stored in the j -th column of array ab as follows: If uplo = 'U' or 'u', $ab(kd+1+i-j,j) = A(i,j)$ for $\max(1,j-kd) \leq i \leq j$; If uplo = 'L' or 'l', $ab(1+i-j,j) = A(i,j)$ for $j \leq i \leq \min(n,j+kd)$. If diag = 'U' or 'u', the diagonal elements of A are not referenced and are assumed to be 1.
ldab	The leading dimension of array ab in the calling program unit. $ldab \geq kd+1$.
b	The n -by- $nrhs$ matrix of right hand side vectors for the system of linear equations $AX = B$.
ldb	The leading dimension of array b in the calling program unit. $ldb \geq \max(1,n)$.
Output	
b	On successful exit, the n -by- $nrhs$ matrix of solution vectors X overwrites the input.
info	Status response: info = 0: Successful exit. info < 0: If info = $-k$, the k -th argument had an invalid value. info > 0: If info = k , $A(k,k)$ is zero; the matrix is singular and the solution could not be computed.

Notes

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

uplo \neq 'L' or 'l' or 'U' or 'u',
trans \neq 'N' or 'n' or 'T' or 't' or 'C' or 'c',
diag \neq 'N' or 'n' or 'U' or 'u',
n < 0,
kd < 0,
nrhs < 0,
ldab < $kd+1$,
ldb < $\max(1,n)$.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding, for example, the **uplo** argument as 'Lower' for 'L' or 'Upper' for 'U'.

Purpose These subprograms estimate the reciprocal of the condition number of a triangular matrix A that is stored in an array in packed form. The condition number can be estimated in either the 1-norm or the ∞ -norm.

$\|A\|$ is computed, an estimate is obtained for $\|A^{-1}\|$, and the reciprocal of the condition number is computed as $\mathbf{rcond} = (\|A\| \|A^{-1}\|)^{-1}$.

Matrix Storage You supply the upper or lower triangle of A , stored column-by-column in packed form in a 1-dimensional array. This saves memory compared to storing the entire matrix.

The following examples illustrate the storage of triangular packed matrices.

Upper triangular matrix. If A is

11	12	13	14
0	22	23	24
0	0	33	34
0	0	0	44

then A is packed column-by-column into an array \mathbf{ap} as follows:

k	1	2	3	4	5	6	7	8	9	10
$\mathbf{ap}(k)$	11	12	22	13	23	33	14	24	34	44

Upper triangular matrix element a_{ij} is stored in array element $\mathbf{ap}(i+j \times (j-1)/2)$.

Lower triangular matrix. If A is

11	0	0	0
21	22	0	0
31	32	33	0
41	42	43	44

then A is packed column-by-column into an array \mathbf{ap} as follows:

k	1	2	3	4	5	6	7	8	9	10
$\mathbf{ap}(k)$	11	21	31	41	22	32	42	33	43	44

Lower triangular matrix element a_{ij} is stored in array element $\mathbf{ap}(i+(j-1) \times (2n-j)/2)$.

Usage

LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```

CHARACTER*1  diag, norm, uplo
INTEGER*4    info, n
REAL*4       rcond
INTEGER*4    iwork(n)
REAL*4       ap((n*(n+1))/2), work(3*n)
CALL STPCON (norm, uplo, diag, n, ap, rcond, work, iwork, info)

```

CHARACTER*1 diag, norm, uplo
INTEGER*4 info, n
REAL*8 rcond
INTEGER*4 iwork(n)
REAL*8 ap((n*(n+1))/2), work(3*n)
CALL DTPCON (norm, uplo, diag, n, ap, rcond, work, iwork, info)

CHARACTER*1 diag, norm, uplo
INTEGER*4 info, n
REAL*4 rcond, rwork(n)
COMPLEX*8 ap((n*(n+1))/2), work(2*n)
CALL CTPCON (norm, uplo, diag, n, ap, rcond, work, rwork, info)

CHARACTER*1 diag, norm, uplo
INTEGER*4 info, n
REAL*8 rcond, rwork(n)
COMPLEX*16 ap((n*(n+1))/2), work(2*n)
CALL ZTPCON (norm, uplo, diag, n, ap, rcond, work, rwork, info)

LAPACK8, available on C Series and Exemplar architectures:

CHARACTER*1 diag, norm, uplo
INTEGER*8 info, n
REAL*8 rcond
INTEGER*8 iwork(n)
REAL*8 ap((n*(n+1))/2), work(3*n)
CALL STPCON (norm, uplo, diag, n, ap, rcond, work, iwork, info)

CHARACTER*1 diag, norm, uplo
INTEGER*8 info, n
REAL*8 rcond, rwork(n)
COMPLEX*16 ap((n*(n+1))/2), work(2*n)
CALL CTPCON (norm, uplo, diag, n, ap, rcond, work, rwork, info)

Input	norm	Specifies whether to use the 1-norm or the ∞ -norm to estimate the condition number, as follows: norm = '1', 'O', or 'o': Use $\ \cdot \ _1$. norm = 'I' or 'i': Use $\ \cdot \ _{\infty}$.
	uplo	Specifies whether the matrix A is an upper or lower triangular matrix, as follows: uplo = 'U' or 'u': A is an upper triangular matrix. uplo = 'L' or 'l': A is a lower triangular matrix.
	diag	Specifies whether the matrix A is unit triangular, that is, $a_{ii} = 1$, or not, as follows: diag = 'N' or 'n': The diagonal of A is stored in the array. diag = 'U' or 'u': The diagonal of A consists of unstored ones.
	n	The order of the matrix A . $n \geq 0$.

	ap	The n -by- n triangular matrix A , packed columnwise in a linear array. The j -th column of A is stored in the array ap as follows: If uplo = 'U' or 'u', $\mathbf{ap}(i + (j-1) \times j/2) = A(i, j)$ for $1 \leq i \leq j$; If uplo = 'L' or 'l', $\mathbf{ap}(i + (j-1) \times (2n-j)/2) = A(i, j)$ for $j \leq i \leq n$. If diag = 'U' or 'u', the diagonal elements of A are not referenced and are assumed to be 1.
Working Storage	work, iwork, rwork	Arrays used for work space.
Output	rcond	On successful exit, the estimate of the reciprocal condition number of the matrix A , computed as $\mathbf{rcond} = (\ A\ \ A^{-1}\)^{-1}$, using the norm specified by norm . If rcond is small enough so that the logical expression $1.0 + \mathbf{rcond} \text{ .EQ. } 1.0$ is true, then A can be regarded as singular to working precision. If rcond is zero, then the companion subprograms for solving and computing the inverse may divide by zero.
	info	Status response: info = 0: Successful exit. info < 0: If info = $-k$, the k -th argument had an invalid value.
Notes		If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are $\mathbf{norm} \neq \text{'I' or 'O' or 'o' or 'l' or 'i'}$, $\mathbf{uplo} \neq \text{'L' or 'l' or 'U' or 'u'}$, $\mathbf{diag} \neq \text{'N' or 'n' or 'U' or 'u'}$, and $\mathbf{n} < 0$. Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the CALL statement may be improved by coding, for example, the norm argument as 'Infinity-Norm' for 'I' or '1-norm' for '1'.

Invert Triangular Packed Matrix**STPTRI/DTPTRI/CTPTRI/ZTPTRI**

Purpose These subprograms compute the inverse of an upper or lower triangular matrix A that is stored in an array in packed form.

Matrix Storage You supply the upper or lower triangle of A , stored column-by-column in packed form in a 1-dimensional array. This saves memory compared to storing the entire matrix.

The following examples illustrate the storage of triangular packed matrices.

Upper triangular matrix. If A is

11	12	13	14
0	22	23	24
0	0	33	34
0	0	0	44

then A is packed column-by-column into an array **ap** as follows:

k	1	2	3	4	5	6	7	8	9	10
ap (k)	11	12	22	13	23	33	14	24	34	44

Upper triangular matrix element a_{ij} is stored in array element **ap**($i+j \times (j-1)/2$).

Lower triangular matrix. If A is

11	0	0	0
21	22	0	0
31	32	33	0
41	42	43	44

then A is packed column-by-column into an array **ap** as follows:

k	1	2	3	4	5	6	7	8	9	10
ap (k)	11	21	31	41	22	32	42	33	43	44

Lower triangular matrix element a_{ij} is stored in array element **ap**($i+(j-1) \times (2n-j)/2$).

Usage

LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```
CHARACTER*1 diag, uplo
INTEGER*4   info, n
REAL*4     ap((n*(n+1))/2)
CALL STPTRI (uplo, diag, n, ap, info)
```

```
CHARACTER*1 diag, uplo
INTEGER*4   info, n
REAL*8     ap((n*(n+1))/2)
CALL DTPTRI (uplo, diag, n, ap, info)
```

```
CHARACTER*1 diag, uplo
INTEGER*4   info, n
COMPLEX*8  ap((n*(n+1))/2)
CALL CTPTRI (uplo, diag, n, ap, info)
```

```

CHARACTER*1 diag, uplo
INTEGER*4   info, n
COMPLEX*16 ap((n*(n+1))/2)
CALL ZTPTRI (uplo, diag, n, ap, info)

```

LAPACK8, available on C Series and Exemplar architectures:

```

CHARACTER*1 diag, uplo
INTEGER*8   info, n
REAL*8     ap((n*(n+1))/2)
CALL STPTRI (uplo, diag, n, ap, info)

```

```

CHARACTER*1 diag, uplo
INTEGER*8   info, n
COMPLEX*16 ap((n*(n+1))/2)
CALL CTPTRI (uplo, diag, n, ap, info)

```

Input

uplo Specifies whether the matrix A is an upper or lower triangular matrix, as follows:

uplo = 'U' or 'u': A is an upper triangular matrix.
uplo = 'L' or 'l': A is a lower triangular matrix.

diag Specifies whether the matrix A is unit triangular, that is, $a_{ii} = 1$, or not, as follows:

diag = 'N' or 'n': The diagonal of A is stored in the array.
diag = 'U' or 'u': The diagonal of A consists of unstored ones.

n The order of the matrix A . $n \geq 0$.

ap The n -by- n triangular matrix A , packed columnwise in a linear array. The j -th column of A is stored in the array **ap** as follows:

If **uplo** = 'U' or 'u', $ap(i + (j-1) \times j/2) = A(i, j)$ for $1 \leq i \leq j$;

If **uplo** = 'L' or 'l', $ap(i + (j-1) \times (2n-j)/2) = A(i, j)$ for $j \leq i \leq n$.

If **diag** = 'U' or 'u', the diagonal elements of A are not referenced and are assumed to be 1.

Output

ap On successful exit, A^{-1} overwrites the input.

If **diag** = 'U' or 'u', then A^{-1} also will have an unstored unit diagonal.

info Status response:

info = 0: Successful exit.
info < 0: If **info** = $-k$, the k -th argument had an invalid value.
info > 0: If **info** = k , $A(k, k)$ is zero; the matrix is singular and its inverse could not be computed.

Notes

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

uplo ≠ 'L' or 'l' or 'U' or 'u',
diag ≠ 'N' or 'n' or 'U' or 'u', and
n < 0.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding, for example, the **uplo** argument as 'Lower' for 'L' or 'Upper' for 'U'.

Purpose These subprograms solve a system of linear equations $AX = B$, $A^T X = B$, or $A^* X = B$ with an upper or lower triangular matrix A that is stored in an array in packed form, where A^T is the transpose of A , and A^* is the conjugate transpose.

Matrix Storage You supply the upper or lower triangle of A , stored column-by-column in packed form in a 1-dimensional array. This saves memory compared to storing the entire matrix.

The following examples illustrate the storage of triangular packed matrices.

Upper triangular matrix. If A is

11	12	13	14
0	22	23	24
0	0	33	34
0	0	0	44

then A is packed column-by-column into an array **ap** as follows:

k	1	2	3	4	5	6	7	8	9	10
ap(k)	11	12	22	13	23	33	14	24	34	44

Upper triangular matrix element a_{ij} is stored in array element **ap**($i+j \times (j-1)/2$).

Lower triangular matrix. If A is

11	0	0	0
21	22	0	0
31	32	33	0
41	42	43	44

then A is packed column-by-column into an array **ap** as follows:

k	1	2	3	4	5	6	7	8	9	10
ap(k)	11	21	31	41	22	32	42	33	43	44

Lower triangular matrix element a_{ij} is stored in array element **ap**($i+(j-1) \times (2n-j)/2$).

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```

CHARACTER*1  diag, trans, uplo
INTEGER*4    info, ldb, n, nrhs
REAL*4      ap((n*(n+1))/2), b(ldb, nrhs)
CALL STPTRS (uplo, trans, diag, n, nrhs, ap, b, ldb, info)

```

```

CHARACTER*1  diag, trans, uplo
INTEGER*4    info, ldb, n, nrhs
REAL*8      ap((n*(n+1))/2), b(ldb, nrhs)
CALL DTPTRS (uplo, trans, diag, n, nrhs, ap, b, ldb, info)

```

```

CHARACTER*1  diag, trans, uplo
INTEGER*4    info, ldb, n, nrhs
COMPLEX*8   ap((n*(n+1))/2), b(ldb, nrhs)
CALL CTPTRS (uplo, trans, diag, n, nrhs, ap, b, ldb, info)

```

```

CHARACTER*1 diag, trans, uplo
INTEGER*4    info, ldb, n, nrhs
COMPLEX*16  ap((n*(n+1))/2), b(ldb, nrhs)
CALL ZTPTRS (uplo, trans, diag, n, nrhs, ap, b, ldb, info)

```

LAPACK8, available on C Series and Exemplar architectures:

```

CHARACTER*1 diag, trans, uplo
INTEGER*8    info, ldb, n, nrhs
REAL*8      ap((n*(n+1))/2), b(ldb, nrhs)
CALL STPTRS (uplo, trans, diag, n, nrhs, ap, b, ldb, info)

```

```

CHARACTER*1 diag, trans, uplo
INTEGER*8    info, ldb, n, nrhs
COMPLEX*16  ap((n*(n+1))/2), b(ldb, nrhs)
CALL CTPTRS (uplo, trans, diag, n, nrhs, ap, b, ldb, info)

```

Input

uplo Specifies whether the matrix A is an upper or lower triangular matrix, as follows:

uplo = 'U' or 'u': A is an upper triangular matrix.
uplo = 'L' or 'l': A is a lower triangular matrix.

trans Specifies the form of the system of equations, as follows:

trans = 'N' or 'n': Solve $AX = B$.
trans = 'T' or 't': Solve $A^T X = B$.
trans = 'C' or 'c': Solve $A^* X = B$.

diag Specifies whether the matrix A is unit triangular, that is, $a_{ii} = 1$, or not, as follows:

diag = 'N' or 'n': The diagonal of A is stored in the array.
diag = 'U' or 'u': The diagonal of A consists of unstored ones.

n The order of the matrix A . $n \geq 0$.

nrhs The number of right hand sides, that is, the number of columns of the matrix B . $nrhs \geq 0$.

ap The n -by- n triangular matrix A , packed columnwise in a linear array. The j -th column of A is stored in the array **ap** as follows:

If **uplo** = 'U' or 'u', $ap(i + (j-1) \times j/2) = A(i, j)$ for $1 \leq i \leq j$;
 If **uplo** = 'L' or 'l', $ap(i + (j-1) \times (2n-j)/2) = A(i, j)$ for $j \leq i \leq n$.

If **diag** = 'U' or 'u', the diagonal elements of A are not referenced and are assumed to be 1.

b The n -by-**nrhs** matrix of right hand side vectors for the system of linear equations $AX = B$.

ldb The leading dimension of array **b** in the calling program unit. $ldb \geq \max(1, n)$.

Output **b** On successful exit, the n -by- $nrhs$ matrix of solution vectors X overwrites the input.

info Status response:

info = 0: Successful exit.

info < 0: If **info** = $-k$, the k -th argument had an invalid value.

info > 0: If **info** = k , $A(k,k)$ is zero; the matrix is singular and the solution could not be computed.

Notes If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

uplo \neq 'L' or 'l' or 'U' or 'u',
trans \neq 'N' or 'n' or 'T' or 't' or 'C' or 'c',
diag \neq 'N' or 'n' or 'U' or 'u',
 $n < 0$,
nrhs < 0, and
ldb < $\max(1,n)$.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding, for example, the **uplo** argument as 'Lower' for 'L' or 'Upper' for 'U'.

Condition Number of Triangular Matrix**STRCON/.../ZTRCON**

Purpose These subprograms estimate the reciprocal of the condition number of a triangular matrix A , in either the 1-norm or the ∞ -norm.

$\|A\|$ is computed, an estimate is obtained for $\|A^{-1}\|$, and the reciprocal of the condition number is computed as $\text{rcond} = (\|A\| \|A^{-1}\|)^{-1}$.

Matrix Storage For these subprograms, you supply A in a two-dimensional array large enough to hold a square matrix. The other triangle of the array is not referenced. If A has an unstored unit diagonal (see input argument `diag`), then the diagonal elements of the array also will not be referenced.

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```

CHARACTER*1  diag, norm, uplo
INTEGER*4    info, lda, n
REAL*4       rcond
INTEGER*4    iwork(n)
REAL*4       a(lda, n), work(3*n)
CALL STRCON (norm, uplo, diag, n, a, lda, rcond, work, iwork,
              info)

```

```

CHARACTER*1  diag, norm, uplo
INTEGER*4    info, lda, n
REAL*8       rcond
INTEGER*4    iwork(n)
REAL*8       a(lda, n), work(3*n)
CALL DTRCON (norm, uplo, diag, n, a, lda, rcond, work, iwork,
              info)

```

```

CHARACTER*1  diag, norm, uplo
INTEGER*4    info, lda, n
REAL*4       rcond, rwork(n)
COMPLEX*8    a(lda, n), work(2*n)
CALL CTRCON (norm, uplo, diag, n, a, lda, rcond, work, rwork,
              info)

```

```

CHARACTER*1  diag, norm, uplo
INTEGER*4    info, lda, n
REAL*8       rcond, rwork(n)
COMPLEX*16   a(lda, n), work(2*n)
CALL ZTRCON (norm, uplo, diag, n, a, lda, rcond, work, rwork,
              info)

```

LAPACK8, available on C Series and Exemplar architectures:

```

CHARACTER*1  diag, norm, uplo
INTEGER*8    info, lda, n
REAL*8       rcond
INTEGER*8    iwork(n)
REAL*8       a(lda, n), work(3*n)
CALL STRCON (norm, uplo, diag, n, a, lda, rcond, work, iwork,
              info)

```

CHARACTER*1 **diag, norm, uplo**
INTEGER*8 **info, lda, n**
REAL*8 **rcond, rwork(n)**
COMPLEX*16 **a(lda, n), work(2*n)**
CALL CTRCON (**norm, uplo, diag, n, a, lda, rcond, work, rwork,**
info)

Input **norm** Specifies whether to use the 1-norm or the ∞ -norm to estimate the condition number, as follows:

norm = '1', 'O', or 'o': Use $\| \cdot \|_1$.

norm = 'I' or 'i': Use $\| \cdot \|_\infty$.

uplo Specifies whether the matrix A is an upper or lower triangular matrix, as follows:

uplo = 'U' or 'u': A is an upper triangular matrix.

uplo = 'L' or 'l': A is a lower triangular matrix.

diag Specifies whether the matrix A is unit triangular, that is, $a_{ii} = 1$, or not, as follows:

diag = 'N' or 'n': The diagonal of A is stored in the array.

diag = 'U' or 'u': The diagonal of A consists of unstored ones.

n The order of the matrix A . $n \geq 0$.

a The n -by- n triangular matrix A .

If **uplo** = 'U' or 'u', the leading n -by- n upper triangular part of **a** contains the upper triangular matrix A , and the strictly lower triangular part of **a** is not referenced.

If **uplo** = 'L' or 'l', the leading n -by- n lower triangular part of **a** contains the lower triangular matrix A , and the strictly upper triangular part of **a** is not referenced.

If **diag** = 'U' or 'u', the diagonal elements of A are not referenced and are assumed to be 1.

lda The leading dimension of array **a** in the calling program unit. $lda \geq \max(1, n)$.

Working Storage **work,** Arrays used for work space.
iwork,
rwork

Output **rcond** On successful exit, the estimate of the reciprocal condition number of the matrix A , computed as $rcond = (\|A\| \|A^{-1}\|)^{-1}$, using the norm specified by **norm**. If **rcond** is small enough so that the logical expression

$$1.0 + rcond .EQ. 1.0$$

is true, then A can be regarded as singular to working precision. If **rcond** is zero, then the companion subprograms for solving and computing the inverse may divide by zero.

info Status response:

info = 0: Successful exit.

info < 0: If **info** = $-k$, the k -th argument had an invalid value.

Notes

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

norm \neq 'I' or 'O' or 'o' or 'I' or 'i',

uplo \neq 'L' or 'l' or 'U' or 'u',

diag \neq 'N' or 'n' or 'U' or 'u',

n < 0, and

lda < max(1,n).

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding, for example, the **norm** argument as 'Infinity-Norm' for 'I' or '1-norm' for '1'.

Purpose These subprograms compute the inverse of an upper or lower triangular matrix A .

Matrix Storage For these subprograms, you supply A in a two-dimensional array large enough to hold a square matrix. The other triangle of the array is not referenced. If A has an unstored unit diagonal (see input argument **diag**), then A^{-1} also will have an unstored unit diagonal and the diagonal elements of the array also will not be referenced.

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```
CHARACTER*1 diag, uplo
INTEGER*4    info, lda, n
REAL*4       a(lda, n)
CALL STRTRI (uplo, diag, n, a, lda, info)
```

```
CHARACTER*1 diag, uplo
INTEGER*4    info, lda, n
REAL*8       a(lda, n)
CALL DTRTRI (uplo, diag, n, a, lda, info)
```

```
CHARACTER*1 diag, uplo
INTEGER*4    info, lda, n
COMPLEX*8    a(lda, n)
CALL CTRTRI (uplo, diag, n, a, lda, info)
```

```
CHARACTER*1 diag, uplo
INTEGER*4    info, lda, n
COMPLEX*16   a(lda, n)
CALL ZTRTRI (uplo, diag, n, a, lda, info)
```

LAPACK8, available on C Series and Exemplar architectures:

```
CHARACTER*1 diag, uplo
INTEGER*8    info, lda, n
REAL*8       a(lda, n)
CALL STRTRI (uplo, diag, n, a, lda, info)
```

```
CHARACTER*1 diag, uplo
INTEGER*8    info, lda, n
COMPLEX*16   a(lda, n)
CALL CTRTRI (uplo, diag, n, a, lda, info)
```

Input **uplo** Specifies whether the matrix A is an upper or lower triangular matrix, as follows:

uplo = 'U' or 'u': A is an upper triangular matrix.

uplo = 'L' or 'l': A is a lower triangular matrix.

diag Specifies whether the matrix A is unit triangular, that is, $a_{ii} = 1$, or not, as follows:

diag = 'N' or 'n': The diagonal of A is stored in the array.

diag = 'U' or 'u': The diagonal of A consists of unstored ones.

n The order of the matrix A . $n \geq 0$.

a The n -by- n triangular matrix A .

If **uplo** = 'U' or 'u', the leading n -by- n upper triangular part of **a** contains the upper triangular matrix A , and the strictly lower triangular part of **a** is not referenced.

If **uplo** = 'L' or 'l', the leading **n**-by-**n** lower triangular part of **a** contains the lower triangular matrix *A*, and the strictly upper triangular part of **a** is not referenced.

If **diag** = 'U' or 'u', the diagonal elements of *A* are not referenced and are assumed to be 1.

lda The leading dimension of array **a** in the calling program unit. $lda \geq \max(1,n)$.

Output **a** On successful exit, A^{-1} overwrites the triangle of **a** that contained the input matrix. The other triangle of **a** is not referenced.

If **diag** = 'U' or 'u', then A^{-1} also will have an unstored unit diagonal and the diagonal elements of the array also will not be referenced.

info Status response:

info = 0: Successful exit.

info < 0: If **info** = $-k$, the k -th argument had an invalid value.

info > 0: If **info** = k , $A(k,k)$ is zero; the matrix is singular and its inverse could not be computed.

Notes If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

uplo ≠ 'L' or 'l' or 'U' or 'u',
diag ≠ 'N' or 'n' or 'U' or 'u',
n < 0, and
lda < $\max(1,n)$.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding, for example, the **uplo** argument as 'Lower' for 'L' or 'Upper' for 'U'.

Purpose These subprograms solve a system of linear equations $AX = B$, $A^T X = B$, or $A^* X = B$ with an upper or lower triangular matrix A , where A^T is the transpose of A , and A^* is the conjugate transpose.

Matrix Storage For these subprograms, you supply A in a two-dimensional array large enough to hold a square matrix. The other triangle of the array is not referenced. If A has an unstored unit diagonal (see input argument **diag**), then the diagonal elements of the array also will not be referenced.

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```

CHARACTER*1  diag, trans, uplo
INTEGER*4    info, lda, ldb, n, nrhs
REAL*4      a(lda, n), b(ldb, nrhs)
CALL STRTRS (uplo, trans, diag, n, nrhs, a, lda, b, ldb, info)

```

```

CHARACTER*1  diag, trans, uplo
INTEGER*4    info, lda, ldb, n, nrhs
REAL*8      a(lda, n), b(ldb, nrhs)
CALL DTRTRS (uplo, trans, diag, n, nrhs, a, lda, b, ldb, info)

```

```

CHARACTER*1  diag, trans, uplo
INTEGER*4    info, lda, ldb, n, nrhs
COMPLEX*8   a(lda, n), b(ldb, nrhs)
CALL CTRTRS (uplo, trans, diag, n, nrhs, a, lda, b, ldb, info)

```

```

CHARACTER*1  diag, trans, uplo
INTEGER*4    info, lda, ldb, n, nrhs
COMPLEX*16  a(lda, n), b(ldb, nrhs)
CALL ZTRTRS (uplo, trans, diag, n, nrhs, a, lda, b, ldb, info)

```

LAPACK8, available on C Series and Exemplar architectures:

```

CHARACTER*1  diag, trans, uplo
INTEGER*8    info, lda, ldb, n, nrhs
REAL*8      a(lda, n), b(ldb, nrhs)
CALL STRTRS (uplo, trans, diag, n, nrhs, a, lda, b, ldb, info)

```

```

CHARACTER*1  diag, trans, uplo
INTEGER*8    info, lda, ldb, n, nrhs
COMPLEX*16  a(lda, n), b(ldb, nrhs)
CALL CTRTRS (uplo, trans, diag, n, nrhs, a, lda, b, ldb, info)

```

Input **uplo** Specifies whether the matrix A is an upper or lower triangular matrix, as follows:

```

uplo = 'U' or 'u':   $A$  is an upper triangular matrix.
uplo = 'L' or 'l':   $A$  is a lower triangular matrix.

```

trans Specifies the form of the system of equations, as follows:

```

trans = 'N' or 'n':  Solve  $AX = B$ .
trans = 'T' or 't':  Solve  $A^T X = B$ .
trans = 'C' or 'c':  Solve  $A^* X = B$ .

```

- diag** Specifies whether the matrix A is unit triangular, that is, $a_{ii} = 1$, or not, as follows:
- diag** = 'N' or 'n': The diagonal of A is stored in the array.
diag = 'U' or 'u': The diagonal of A consists of unstored ones.
- n** The order of the matrix A . $n \geq 0$.
- nrhs** The number of right hand sides, that is, the number of columns of the matrix B . $nrhs \geq 0$.
- a** The n -by- n triangular matrix A .
- If **uplo** = 'U' or 'u', the leading n -by- n upper triangular part of **a** contains the upper triangular matrix A , and the strictly lower triangular part of **a** is not referenced.
- If **uplo** = 'L' or 'l', the leading n -by- n lower triangular part of **a** contains the lower triangular matrix A , and the strictly upper triangular part of **a** is not referenced.
- If **diag** = 'U' or 'u', the diagonal elements of A are not referenced and are assumed to be 1.
- lda** The leading dimension of array **a** in the calling program unit. $lda \geq \max(1, n)$.
- b** The n -by- $nrhs$ matrix of right hand side vectors for the system of linear equations $AX = B$.
- ldb** The leading dimension of array **b** in the calling program unit. $ldb \geq \max(1, n)$.
- Output**
- b** On successful exit, the n -by- $nrhs$ matrix of solution vectors X overwrites the input.
- info** Status response:
- info** = 0: Successful exit.
info < 0: If **info** = $-k$, the k -th argument had an invalid value.
info > 0: If **info** = k , $A(k, k)$ is zero; the matrix is singular and the solution could not be computed.

Notes

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

uplo \neq 'L' or 'l' or 'U' or 'u',
trans \neq 'N' or 'n' or 'T' or 't' or 'C' or 'c',
diag \neq 'N' or 'n' or 'U' or 'u',
 $n < 0$,
 $nrhs < 0$,
 $lda < \max(1, n)$, and
 $ldb < \max(1, n)$.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding, for example, the **uplo** argument as 'Lower' for 'L' or 'Upper' for 'U'.

Subprograms not in this Guide

Although LAPACK includes all computational subprograms for linear equations, as defined in Table 1-5, the following subprograms are not documented in the *ConvexMLIB User's Guide: LAPACK*. Refer to (Anderson *et al.*) for documentation for these subprograms.

Table 4-1: Subprograms not in this Guide

Name	Function
SGBEQU	Equilibrate a General Band Matrix
DGBEQU	Equilibrate a General Band Matrix
CGBEQU	Equilibrate a General Band Matrix
ZGBEQU	Equilibrate a General Band Matrix
SGBRFS	Iteratively Refine the Solution of a General Band Linear System
DGBRFS	Iteratively Refine the Solution of a General Band Linear System
CGBRFS	Iteratively Refine the Solution of a General Band Linear System
ZGBRFS	Iteratively Refine the Solution of a General Band Linear System
SGEQU	Equilibrate a General Full Matrix
DGEQU	Equilibrate a General Full Matrix
CGEQU	Equilibrate a General Full Matrix
ZGEQU	Equilibrate a General Full Matrix
SGERFS	Iteratively Refine the Solution of a General Full Linear System
DGERFS	Iteratively Refine the Solution of a General Full Linear System
CGERFS	Iteratively Refine the Solution of a General Full Linear System
ZGERFS	Iteratively Refine the Solution of a General Full Linear System
SGTRFS	Iteratively Refine the Solution of a General Tridiagonal Linear System
DGTRFS	Iteratively Refine the Solution of a General Tridiagonal Linear System
CGTRFS	Iteratively Refine the Solution of a General Tridiagonal Linear System
ZGTRFS	Iteratively Refine the Solution of a General Tridiagonal Linear System
CHERFS	Iteratively Refine the Solution of a Hermitian Indefinite Linear System
ZHERFS	Iteratively Refine the Solution of a Hermitian Indefinite Linear System
CHPRFS	Iteratively Refine the Solution of a Hermitian Indefinite Packed Linear System
ZHPRFS	Iteratively Refine the Solution of a Hermitian Indefinite Packed Linear System
SPBEQU	Equilibrate a Positive Definite Band Matrix
DPBEQU	Equilibrate a Positive Definite Band Matrix
CPBEQU	Equilibrate a Positive Definite Band Matrix
ZPBEQU	Equilibrate a Positive Definite Band Matrix
SPBRFS	Iteratively Refine the Solution of a Positive Definite Band Linear System
DPBRFS	Iteratively Refine the Solution of a Positive Definite Band Linear System
CPBRFS	Iteratively Refine the Solution of a Positive Definite Band Linear System
ZPBRFS	Iteratively Refine the Solution of a Positive Definite Band Linear System
SPOEQU	Equilibrate a Positive Definite Full Matrix
DPOEQU	Equilibrate a Positive Definite Full Matrix
CPOEQU	Equilibrate a Positive Definite Full Matrix
ZPOEQU	Equilibrate a Positive Definite Full Matrix

Name	Function
SPORFS	Iteratively Refine the Solution of a Positive Definite Full Linear System
DPORFS	Iteratively Refine the Solution of a Positive Definite Full Linear System
CPORFS	Iteratively Refine the Solution of a Positive Definite Full Linear System
ZPORFS	Iteratively Refine the Solution of a Positive Definite Full Linear System
SPPEQU	Equilibrate a Positive Definite Packed Matrix
DPPEQU	Equilibrate a Positive Definite Packed Matrix
CPPEQU	Equilibrate a Positive Definite Packed Matrix
ZPPEQU	Equilibrate a Positive Definite Packed Matrix
SPPRFS	Iteratively Refine the Solution of a Positive Definite Packed Linear System
DPPRFS	Iteratively Refine the Solution of a Positive Definite Packed Linear System
CPPRFS	Iteratively Refine the Solution of a Positive Definite Packed Linear System
ZPPRFS	Iteratively Refine the Solution of a Positive Definite Packed Linear System
SPTRFS	Iteratively Refine the Solution of a Positive Definite Tridiagonal Linear System
DPTRFS	Iteratively Refine the Solution of a Positive Definite Tridiagonal Linear System
CPTRFS	Iteratively Refine the Solution of a Positive Definite Tridiagonal Linear System
ZPTRFS	Iteratively Refine the Solution of a Positive Definite Tridiagonal Linear System
SSPRFS	Iteratively Refine the Solution of a Symmetric Indefinite Packed Linear System
DSPRFS	Iteratively Refine the Solution of a Symmetric Indefinite Packed Linear System
CSPRFS	Iteratively Refine the Solution of a Symmetric Packed Linear System
ZSPRFS	Iteratively Refine the Solution of a Symmetric Packed Linear System
SSYRFS	Iteratively Refine the Solution of a Symmetric Indefinite Linear System
DSYRFS	Iteratively Refine the Solution of a Symmetric Indefinite Linear System
CSYRFS	Iteratively Refine the Solution of a Symmetric Linear System
ZSYRFS	Iteratively Refine the Solution of a Symmetric Linear System
STBRFS	Iteratively Refine the Solution of a Triangular Band Linear System
DTBRFS	Iteratively Refine the Solution of a Triangular Band Linear System
CTBRFS	Iteratively Refine the Solution of a Triangular Band Linear System
ZTBRFS	Iteratively Refine the Solution of a Triangular Band Linear System
STPRFS	Iteratively Refine the Solution of a Triangular Packed Linear System
DTPRFS	Iteratively Refine the Solution of a Triangular Packed Linear System
CTPRFS	Iteratively Refine the Solution of a Triangular Packed Linear System
ZTPRFS	Iteratively Refine the Solution of a Triangular Packed Linear System
STRRFS	Iteratively Refine the Solution of a Triangular Linear System
DTRRFS	Iteratively Refine the Solution of a Triangular Linear System
CTRRFS	Iteratively Refine the Solution of a Triangular Linear System
ZTRRFS	Iteratively Refine the Solution of a Triangular Linear System

Drivers for Linear Least Squares Problems

Overview

This chapter explains how to use LAPACK drivers to solve linear least squares problems for under- and over-determined systems of linear equations. The operations covered are:

- solve least squares problems using the QR factorization
- solve least squares problems using the singular value decomposition
- solve least squares problems using the complete orthogonal factorization
- solve generalized linear regression model (GLM) problems
- solve general least squares problems with linear equality constraints

Chapter Objectives

After reading this chapter you will

- know about the various types of least squares problems
- understand the weaknesses of the normal equations solution method
- know how to use the described subprograms

What You Need to Know to Use These Subprograms

The Linear Least Squares Problem

If A is a given m -by- n matrix and b is a given m -dimensional vector, the problem of finding an n -dimensional vector x satisfying $Ax = b$ is said to be *overdetermined* if $m > n$, that is, if there are more equations than unknowns. Usually an overdetermined system has no exact solution, so it is common to try to find an approximate solution that minimizes $\|Ax - b\|$ for some suitable norm. If you choose the Euclidean norm, the minimization problem itself is linear and the problem is called *the least squares problem* (because the solution has the least sum of squares of the residual vector $r = Ax - b$). If A has full column rank, that is, if the columns of A are linearly independent, then the least squares solution is unique.

On the other hand, if $m < n$ then the system $Ax = b$ is said to be *underdetermined*. If the problem is underdetermined or if A does not have full column rank, then there are an infinite number of solutions to the least squares problem, for if x minimizes $\|Ax - b\|^2$ and y is in the null space of A , then $x+y$ is also a minimizer. As the set of all minimizers is convex, there exists a unique minimizer with minimum Euclidean norm, called the *minimum-norm solution*.

The LAPACK drivers described in this chapter provide several options for handling over- or underdetermined linear least squares problems.

The Method of Normal Equations

Probably due to its simple exposition, the relative ease of solving small problems by hand, and the low sophistication of software required to implement it on a computer, the method of normal equations is widely used for solving linear least squares problems $Ax \approx b$ when the matrix A is of size m -by- n with $m > n$ and A has full column rank. Solving the normal equations is theoretically important and is effective in certain cases, but it is not implemented in any of the subprograms described in this chapter.

The method of normal equations reduces the problem of minimizing $\|Ax - b\|_2$ to the seemingly simpler problem of solving $A^T Ax = A^T b$. The matrix $A^T A$ is of size only n -by- n , which is attractive if $m \gg n$. If A has full column rank, then $A^T A$, if computed exactly, is positive definite, so the normal equations can be solved by Cholesky factorization.

The following points are gleaned from (Golub and Van Loan), where the condition number, $\kappa_2(A)$, is the ratio of largest and smallest singular values of A , and $\rho = \min \|Ax - b\|_2$.

- The sensitivity of the least squares solution to changes in A and b is roughly proportional to the quantity $\kappa_2(A) + \rho\kappa_2(A)^2$.
- The method of normal equations produces a computed solution that has a relative error given approximately by $\epsilon\kappa_2(A)^2$.
- The orthogonal factorization methods in LAPACK produce a solution that has a relative error given approximately by $\epsilon(\kappa_2(A) + \rho\kappa_2(A)^2)$.

Thus, if ρ is small and $\kappa_2(A)$ is large, the method of normal equations usually gives a least squares solution that is less accurate than given by the subprograms described in this chapter. In addition, when applied to ill-conditioned problems, the LAPACK subprograms usually will not break down as easily as the normal equations.

In the full-rank case ($\text{rank}(A) = \min(m, n)$), the orthogonal factorization method of `_GELS` is often appropriate. But in the rank deficient case ($\text{rank}(A) < \min(m, n)$), a different method should be used, either the SVD of `_GELSS` or the complete orthogonal factorization of `_GELSX`.

Supplemental Reading

Golub, G.H. and C.F. Van Loan. *Matrix Computations*, 2nd Edition. Baltimore, MD: The Johns Hopkins University Press. 1989.

Lawson, C.L. and R.J. Hanson. *Solving Least Squares Problems*. Englewood Cliffs, NJ: Prentice-Hall, Inc. 1974.

Subprogram Descriptions

Solve a General Least Squares Problem Using Orthogonal Factorization SGELS, DGELS, CGELS, ZGELS	5-4
Solve a General Least Squares Problem Using Singular Value Decomposition SGELSS, DGELSS, CGELSS, ZGELSS	5-7
Solve a General Least Squares Problem Using Complete Orthogonal Factorization SGELSX, DGELSX, CGELSX, ZGELSX	5-10
Solve a Generalized Linear Regression Model (GLM) Problem SGGGLM, DGGGLM, CGGGLM, ZGGGLM	5-13
Solve a General Least Squares Problem with Linear Equality Constraints SGGLSE, DGGLSE, CGGLSE, ZGGLSE	5-15

Purpose These subprograms solve square or over- and under-determined linear systems involving the m -by- n matrix A and the right hand side B using orthogonal factorization of A . A must have full rank, that is, $\text{rank}(A) = \min(m, n)$.

There are several cases, depending on the values of m and n and a transposition option:

1. $m \geq n$, **trans** = 'N' or 'n': Solve the least squares problem of finding X to minimize the Euclidean norm of each column of $AX-B$, where A is an m -by- n matrix, B is an m -by- $nrhs$ matrix, and X is an n -by- $nrhs$ matrix. After computing the QR factorization

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

where Q_1 is m -by- n , Q_2 is m -by- $(m-n)$, R is n -by- n , and 0 is an $(m-n)$ -by- n matrix of zeros, the least squares solution is $X = R^{-1}Q_1^*B$.

2. $m \geq n$, **trans** = 'T' or 't' or 'C' or 'c': Solve the underdetermined system $A^T X = B$ or $A^* X = B$, where A^T is the transpose of the real matrix A and A^* is the conjugate transpose of the complex matrix A . After computing the QR factorization

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

where Q_1 is m -by- n , Q_2 is m -by- $(m-n)$, R is n -by- n , and 0 is an $(m-n)$ -by- n matrix of zeros, the minimum-norm solution is $X = Q_1 R^{-*} B$, where R^{-*} is the inverse of the conjugate transpose of R .

3. $m < n$, **trans** = 'N' or 'n': Solve the underdetermined system $AX = B$. After computing the LQ factorization

$$A = \begin{bmatrix} L & 0 \end{bmatrix} \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix}$$

where L is m -by- m , 0 is an m -by- $(n-m)$ matrix of zeros, Q_1 is m -by- n , and Q_2 is $(n-m)$ -by- n , the minimum-norm solution is $X = Q_1^* L^{-1} B$.

4. $m < n$, **trans** = 'T' or 't' or 'C' or 'c': Solve the least squares problem of finding X to minimize the Euclidean norm of each column of $A^T X - B$ or $A^* X - B$, where A is an m -by- n matrix, A^T is the transpose of the real matrix A , A^* is the conjugate transpose of the complex matrix A , B is an m -by- $nrhs$ matrix, and X is an n -by- $nrhs$ matrix. After computing the LQ factorization

$$A = \begin{bmatrix} L & 0 \end{bmatrix} \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix}$$

where L is m -by- m , 0 is an m -by- $(n-m)$ matrix of zeros, Q_1 is m -by- n , and Q_2 is $(n-m)$ -by- n , the least squares solution is $X = L^{-*} Q_1 B$, where L^{-*} is the inverse of the conjugate transpose of L .

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```
CHARACTER*1 trans
INTEGER*4      info, lda, ldb, lwork, m, n, nrhs
REAL*8         a(lda, n), b(ldb, nrhs), work(lwork)
CALL SGELS (trans, m, n, nrhs, a, lda, b, ldb, work, lwork, info)
```

```
CHARACTER*1 trans
INTEGER*4      info, lda, ldb, lwork, m, n, nrhs
REAL*8         a(lda, n), b(ldb, nrhs), work(lwork)
CALL DGELS (trans, m, n, nrhs, a, lda, b, ldb, work, lwork, info)
```

```
CHARACTER*1 trans
INTEGER*4      info, lda, ldb, lwork, m, n, nrhs
COMPLEX*8     a(lda, n), b(ldb, nrhs), work(lwork)
CALL CGELS (trans, m, n, nrhs, a, lda, b, ldb, work, lwork, info)
```

```
CHARACTER*1 trans
INTEGER*4      info, lda, ldb, lwork, m, n, nrhs
COMPLEX*16    a(lda, n), b(ldb, nrhs), work(lwork)
CALL ZGELS (trans, m, n, nrhs, a, lda, b, ldb, work, lwork, info)
```

LAPACK8, available on C Series and Exemplar architectures:

```
CHARACTER*1 trans
INTEGER*8      info, lda, ldb, lwork, m, n, nrhs
REAL*8         a(lda, n), b(ldb, nrhs), work(lwork)
CALL SGELS (trans, m, n, nrhs, a, lda, b, ldb, work, lwork, info)
```

```
CHARACTER*1 trans
INTEGER*8      info, lda, ldb, lwork, m, n, nrhs
COMPLEX*16    a(lda, n), b(ldb, nrhs), work(lwork)
CALL CGELS (trans, m, n, nrhs, a, lda, b, ldb, work, lwork, info)
```

Input **trans** Transposition option for the matrix A :

trans = 'N' or 'n': Solve $AX = B$.

trans = 'T' or 't': Solve $A^T X = B$.

trans = 'C' or 'c': Solve $A^* X = B$.

trans = 'T' or 't' is valid only in subprograms SGELS and DGELS, and **trans** = 'C' or 'c' is valid only in subprograms CGELS and ZGELS.

m The number of rows of the matrix A . $m \geq 0$.

n The number of columns of the matrix A . $n \geq 0$.

nrhs The number of right hand sides; that is, the number of columns of the matrix B . $nrhs \geq 0$.

a The m -by- n matrix A .

lda The leading dimension of array **a** in the calling program unit. $lda \geq \max(1, m)$.

	b	If trans = 'N' or 'n', the m -by- nrhs right hand side matrix B . If trans = 'T' or 't' or 'C' or 'c', the n -by- nrhs right hand side matrix B .
	ldb	The leading dimension of array b in the calling program unit. ldb \geq $\max(1, m, n)$.
	lwork	The length of array work . lwork \geq $\max(1, \min(m, n) + \max(m, n, nrhs))$. For good performance, lwork must generally be larger. The optimum value of lwork for high performance is returned in work (1).
Working Storage	work	An array used for work space. On successful exit, work (1) contains the optimal work space length lwork for high performance.
Output	a	If m \geq n , a has been overwritten by the <i>QR</i> factorization of A . If m $<$ n , a has been overwritten by the <i>LQ</i> factorization of A .
	b	On successful exit, if m \geq n and trans = 'N' or '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. On successful exit, if m \geq n and trans = 'T' or 't', the m -by- nrhs minimum-norm solution. On successful exit, if m $<$ n and trans = 'N' or 'n', the n -by- nrhs minimum-norm solution. On successful exit, if m $<$ n and trans = 'T' or 't', 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.
	info	Status response: info = 0: Successful exit. info $<$ 0: If info = $-k$, the k -th argument had an invalid value.

Notes If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

trans \neq 'N' or 'n' or 'T' or 't' or 'C' or 'c',
m $<$ 0,
n $<$ 0,
nrhs $<$ 0,
lda $<$ $\max(1, m)$,
ldb too small, and
lwork too small.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding the **trans** argument as 'NotTransposed' for 'N', 'Transposed' for 'T', or 'ConjugateTransposed' for 'C'.

Using Singular Value Decomposition

SGELSS/DGELSS/CGELSS/ZGELSS

Purpose These subprograms use the singular value decomposition of A to solve the least squares problem of finding X to minimize the Euclidean norm of each column of $AX-B$, where A is an m -by- n matrix, B is an m -by- $nrhs$ matrix, and X is an n -by- $nrhs$ matrix. If $m \geq n$, the problem is overdetermined and the least squares solution is computed. If $m < n$, the problem is underdetermined; in this case a minimum-norm solution is returned.

The singular values of A which are smaller than a specified tolerance times the largest singular value are treated as zero in solving the least squares problem; in this case a minimum-norm solution is returned.

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```

INTEGER*4 info, lda, ldb, lwork, m, n, nrhs, rank
REAL*4 rcond
REAL*4 a(lda, n), b(ldb, nrhs), s(min(m,n)), work(lwork)
CALL SGELSS (m, n, nrhs, a, lda, b, ldb, s, rcond, rank, work,
              lwork, info)

```

```

INTEGER*4 info, lda, ldb, lwork, m, n, nrhs, rank
REAL*8 rcond
REAL*8 a(lda, n), b(ldb, nrhs), s(min(m,n)), work(lwork)
CALL DGELSS (m, n, nrhs, a, lda, b, ldb, s, rcond, rank, work,
              lwork, info)

```

```

INTEGER*4 info, lda, ldb, lwork, m, n, nrhs, rank
REAL*4 rcond
REAL*4 rwork(max(1,5*min(m,n)-1)), s(min(m,n))
COMPLEX*8 a(lda, n), b(ldb, nrhs), work(lwork)
CALL CGELSS (m, n, nrhs, a, lda, b, ldb, s, rcond, rank, work,
              lwork, rwork, info)

```

```

INTEGER*4 info, lda, ldb, lwork, m, n, nrhs, rank
REAL*8 rcond
REAL*8 rwork(max(1,5*min(m,n)-1)), s(min(m,n))
COMPLEX*16 a(lda, n), b(ldb, nrhs), work(lwork)
CALL ZGELSS (m, n, nrhs, a, lda, b, ldb, s, rcond, rank, work,
              lwork, rwork, info)

```

LAPACK8, available on C Series and Exemplar architectures:

```

INTEGER*8 info, lda, ldb, lwork, m, n, nrhs, rank
REAL*8 rcond
REAL*8 a(lda, n), b(ldb, nrhs), s(min(m,n)), work(lwork)
CALL SGELSS (m, n, nrhs, a, lda, b, ldb, s, rcond, rank, work,
              lwork, info)

```

```

INTEGER*8 info, lda, ldb, lwork, m, n, nrhs, rank
REAL*8 rcond
REAL*8 rwork(max(1,5*min(m,n)-1)), s(min(m,n))
COMPLEX*16 a(lda, n), b(ldb, nrhs), work(lwork)
CALL CGELSS (m, n, nrhs, a, lda, b, ldb, s, rcond, rank, work,
              lwork, rwork, info)

```

Input	m	The number of rows of the matrix A . $m \geq 0$.
	n	The number of columns of the matrix A . $n \geq 0$.
	nrhs	The number of right hand sides; that is, the number of columns of the matrix B . $nrhs \geq 0$.
	a	The matrix A specifying the least squares problem.
	lda	The leading dimension of array a in the calling program unit. $lda \geq \max(1,m)$.
	b	The m -by- $nrhs$ matrix of right hand side vectors for the least squares problem.
	ldb	The leading dimension of array b in the calling program unit. $ldb \geq \max(1,m,n)$.
	rcond	A tolerance: the singular values of A that are less than or equal to rcond times the largest singular value are treated as zero in solving the least squares problem. If rcond is negative, machine epsilon is used instead. For example, if $Sx = b$ were the least squares problem, where S is a diagonal matrix of singular values and x and b are vectors, the i -th component of the solution would be $x_i = \begin{cases} b_i/s_{ii} & \text{if } s_{ii} > \mathbf{rcond} \times \max_j(s_{jj}) \\ 0 & \text{if } s_{ii} \leq \mathbf{rcond} \times \max_j(s_{jj}) \end{cases}$
	lwork	The length of array work . For SGELSS and DGELSS, $lwork \geq \max(1, 3\min(m,n) + \max(2\min(m,n), m,n,nrhs))$. For CGELSS and ZGELSS, $lwork \geq \max(1, 2\min(m,n) + \max(m,n, nrhs))$. For good performance, lwork must generally be larger. The optimum value of lwork for high performance is returned in work(1) .
	Working Storage	work, rwork
Output	a	On successful exit, the input has been overwritten with the right singular vectors of A .
	b	On successful exit, the solution X in rows 1 through n .
	s	On successful exit, the singular values of A , sorted into decreasing order.
	rank	On successful exit, the number of singular values of A that are greater than rcond times the largest singular value.
	info	Status response: info = 0: Successful exit. info < 0: If info = $-k$, the k -th argument had an invalid value. info > 0: If info = k , the algorithm terminated with k unconverged elements of an intermediate bidiagonal matrix.

Notes

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

$m < 0$,
 $n < 0$,
 $nrhs < 0$,
 $lda < \max(1,m)$,
 $ldb < \max(1,m,n)$, and
 $lwork$ too small.

Purpose Solve the least squares problem of finding X to minimize the Euclidean norm of each column of $AX-B$, where A is an m -by- n matrix, B is an m -by- $nrhs$ matrix, and X is an n -by- $nrhs$ matrix using a complete orthogonal factorization. The subprograms compute the QR factorization of A with column pivoting

$$AP = Q \begin{bmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{bmatrix}$$

where P is a permutation matrix designed to make R_{11} the largest leading square submatrix whose condition number is less than $1/rcond$. Then, treating R_{22} as negligible, R_{12} is annihilated by orthogonal or unitary transformations from the right, giving

$$AP = Q \begin{bmatrix} T_{11} & 0 \\ 0 & 0 \end{bmatrix} Y$$

from which the minimum-norm solution is

$$X = PY^* \begin{bmatrix} T_{11}^{-1} Q_1^* B \\ 0 \end{bmatrix}$$

where Q_1 is the submatrix of Q having the same number of columns as T_{11} .

Usage

LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```

INTEGER*4 info, lda, ldb, m, n, nrhs, rank
REAL*4 rcond
INTEGER*4 jpvt(n)
REAL*4 a(lda, n), b(ldb, nrhs), work(lwork)
CALL SGELSX (m, n, nrhs, a, lda, b, ldb, jpvt, rcond, rank,
              work, info)

```

```

INTEGER*4 info, lda, ldb, m, n, nrhs, rank
REAL*8 rcond
INTEGER*4 jpvt(n)
REAL*8 a(lda, n), b(ldb, nrhs), work(lwork)
CALL DGELSX (m, n, nrhs, a, lda, b, ldb, jpvt, rcond, rank,
              work, info)

```

```

INTEGER*4 info, lda, ldb, m, n, nrhs, rank
REAL*4 rcond
INTEGER*4 jpvt(n)
REAL*4 rwork(2*n)
COMPLEX*8 a(lda, n), b(ldb, nrhs), work(lwork)
CALL CGELSX (m, n, nrhs, a, lda, b, ldb, jpvt, rcond, rank,
              work, rwork, info)

```

```

INTEGER*4 info, lda, ldb, m, n, nrhs, rank
REAL*8 rcond
INTEGER*4 jpvt(n)
REAL*8 rwork(2*n)
COMPLEX*16 a(lda, n), b(ldb, nrhs), work(lwork)
CALL ZGELSX (m, n, nrhs, a, lda, b, ldb, jpvt, rcond, rank,
              work, rwork, info)

```

LAPACK8, available on C Series and Exemplar architectures:

```

INTEGER*8 info, lda, ldb, m, n, nrhs, rank
REAL*8    rcond
INTEGER*8 jpvt(n)
REAL*8    a(lda, n), b(ldb, nrhs), work(lwork)
CALL SGELSX (m, n, nrhs, a, lda, b, ldb, jpvt, rcond, rank,
            work, info)

```

```

INTEGER*8 info, lda, ldb, m, n, nrhs, rank
REAL*8    rcond
INTEGER*8 jpvt(n)
REAL*8    rwork(2*n)
COMPLEX*16 a(lda, n), b(ldb, nrhs), work(lwork)
CALL CGELSX (m, n, nrhs, a, lda, b, ldb, jpvt, rcond, rank,
            work, rwork, info)

```

Input	m	The number of rows of the matrix A . $m \geq 0$.
	n	The number of columns of the matrix A . $n \geq 0$.
	nrhs	The number of right hand sides; that is, the number of columns of the matrix B . $nrhs \geq 0$.
	a	The m -by- n matrix A .
	lda	The leading dimension of array a in the calling program unit. $lda \geq \max(1, m)$.
	b	The m -by- $nrhs$ matrix of right hand side vectors for the least squares problem.
	ldb	The leading dimension of array b in the calling program unit. $ldb \geq \max(1, m, n)$.
	jpvt	If $jpvt(j) \neq 0$, column j is permuted to the front of AP ; otherwise, column j is a free column.
	rcond	A tolerance: the goal of the complete orthogonal factorization is to identify a well-conditioned triangular matrix whose condition number is less than $1/rcond$.
	Working Storage	work
rwork		An array used for work space.
Output	a	On successful exit, the input has been overwritten by the complete orthogonal factorization of A .
	b	On successful exit, the first n rows of b contain the minimum-norm solution.
	jpvt	On successful exit, if $jpvt(j) = k$, then the j -th column of AP was the k -th column of A .
	rank	On successful exit, the size of the largest leading triangular matrix in the QR factorization (with pivoting) of A , whose condition number is less than $1/rcond$.

info Status response:

info = 0: Successful exit.

info < 0: If **info** = $-k$, the k -th argument had an invalid value.

Notes

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

$m < 0$,
 $n < 0$,
 $nrhs < 0$,
 $lda < \max(1,m)$, and
 $ldb < \max(1,m,n)$.

Generalized Linear Regression**SGGGLM/DGGGLM/CGGGLM/ZGGGLM**

Purpose These subprograms solve a generalized linear regression model (GLM) problem:

$$\min_{x,y} \|y\|_2 \quad \text{subject to} \quad d = Ax + By$$

using a generalized *QR* factorization of matrices *A* and *B*, where *A* is an *n*-by-*m* matrix, *B* is an *n*-by-*p* matrix, and $\| \cdot \|_2$ denotes the Euclidean vector norm.

These subprograms require that $m \leq n \leq m+p$, and

$$\text{rank}(A) = m \quad \text{and} \quad \text{rank}([A \ B]) = n.$$

Under these conditions, the constraint equation is always consistent and there is a unique solution *x* and a minimal 2-norm solution *y*.

In particular, if matrix *B* is square and nonsingular, then the GLM problem is equivalent to the following weighted linear least squares problem

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

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```
INTEGER*4 info, lda, ldb, lwork, m, n, p
REAL*4    a(lda, m), b(ldb, p), d(n), work(lwork), x(m), y(p)
CALL SGGGLM (n, m, p, a, lda, b, ldb, d, x, y, work, lwork, info)
```

```
INTEGER*4 info, lda, ldb, lwork, m, n, p
REAL*8    a(lda, m), b(ldb, p), d(n), work(lwork), x(m), y(p)
CALL DGGGLM (n, m, p, a, lda, b, ldb, d, x, y, work, lwork, info)
```

```
INTEGER*4 info, lda, ldb, lwork, m, n, p
COMPLEX*8 a(lda, m), b(ldb, p), d(n), work(lwork), x(m), y(p)
CALL CGGGLM (n, m, p, a, lda, b, ldb, d, x, y, work, lwork, info)
```

```
INTEGER*4 info, lda, ldb, lwork, m, n, p
COMPLEX*16 a(lda, m), b(ldb, p), d(n), work(lwork), x(m), y(p)
CALL ZGGGLM (n, m, p, a, lda, b, ldb, d, x, y, work, lwork, info)
```

LAPACK8, available on C Series and Exemplar architectures:

```
INTEGER*8 info, lda, ldb, lwork, m, n, p
REAL*8    a(lda, m), b(ldb, p), d(n), work(lwork), x(m), y(p)
CALL SGGGLM (n, m, p, a, lda, b, ldb, d, x, y, work, lwork, info)
```

```
INTEGER*8 info, lda, ldb, lwork, m, n, p
COMPLEX*16 a(lda, m), b(ldb, p), d(n), work(lwork), x(m), y(p)
CALL CGGGLM (n, m, p, a, lda, b, ldb, d, x, y, work, lwork, info)
```

Input

- n** The number of rows of the matrices *A* and *B*. $n \geq m$ and $n \leq m+p$.
- m** The number of columns of the matrix *A*. $m \geq 0$.
- p** The number of columns of the matrix *B*. $p \geq 0$.

	a	The n -by- m matrix A .
	lda	The leading dimension of array a in the calling program unit. $lda \geq \max(1,n)$.
	b	The n -by- p matrix B .
	ldb	The leading dimension of array b in the calling program unit. $ldb \geq \max(1,n)$.
	d	The left hand side vector d of the GLM equation.
	lwork	The length of array work . $lwork \geq m + p + \max(n,m,p)$. For good performance, lwork must generally be larger. The optimum value of lwork for high performance is returned in work(1) .
Working Storage	work	Array used for work space. On successful exit, work(1) contains the optimal work space length lwork for high performance.
Output	a, b, d	Destroyed.
	x, y	The solution vectors x and y of the GLM problem.
	info	Status response: info = 0: Successful exit. info < 0: If info = $-k$, the k -th argument had an invalid value.

Notes If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

$n < 0$,
 $m < 0$,
 $p < 0$,
 $n < m$,
 $n > m+p$,
 $lda < \max(1,n)$,
 $ldb < \max(1,n)$, and
lwork too small.

Linear Equality Constraints**SGGLSE/DGGLSE/CGGLSE/ZGGLSE**

Purpose These subprograms solve the linear equality constrained least squares (LSE) problem:

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

using a generalized *RQ* factorization of matrices *A* and *B*, where *A* is an *m*-by-*n* matrix, *B* is a *p*-by-*n* matrix, and $\| \cdot \|_2$ denotes the vector Euclidean norm.

It is assumed that *B* is of rank *p*, with $p \leq n \leq m+p$, and that the null spaces of *A* and *B* intersect only trivially:

$$\text{null}(A) \cap \text{null}(B) = \{0\}.$$

The latter condition is equivalent to

$$\text{rank} \begin{bmatrix} A \\ B \end{bmatrix} = n.$$

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```
INTEGER*4 info, lda, ldb, lwork, m, n, p
REAL*4      a(lda, n), b(ldb, n), c(m), d(p), work(lwork), x(n)
CALL SGGLSE (m, n, p, a, lda, b, ldb, c, d, x, work, lwork, info)
```

```
INTEGER*4 info, lda, ldb, lwork, m, n, p
REAL*8      a(lda, n), b(ldb, n), c(m), d(p), work(lwork), x(n)
CALL DGGLSE (m, n, p, a, lda, b, ldb, c, d, x, work, lwork, info)
```

```
INTEGER*4 info, lda, ldb, lwork, m, n, p
COMPLEX*8 a(lda, n), b(ldb, n), c(m), d(p), work(lwork), x(n)
CALL CGGLSE (m, n, p, a, lda, b, ldb, c, d, x, work, lwork, info)
```

```
INTEGER*4 info, lda, ldb, lwork, m, n, p
COMPLEX*16 a(lda, n), b(ldb, n), c(m), d(p), work(lwork), x(n)
CALL ZGGLSE (m, n, p, a, lda, b, ldb, c, d, x, work, lwork, info)
```

LAPACK8, available on C Series and Exemplar architectures:

```
INTEGER*8 info, lda, ldb, lwork, m, n, p
REAL*8      a(lda, n), b(ldb, n), c(m), d(p), work(lwork), x(n)
CALL SGGLSE (m, n, p, a, lda, b, ldb, c, d, x, work, lwork, info)
```

```
INTEGER*8 info, lda, ldb, lwork, m, n, p
COMPLEX*16 a(lda, n), b(ldb, n), c(m), d(p), work(lwork), x(n)
CALL CGGLSE (m, n, p, a, lda, b, ldb, c, d, x, work, lwork, info)
```

Input

- m** The number of rows of the matrix *A*. $m \geq 0$.
- n** The number of columns of the matrices *A* and *B*. $n \geq 0$.
- p** The number of rows of the matrix *B*. $p \geq 0$.
- a** The *m*-by-*n* matrix *A* for the least squares part of the LSE problem.
- lda** The leading dimension of array *a* in the calling program unit. $lda \geq \max(1, m)$.

	b	The p -by- n matrix B for the constraint equations part of the LSE problem.
	ldb	The leading dimension of array b in the calling program unit. $ldb \geq \max(1,p)$.
	c	The right hand side vector c for the least squares part of the LSE problem.
	d	The right hand side vector d for the constraint equations part of the LSE problem.
	lwork	The length of array work . $lwork \geq n + p + \max(m,n,p)$. For good performance, lwork must generally be larger. The optimum value of lwork for high performance is returned in work(1) .
Working Storage	work	Array used for work space. On successful exit, work(1) contains the optimal work space length lwork for high performance.
Output	a, b	Destroyed.
	c	The Euclidean norm of the residual vector $Ax - c$ is given by the Euclidean norm of elements $n-p+1$ to m of c .
	d	Destroyed.
	x	The solution vector x of the LSE problem.
	info	Status response: info = 0: Successful exit. info < 0: If info = $-k$, the k -th argument had an invalid value.
Notes		If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

m < 0,
n < 0,
p < 0,
n < **p**,
n > **m+p**,
lda < $\max(1,m)$,
ldb < $\max(1,p)$, and
lwork too small.

Computational Subprograms for Orthogonal Factorizations

Overview

This chapter describes the computational subprograms to compute several forms of orthogonal factorizations of a matrix, to compute several forms of the generalized orthogonal factorization of a pair of matrices, and to make use of the resulting orthogonal matrix.

This chapter also shows how to use these and other LAPACK subprograms to solve linear least squares problems.

These operations are performed for both real and complex general matrices:

- compute the QR factorization
- compute the RQ factorization
- compute the QL factorization
- compute the LQ factorization
- compute the QR factorization using column pivoting
- compute the generalized QR factorization of a pair of matrices
- compute the generalized RQ factorization of a pair of matrices
- multiply another matrix by the Q matrix produced by one of these factorizations
- generate the Q matrix from the factored form produced during the above computations

Chapter Objectives

After you read this chapter you will:

- know how orthogonal and unitary matrices are represented in factored form
- know how to use the described subprograms

What You Need to Know to Use These Subprograms

The LAPACK subprograms in this chapter are organized so that it is usually necessary to call two or more subprograms to perform the above operations. This division of labor significantly enhances the number of processing options you may form by combining these and other LAPACK subprograms.

Combining Computational Subprograms

When you use the computational subprograms instead of calling the drivers, you usually must put two or more of them together to carry out your entire computation. This section shows how to assemble an algorithm from several computational subprograms. In these examples, we assume your matrix is a 10-by-5 real general matrix stored in a single precision array large enough to handle a 20-by-10 problem. We do not show how the matrix, or the right hand side, if needed, are generated.

Solving a Full-Rank Linear Least Squares Problem

The following code segment shows how to solve a full-rank linear least squares problem

$$\min_x \|b - Ax\|_2$$

SGEQRF computes the QR factorization of the coefficient matrix A . Then SORMQR computes $Q^T b$. Finally, STRTRS, a computational subprogram for linear equations, computes the solution vector $x = R^{-1}(Q^T b)$, overwriting the right hand side vector B with it.

```

INTEGER*4 INFO, LDA, LDB, M, MMAX, N, NMAX, NRHS
PARAMETER ( MMAX = 20 )
PARAMETER ( NMAX = 10 )
PARAMETER ( LDA = MMAX )
PARAMETER ( LDB = MMAX )
PARAMETER ( LWORK = NMAX )
REAL*4 A(LDA,NMAX), B(LDB), TAU(NMAX), WORK(LWORK)

M      = 10
N      = 5
NRHS   = 1
CALL SGEQRF (M, N, A, LDA, TAU, WORK, LWORK, INFO)
CALL SORMQR ('Left', 'Transposed', M, NRHS, N, A, LDA, TAU, B, LDB,
&           WORK, LWORK, INFO)
CALL STRTRS ('Upper', 'NonTransposed', 'NonUnit', N, NRHS, A, LDA,
&           B, LDB, INFO)

```

Determine the Effective Rank of a Matrix

The QR factorization with column pivoting can be used to determine the effective numerical rank of a matrix less expensively than the more robust Singular Value Decomposition. SLANGE is used to compute $\|A\|_1$, from which a tolerance is determined to measure smallness. SGEQPF computes the QR factorization of A with column pivoting. The subsequent loop determines the rank by counting the number of nonsmall diagonal elements of the R matrix.

```

INTEGER*4 INFO, LDA, M, N, NMAX, RANK
PARAMETER ( NMAX = 10 )
PARAMETER ( LDA = 20 )
REAL*4 ANORM, SLAMCH, SLANGE, TOL
INTEGER*4 JPVT(NMAX), RANK
REAL*4 A(LDA,NMAX), TAU(NMAX), WORK(3*NMAX)

M = 10
N = 5
ANORM = SLANGE('1', M, N, A, LDA, WORK)
TOL = MAX(M, N) * ANORM * SLAMCH('Precision')

DO 10 I = 1, N
    JPVT(I) = 0
10 CONTINUE
CALL SGEQPF (M, N, A, LDA, JPVT, TAU, WORK, INFO)

RANK = 0
DO 20 I = 1, MIN(M, N)
    IF ( ABS(A(I,I)) .GT. TOL ) RANK = RANK + 1
20 CONTINUE

```

Supplemental Reading

- Anderson, E. *et al.* *LAPACK Users' Guide*, Second Edition. Philadelphia, PA: Society for Industrial and Applied Mathematics. 1995.
- Forsythe, G. and C.B. Moler. *Computer Solution of Linear Algebraic Systems*. Englewood Cliffs, NJ: Prentice-Hall, Inc. 1967.
- Golub, G.H. and C.F. Van Loan. *Matrix Computations*, 2nd Edition. Baltimore, MD: The Johns Hopkins University Press. 1989.

Subprogram Descriptions

<i>LQ</i> Factorization of a General Full Matrix SGELQF, DGELQF, CGELQF, ZGELQF	6-4
<i>QL</i> Factorization of a General Full Matrix SGEQLF, DGEQLF, CGEQLF, ZGEQLF	6-6
<i>QR</i> Factorization of a General Full Matrix with Column Pivoting SGEQPF, DGEQPF, CGEQPF, ZGEQPF	6-8
<i>QR</i> Factorization of a General Full Matrix SGEQRF, DGEQRF, CGEQRF, ZGEQRF	6-11
<i>RQ</i> Factorization of a General Full Matrix SGERQF, DGERQF, CGERQF, ZGERQF	6-13
Generalized <i>QR</i> Factorization SGGQRF, DGGQRF, CGGQRF, ZGGQRF	6-15
Generalized <i>RQ</i> Factorization SGGRQF, DGRQF, CGGRQF, ZGRQF	6-19
Generate the <i>Q</i> Matrix in Unfactored Form from an <i>LQ</i> Factorization SORGLQ, DORGLQ, CUNGLQ, ZUNGLQ	6-23
Generate the <i>Q</i> Matrix in Unfactored Form from a <i>QL</i> Factorization SORGQL, DORGQL, CUNGQL, ZUNGQL	6-25
Generate the <i>Q</i> Matrix in Unfactored Form from a <i>QR</i> Factorization SORGQR, DORGQR, CUNGQR, ZUNGQR	6-27
Generate the <i>Q</i> Matrix in Unfactored Form from an <i>RQ</i> Factorization SORGRQ, DORGRQ, CUNGRQ, ZUNGRQ	6-29
Multiply a General Matrix by <i>Q</i> from an <i>LQ</i> Factorization SORMLQ, DORMLQ, CUNMLQ, ZUNMLQ	6-31
Multiply a General Matrix by <i>Q</i> from a <i>QL</i> Factorization SORMQL, DORMQL, CUNMQL, ZUNMQL	6-34
Multiply a General Matrix by <i>Q</i> from a <i>QR</i> Factorization SORMQR, DORMQR, CUNMQR, ZUNMQR	6-37
Multiply a General Matrix by <i>Q</i> from an <i>RQ</i> Factorization SORMRQ, DORMRQ, CUNMRQ, ZUNMRQ	6-40
<i>RQ</i> Factorization of an Upper Trapezoidal Matrix STZRQF, DTZRQF, CTZRQF, ZTZRQF	6-43

Purpose These subprograms compute the *LQ* factorization of a general *m*-by-*n* matrix *A*. The factorization has the form $A = LQ$, where *L* is a lower triangular matrix (lower trapezoidal if $m > n$) and *Q* is an orthogonal or unitary matrix. The computed matrix *Q* is represented as a product of elementary reflection matrices

$$Q = H_k H_{k-1} \cdots H_1$$

where $k = \min(m, n)$. Each H_i has the form

$$H_i = I - \tau_i v_i v_i^*$$

where τ_i is a scalar and v_i is an *n*-dimensional vector whose first $i-1$ components are zero and whose *i*th component is 1.

Additional subprograms are provided to make it easier to use the matrix *Q* in its factored form. The operations furnished are to multiply a general matrix by the *Q* matrix, and to generate (expand) the *Q* matrix into regular, unfactored form. The names of these companion subprograms depend on the data type:

Data Type	<i>LQ</i> Factorization	Generate <i>Q</i>	Multiply Matrix by <i>Q</i>
REAL*4	SGELQF	SORGLQ	SORMLQ
REAL*8	DGELQF	DORGLQ	DORMLQ
COMPLEX*8	CGELQF	CUNGLQ	CUNMLQ
COMPLEX*16	ZGELQF	ZUNGLQ	ZUNMLQ

Usage

LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

INTEGER*4 info, lda, lwork, m, n
 REAL*4 a(lda, n), tau(min(m,n)), work(lwork)
 CALL SGELQF (m, n, a, lda, tau, work, lwork, info)

INTEGER*4 info, lda, lwork, m, n
 REAL*8 a(lda, n), tau(min(m,n)), work(lwork)
 CALL DGELQF (m, n, a, lda, tau, work, lwork, info)

INTEGER*4 info, lda, lwork, m, n
 COMPLEX*8 a(lda, n), tau(min(m,n)), work(lwork)
 CALL CGELQF (m, n, a, lda, tau, work, lwork, info)

INTEGER*4 info, lda, lwork, m, n
 COMPLEX*16 a(lda, n), tau(min(m,n)), work(lwork)
 CALL ZGELQF (m, n, a, lda, tau, work, lwork, info)

LAPACK8, available on C Series and Exemplar architectures:

INTEGER*8 info, lda, lwork, m, n
 REAL*8 a(lda, n), tau(min(m,n)), work(lwork)
 CALL SGELQF (m, n, a, lda, tau, work, lwork, info)

INTEGER*8 info, lda, lwork, m, n
 COMPLEX*16 a(lda, n), tau(min(m,n)), work(lwork)
 CALL CGELQF (m, n, a, lda, tau, work, lwork, info)

Input	m	The number of rows of the matrix A . $m \geq 0$.
	n	The number of columns of the matrix A . $n \geq 0$.
	a	The m -by- n matrix A to be factored.
	lda	The leading dimension of array a in the calling program unit. $lda \geq \max(1,m)$.
	lwork	The length of array work . $lwork \geq \max(1,m)$. For good performance, lwork must generally be larger. The optimum value of lwork for high performance is returned in work(1) .
Working Storage	work	An array used for work space. On successful exit, work(1) contains the optimal work space length lwork for high performance.
Output	a	On successful exit, the factors L and Q from the factorization $A = LQ$, stored as follows: If $m \leq n$, the elements on and below the principal diagonal of a contain the m -by- m lower triangular matrix L . If $m > n$, the elements on and below the principal diagonal of a contain the n -by- m lower trapezoidal matrix L . The elements above the principal diagonal of the i th row of a , $i = 1, 2, \dots, \min(m,n-1)$, contain components $i+1$ to n of v_i .
	tau	On successful exit, the scalar factors, τ_i , $i = 1, 2, \dots, \min(m,n)$, of the elementary reflection matrices, $H_i = I - \tau_i v_i v_i^*$.
	info	Status response: info = 0: Successful exit. info < 0: If info = $-k$, the k -th argument had an invalid value.

Notes

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

$m < 0$,
 $n < 0$,
 $lda < \max(1,m)$, and
 $lwork < \max(1,m)$.

Purpose These subprograms compute the *QL* factorization of a general *m*-by-*n* matrix *A*. The factorization has the form $A = QL$, where *Q* is an orthogonal or unitary matrix and *L* is a lower triangular matrix (lower trapezoidal if $m < n$). The computed matrix *Q* is represented as a product of elementary reflection matrices

$$Q = H_k H_{k-1} \cdots H_1$$

where $k = \min(m, n)$. Each H_i has the form

$$H_i = I - \tau_i v_i v_i^*$$

where τ_i is a scalar and v_i is an *m*-dimensional vector whose $(n-k+i)$ th component is 1 and whose last $k-i$ components are zero.

Additional subprograms are provided to make it easier to use the matrix *Q* in its factored form. The operations furnished are to multiply a general matrix by the *Q* matrix, and to generate (expand) the *Q* matrix into regular, unfactored form. The names of these companion subprograms depend on the data type:

Data Type	<i>QL</i> Factorization	Generate <i>Q</i>	Multiply Matrix by <i>Q</i>
REAL*4	SGEQLF	SORGQL	SORMQL
REAL*8	DGEQLF	DORGQL	DORMQL
COMPLEX*8	CGEQLF	CUNGQL	CUNMQL
COMPLEX*16	ZGEQLF	ZUNGQL	ZUNMQL

Usage

LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

INTEGER*4 info, lda, lwork, m, n
REAL*4 a(lda, n), tau(min(m,n)), work(lwork)
CALL SGEQLF (m, n, a, lda, tau, work, lwork, info)

INTEGER*4 info, lda, lwork, m, n
REAL*8 a(lda, n), tau(min(m,n)), work(lwork)
CALL DGEQLF (m, n, a, lda, tau, work, lwork, info)

INTEGER*4 info, lda, lwork, m, n
COMPLEX*8 a(lda, n), tau(min(m,n)), work(lwork)
CALL CGEQLF (m, n, a, lda, tau, work, lwork, info)

INTEGER*4 info, lda, lwork, m, n
COMPLEX*16 a(lda, n), tau(min(m,n)), work(lwork)
CALL ZGEQLF (m, n, a, lda, tau, work, lwork, info)

LAPACK8, available on C Series and Exemplar architectures:

INTEGER*8 info, lda, lwork, m, n
REAL*8 a(lda, n), tau(min(m,n)), work(lwork)
CALL SGEQLF (m, n, a, lda, tau, work, lwork, info)

INTEGER*8 info, lda, lwork, m, n
COMPLEX*16 a(lda, n), tau(min(m,n)), work(lwork)
CALL CGEQLF (m, n, a, lda, tau, work, lwork, info)

Input	m	The number of rows of the matrix A . $m \geq 0$.
	n	The number of columns of the matrix A . $n \geq 0$.
	a	The m -by- n matrix A to be factored.
	lda	The leading dimension of array a in the calling program unit. $lda \geq \max(1,m)$.
	lwork	The length of array work . $lwork \geq \max(1,n)$. For good performance, lwork must generally be larger. The optimum value of lwork for high performance is returned in work(1) .
Working Storage	work	An array used for work space. On successful exit, work(1) contains the optimal work space length lwork for high performance.
Output	a	On successful exit, the factors Q and L from the factorization $A = QL$, stored as follows:
		If $m > n$, the elements on and below the $(m-n)$ -th subdiagonal of a contain the n -by- n lower triangular matrix L .
		If $m = n$, the elements on and below the principal diagonal of a contain the n -by- n lower triangular matrix L .
		If $m < n$, the elements on and below the $(n-m)$ -th superdiagonal of a contain the m -by- n lower trapezoidal matrix L .
		The remaining elements of a contain components of the v vectors:
	If $m \geq n$, column i of a , $i = 1, 2, \dots, n$, contains components 1 to $m-n+i-1$ of v_i .	
If $m < n$, column $n-m+i$ of a , $i = 2, 3, \dots, n$, contains components 1 to $i-1$ of v_i .		
tau	On successful exit, the scalar factors, τ_i , $i = 1, 2, \dots, \min(m,n)$, of the elementary reflection matrices, $H_i = I - \tau_i v_i v_i^*$.	
info	Status response:	
	info = 0: Successful exit.	
	info < 0: If info = $-k$, the k -th argument had an invalid value.	

Notes If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

$m < 0$,
 $n < 0$,
 $lda < \max(1,m)$, and
 $lwork < \max(1,n)$.

Purpose

These subprograms compute the QR factorization of a general m -by- n matrix A using column pivoting. The factorization has the form $AP = QR$, where Q is an orthogonal or unitary matrix, R is an upper triangular matrix (upper trapezoidal if $m < n$), and P is a permutation matrix chosen so that if A is of rank r then the first r columns of Q span the range of A .

The computed matrix Q is represented as a product of elementary reflection matrices

$$Q = H_1 H_2 \cdots H_k$$

where $k = \min(m, n)$. Each H_i has the form

$$H_i = I - \tau_i v_i v_i^*$$

where τ_i is a scalar and v_i is an m -dimensional vector whose first $i-1$ components are zero and whose i th component is 1.

Additional subprograms are provided to make it easier to use the matrix Q in its factored form. The operations furnished are to multiply a general matrix by the Q matrix, and to generate (expand) the Q matrix into regular, unfactored form. The names of these companion subprograms depend on the data type:

Data Type	QR Factorization	Generate Q	Multiply Matrix by Q
REAL*4	SGEQPF	SORGQR	SORMQR
REAL*8	DGEQPF	DORGQR	DORMQR
COMPLEX*8	CGEQPF	CUNGQR	CUNMQR
COMPLEX*16	ZGEQPF	ZUNGQR	ZUNMQR

Usage

LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```
INTEGER*4 info, lda, m, n
INTEGER*4 jpvt(n)
REAL*4 a(lda, n), tau(min(m,n)), work(3*n)
CALL SGEQPF (m, n, a, lda, jpvt, tau, work, info)
```

```
INTEGER*4 info, lda, m, n
INTEGER*4 jpvt(n)
REAL*8 a(lda, n), tau(min(m,n)), work(3*n)
CALL DGEQPF (m, n, a, lda, jpvt, tau, work, info)
```

```
INTEGER*4 info, lda, m, n
INTEGER*4 jpvt(n)
REAL*4 rwork(2*n)
COMPLEX*8 a(lda, n), tau(min(m,n)), work(n)
CALL CGEQPF (m, n, a, lda, jpvt, tau, work, rwork, info)
```

```
INTEGER*4 info, lda, m, n
INTEGER*4 jpvt(n)
REAL*8 rwork(2*n)
COMPLEX*16 a(lda, n), tau(min(m,n)), work(n)
CALL ZGEQPF (m, n, a, lda, jpvt, tau, work, rwork, info)
```

LAPACK8, available on C Series and Exemplar architectures:

```

INTEGER*8 info, lda, m, n
INTEGER*8 jpvt(n)
REAL*8    a(lda, n), tau(min(m,n)), work(3*n)
CALL SGEQPF (m, n, a, lda, jpvt, tau, work, info)

```

```

INTEGER*8 info, lda, m, n
INTEGER*8 jpvt(n)
REAL*8    rwork(2*n)
COMPLEX*16 a(lda, n), tau(min(m,n)), work(n)
CALL CGEQPF (m, n, a, lda, jpvt, tau, work, rwork, info)

```

Input	m	The number of rows of the matrix A . $m \geq 0$.
	n	The number of columns of the matrix A . $n \geq 0$.
	a	The m -by- n matrix A to be factored.
	lda	The leading dimension of array a in the calling program unit. $lda \geq \max(1, m)$.
	jpvt	If $jpvt(i) \neq 0$, column i of A is permuted to the front of matrix AP and will not be subject to further algorithmic pivoting. If $jpvt(i) = 0$, column i of A is a free column and will be pivoted as the algorithm determines.
Working Storage	work,	Arrays used for work space.
	rwork	
Output	a	On successful exit, the factors Q and R from the factorization $AP = QR$, stored as follows: The elements on and above the diagonal of a contain the $\min(m, n)$ -by- n upper triangular or upper trapezoidal matrix R . The elements below the diagonal of the i th column of a , $i = 1, 2, \dots, \min(m-1, n)$, contain the last $m-i$ components of v_i . See "Purpose" for details.
	jpvt	If $jpvt(i) = k$, then column k of A was permuted to column i of the matrix AP . Thus, the i th column of P is e_k , the k th canonical unit vector, $(0, 0, \dots, 0, 1, 0, \dots, 0)^T$, where the "1" is in the k th position.
	tau	On successful exit, the scalar factors, τ_i , $i = 1, 2, \dots, \min(m, n)$, of the elementary reflection matrices, $H_i = I - \tau_i v_i v_i^*$.
	info	Status response: info = 0: Successful exit. info < 0: If info = $-k$, the k -th argument had an invalid value.

Notes

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

$m < 0$,
 $n < 0$, and
 $lda < \max(1, m)$.

QR Factorization of General Matrix**SGEQRF/DGEQRF/.../ZGEQRF****Purpose**

These subprograms compute the QR factorization of a general m -by- n matrix A . The factorization has the form $A = QR$, where Q is an orthogonal or unitary matrix and R is an upper triangular matrix (upper trapezoidal if $m < n$). The computed matrix Q is represented as a product of elementary reflection matrices

$$Q = H_1 H_2 \cdots H_k$$

where $k = \min(m, n)$. Each H_i has the form

$$H_i = I - \tau_i v_i v_i^*$$

where τ_i is a scalar and v_i is an m -dimensional vector whose first $i-1$ components are zero and whose i th component is 1.

Additional subprograms are provided to make it easier to use the matrix Q in its factored form. The operations furnished are to multiply a general matrix by the Q matrix, and to generate (expand) the Q matrix into regular, unfactored form. The names of these companion subprograms depend on the data type:

Data Type	QR Factorization	Generate Q	Multiply Matrix by Q
REAL*4	SGEQRF	SORGQR	SORMQR
REAL*8	DGEQRF	DORGQR	DORMQR
COMPLEX*8	CGEQRF	CUNGQR	CUNMQR
COMPLEX*16	ZGEQRF	ZUNGQR	ZUNMQR

Usage

LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```
INTEGER*4 info, lda, lwork, m, n
REAL*4    a(lda, n), tau(min(m,n)), work(lwork)
CALL SGEQRF (m, n, a, lda, tau, work, lwork, info)
```

```
INTEGER*4 info, lda, lwork, m, n
REAL*8    a(lda, n), tau(min(m,n)), work(lwork)
CALL DGEQRF (m, n, a, lda, tau, work, lwork, info)
```

```
INTEGER*4 info, lda, lwork, m, n
COMPLEX*8 a(lda, n), tau(min(m,n)), work(lwork)
CALL CGEQRF (m, n, a, lda, tau, work, lwork, info)
```

```
INTEGER*4 info, lda, lwork, m, n
COMPLEX*16 a(lda, n), tau(min(m,n)), work(lwork)
CALL ZGEQRF (m, n, a, lda, tau, work, lwork, info)
```

LAPACK8, available on C Series and Exemplar architectures:

```
INTEGER*8 info, lda, lwork, m, n
REAL*8    a(lda, n), tau(min(m,n)), work(lwork)
CALL SGEQRF (m, n, a, lda, tau, work, lwork, info)
```

```
INTEGER*8 info, lda, lwork, m, n
COMPLEX*16 a(lda, n), tau(min(m,n)), work(lwork)
CALL CGEQRF (m, n, a, lda, tau, work, lwork, info)
```

Input	m	The number of rows of the matrix A . $m \geq 0$.
	n	The number of columns of the matrix A . $n \geq 0$.
	a	The m -by- n matrix A to be factored.
	lda	The leading dimension of array a in the calling program unit. $lda \geq \max(1,m)$.
	lwork	The length of array $work$. $lwork \geq \max(1,n)$. For good performance, $lwork$ must generally be larger. The optimum value of $lwork$ for high performance is returned in $work(1)$.
Working Storage	work	An array used for work space. On successful exit, $work(1)$ contains the optimal work space length $lwork$ for high performance.
Output	a	On successful exit, the factors Q and R from the factorization $A = QR$, stored as follows: If $m \geq n$, the elements on and above the principal diagonal of a contain the n -by- n upper triangular matrix R . If $m < n$, the elements on and above the principal diagonal of a contain the m -by- n upper trapezoidal matrix R . The elements below the principal diagonal of the j th column of a , $j = 1, 2, \dots, \min(m-1,n)$, contain components $j+1$ to m of v_j .
	tau	On successful exit, the scalar factors, τ_i , $i = 1, 2, \dots, \min(m,n)$, of the elementary reflection matrices, $H_i = I - \tau_i v_i v_i^*$.
	info	Status response: info = 0: Successful exit. info < 0: If info = $-k$, the k -th argument had an invalid value.
	Notes	If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

m < 0,
n < 0,
lda < $\max(1,m)$, and
lwork < $\max(1,n)$.

RQ* Factorization of General Matrix*SGERQF/DGERQF/.../ZGERQF****Purpose**

These subprograms compute the *RQ* factorization of a general *m*-by-*n* matrix *A*. The factorization has the form $A = RQ$, where *R* is an upper triangular matrix (upper trapezoidal if $m > n$) and *Q* is an orthogonal or unitary matrix. The computed matrix *Q* is represented as a product of elementary reflection matrices

$$Q = H_1 H_2 \cdots H_k$$

where $k = \min(m, n)$. Each H_i has the form

$$H_i = I - \tau_i v_i v_i^*$$

where τ_i is a scalar and v_i is an *m*-dimensional vector whose $(n-k+i)$ th component is 1 and whose last $k-i$ components are zero.

Additional subprograms are provided to make it easier to use the matrix *Q* in its factored form. The operations furnished are to multiply a general matrix by the *Q* matrix, and to generate (expand) the *Q* matrix into regular, unfactored form. The names of these companion subprograms depend on the data type:

Data Type	<i>RQ</i> Factorization	Generate <i>Q</i>	Multiply Matrix by <i>Q</i>
REAL*4	SGERQF	SORGRQ	SORMRQ
REAL*8	DGERQF	DORGRQ	DORMRQ
COMPLEX*8	CGERQF	CUNGRQ	CUNMRQ
COMPLEX*16	ZGERQF	ZUNGRQ	ZUNMRQ

Usage

LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```
INTEGER*4 info, lda, lwork, m, n
REAL*4      a(lda, n), tau(min(m,n)), work(lwork)
CALL SGERQF (m, n, a, lda, tau, work, lwork, info)
```

```
INTEGER*4 info, lda, lwork, m, n
REAL*8      a(lda, n), tau(min(m,n)), work(lwork)
CALL DGERQF (m, n, a, lda, tau, work, lwork, info)
```

```
INTEGER*4 info, lda, lwork, m, n
COMPLEX*8 a(lda, n), tau(min(m,n)), work(lwork)
CALL CGERQF (m, n, a, lda, tau, work, lwork, info)
```

```
INTEGER*4 info, lda, lwork, m, n
COMPLEX*16 a(lda, n), tau(min(m,n)), work(lwork)
CALL ZGERQF (m, n, a, lda, tau, work, lwork, info)
```

LAPACK8, available on C Series and Exemplar architectures:

```
INTEGER*8 info, lda, lwork, m, n
REAL*8      a(lda, n), tau(min(m,n)), work(lwork)
CALL SGERQF (m, n, a, lda, tau, work, lwork, info)
```

```
INTEGER*8 info, lda, lwork, m, n
COMPLEX*16 a(lda, n), tau(min(m,n)), work(lwork)
CALL CGERQF (m, n, a, lda, tau, work, lwork, info)
```

Input	m	The number of rows of the matrix A . $m \geq 0$.
	n	The number of columns of the matrix A . $n \geq 0$.
	a	The m -by- n matrix A to be factored.
	lda	The leading dimension of array a in the calling program unit. $lda \geq \max(1,m)$.
	lwork	The length of array work . $lwork \geq \max(1,m)$. For good performance, lwork must generally be larger. The optimum value of lwork for high performance is returned in work(1) .
Working Storage	work	An array used for work space. On successful exit, work(1) contains the optimal work space length lwork for high performance.
Output	a	On successful exit, the factors R and Q from the factorization $A = RQ$, stored as follows: If $m < n$, the elements on and above the $(n-m)$ -th superdiagonal of a contain the m -by- m upper triangular matrix R . If $m = n$, the elements on and above the principal diagonal of a contain the m -by- m upper triangular matrix R . If $m > n$, the elements on and above the $(m-n)$ -th subdiagonal of a contain the m -by- n upper trapezoidal matrix R . The remaining elements of a contain components of the v vectors: If $m \leq n$, row i of a , $i = 1, 2, \dots, m$, contains components 1 to $n-m+i-1$ of v_i . If $m > n$, row $m-n+i$ of a , $i = 2, 3, \dots, n$, contains components 1 to $i-1$ of v_i .
	tau	On successful exit, the scalar factors, τ_i , $i = 1, 2, \dots, \min(m,n)$, of the elementary reflection matrices, $H_i = I - \tau_i v_i v_i^*$.
	info	Status response: info = 0: Successful exit. info < 0: If info = $-k$, the k -th argument had an invalid value.
	Notes	If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are $m < 0$, $n < 0$, $lda < \max(1,m)$, and $lwork < \max(1,m)$.

Generalized QR Factorization

SGGQRF/DGGQRF/CGGQRF/ZGGQRF

Purpose These subprograms compute the generalized QR (GQR) factorization of an m -by- n matrix A and an m -by- p matrix B . The factorization has the form

$$A = QR, \quad B = QTZ$$

where Q is an m -by- m orthogonal or unitary matrix, Z is a p -by- p orthogonal or unitary matrix, R assumes one of the forms:

$$\text{if } m \leq n, R = [R_{11} \ R_{12}], \quad \text{if } m > n, R = \begin{bmatrix} R_{11} \\ 0 \end{bmatrix},$$

where R_{11} is upper triangular, and T assumes one of the forms:

$$\text{if } m \leq p, T = \begin{bmatrix} 0 & T_{12} \end{bmatrix}, \quad \text{if } m > p, T = \begin{bmatrix} T_{11} \\ T_{21} \end{bmatrix},$$

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

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

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

The computed matrix Q is represented as a product of elementary reflection matrices

$$Q = H_1 H_2 \cdots H_k$$

where $k = \min(m, n)$. Each H_i has the form

$$H_i = I - \tau_i v_i v_i^*$$

where τ_i is a scalar and v_i is an m -dimensional vector whose first $i-1$ components are zero and whose i th component is 1.

Z is represented as a product of elementary reflection matrices

$$Z = \bar{H}_1 \bar{H}_2 \cdots \bar{H}_{\bar{k}}$$

where $\bar{k} = \min(m, p)$. Each \bar{H}_i has the form

$$\bar{H}_i = I - \bar{\tau}_i \bar{v}_i \bar{v}_i^*$$

where $\bar{\tau}_i$ is a scalar and \bar{v}_i is an m -dimensional vector whose $(p-\bar{k}+i)$ th component is 1 and whose last $\bar{k}-i$ components are zero.

Additional subprograms are provided to make it easier to use Q and Z in their factored forms. The operations furnished are to multiply a general matrix by Q or Z , and to generate (expand) Q or Z into regular, unfactored form. The names of these companion subprograms depend on the data type:

Data Type	GQR Factorization	Generate		Multiply Matrix by	
		Q	Z	Q	Z
REAL*4	SGGQRF	SORGQR	SORGRQ	SORMQR	SORMRQ
REAL*8	DGGQRF	DORGQR	DORGRQ	DORMQR	DORMRQ
COMPLEX*8	CGGQRF	CUNGQR	CUNGRQ	CUNMQR	CUNMRQ
COMPLEX*16	ZGGQRF	ZUNGQR	ZUNGRQ	ZUNMQR	ZUNMRQ

Usage

LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```

INTEGER*4 info, lda, ldb, lwork, m, n, p
REAL*4 a(lda, n), b(ldb, p), taua(min(m,n)), taub(min(m,p)),
work(lwork)
CALL SGGQRF (m, n, p, a, lda, taua, b, ldb, taub, work, lwork,
info)

```

```

INTEGER*4 info, lda, ldb, lwork, m, n, p
REAL*8 a(lda, n), b(ldb, p), taua(min(m,n)), taub(min(m,p)),
work(lwork)
CALL DGGQRF (m, n, p, a, lda, taua, b, ldb, taub, work, lwork,
info)

```

```

INTEGER*4 info, lda, ldb, lwork, m, n, p
COMPLEX*8 a(lda, n), b(ldb, p), taua(min(m,n)), taub(min(m,p)),
work(lwork)
CALL CGGQRF (m, n, p, a, lda, taua, b, ldb, taub, work, lwork,
info)

```

```

INTEGER*4 info, lda, ldb, lwork, m, n, p
COMPLEX*16 a(lda, n), b(ldb, p), taua(min(m,n)), taub(min(m,p)),
work(lwork)
CALL ZGGQRF (m, n, p, a, lda, taua, b, ldb, taub, work, lwork,
info)

```

LAPACK8, available on C Series and Exemplar architectures:

```

INTEGER*8 info, lda, ldb, lwork, m, n, p
REAL*8 a(lda, n), b(ldb, p), taua(min(m,n)), taub(min(m,p)),
work(lwork)
CALL SGGQRF (m, n, p, a, lda, taua, b, ldb, taub, work, lwork,
info)

```

```

INTEGER*8 info, lda, ldb, lwork, m, n, p
COMPLEX*16 a(lda, n), b(ldb, p), taua(min(m,n)), taub(min(m,p)),
work(lwork)
CALL CGGQRF (m, n, p, a, lda, taua, b, ldb, taub, work, lwork,
info)

```

Input	m	The number of rows of the matrices A and B . $m \geq 0$.
	n	The number of columns of the matrix A . $n \geq 0$.
	p	The number of columns of the matrix B . $p \geq 0$.
	a	The m -by- n matrix A .
	lda	The leading dimension of array a in the calling program unit. $lda \geq \max(1,m)$.
	b	The m -by- p matrix B .
	ldb	The leading dimension of array b in the calling program unit. $ldb \geq \max(1,m)$.
	lwork	The length of array $work$. $lwork \geq \max(1,m,n,p)$. For good performance, $lwork$ must generally be larger. The optimum value of $lwork$ for high performance is returned in $work(1)$.
Working Storage	work	An array used for work space. On successful exit, $work(1)$ contains the optimal work space length $lwork$ for high performance.
Output	a	On successful exit, the factors Q and R from the factorization $A = QR$, stored as follows: If $m \geq n$, the elements on and above the principal diagonal of a contain the n -by- n upper triangular matrix R . If $m < n$, the elements on and above the principal diagonal of a contain the m -by- n upper trapezoidal matrix R . The elements below the principal diagonal of the j th column of a , $j = 1, 2, \dots, \min(m-1,n)$, contain components $j+1$ to m of v_j .
	taua	On successful exit, the scalar factors, τ_i , $i = 1, 2, \dots, \min(m,n)$, of the elementary reflection matrices, $H_i = I - \tau_i v_i v_i^*$.
	b	On successful exit, the factors T and Z from the factorization $B = QTZ$, stored as follows: If $m < p$, the elements on and above the $(p-m)$ -th superdiagonal of b contain the m -by- m upper triangular matrix T . If $m = p$, the elements on and above the principal diagonal of b contain the m -by- m upper triangular matrix T . If $m > p$, the elements on and above the $(m-p)$ -th subdiagonal of b contain the m -by- p upper trapezoidal matrix R . The remaining elements of b contain components of the \bar{v} vectors: If $m \leq p$, row i of b , $i = 1, 2, \dots, m$, contains components 1 to $p-m+i-1$ of \bar{v}_i . If $m > p$, row $m-p+i$ of b , $i = 2, 3, \dots, p$, contains components 1 to $i-1$ of \bar{v}_i .

taub On successful exit, the scalar factors, $\bar{\tau}_i$, $i = 1, 2, \dots, \min(\mathbf{m}, \mathbf{p})$, of the elementary reflection matrices, $\bar{H}_i = I - \bar{\tau}_i \bar{v}_i \bar{v}_i^*$.

info Status response:

info = 0: Successful exit.

info < 0: If **info** = $-k$, the k -th argument had an invalid value.

Notes

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

m < 0,
n < 0,
p < 0,
lda < max(1,**m**),
ldb < max(1,**m**), and
lwork < max(1,**m**,**n**,**p**).

Generalized RQ Factorization**SGGRQF/DGGRQF/CGGRQF/ZGGRQF**

Purpose These subprograms compute the generalized RQ (GRQ) factorization of an m -by- p matrix A and an n -by- p matrix B . The factorization has the form

$$A = RQ, \quad B = ZTQ$$

where Q is an m -by- m orthogonal or unitary matrix, Z is a p -by- p orthogonal or unitary matrix, R assumes one of the forms:

$$\text{if } m \leq p, R = \begin{bmatrix} R_{11} \\ 0 & R_{12} \end{bmatrix}, \quad \text{if } m > p, R = \begin{bmatrix} R_{11} \\ R_{21} \end{bmatrix},$$

where R_{12} or R_{21} is an upper triangular matrix, and T assumes one of the forms:

$$\text{if } n < p, T = \begin{bmatrix} T_{11} & T_{12} \end{bmatrix}, \quad \text{if } n \geq p, T = \begin{bmatrix} T_{11} \\ 0 \end{bmatrix},$$

where T_{11} is upper triangular.

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

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

The computed matrix Q is represented as a product of elementary reflection matrices

$$Q = H_1 H_2 \cdots H_k$$

where $k = \min(m, p)$. Each H_i has the form

$$H_i = I - \tau_i v_i v_i^*$$

where τ_i is a scalar and v_i is a p -dimensional vector whose $(i-k+i)$ th component is 1 and whose last $k-i$ components are zero.

Z is represented as a product of elementary reflection matrices

$$Z = \bar{H}_1 \bar{H}_2 \cdots \bar{H}_{\bar{k}}$$

where $\bar{k} = \min(n, p)$. Each \bar{H}_i has the form

$$\bar{H}_i = I - \bar{\tau}_i \bar{v}_i \bar{v}_i^*$$

where $\bar{\tau}_i$ is a scalar and \bar{v}_i is an m -dimensional vector whose first $i-i$ components are zero and whose i th component is 1.

Additional subprograms are provided to make it easier to use Q and Z in their factored forms. The operations furnished are to multiply a general matrix by Q or Z , and to generate (expand) Q or Z into regular, unfactored form. The names of these companion subprograms depend on the data type:

Data Type	GRQ Factorization	Generate		Multiply Matrix by	
		Q	Z	Q	Z
REAL*4	SGGRQF	SORGRQ	SORGQR	SORMRQ	SORMQR
REAL*8	DGGRQF	DORGRQ	DORGQR	DORMRQ	DORMQR
COMPLEX*8	CGGRQF	CUNGRQ	CUNGQR	CUNMRQ	CUNMQR
COMPLEX*16	ZGGRQF	ZUNGRQ	ZUNGQR	ZUNMRQ	ZUNMQR

Usage

LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

INTEGER*4 info, lda, ldb, lwork, m, n, p
REAL*4 a(lda, p), b(ldb, p), taua(min(m,p)), taub(min(n,p)),
work(lwork)
CALL SGGRQF (m, n, p, a, lda, taua, b, ldb, taub, work, lwork,
info)

INTEGER*4 info, lda, ldb, lwork, m, n, p
REAL*8 a(lda, p), b(ldb, p), taua(min(m,p)), taub(min(n,p)),
work(lwork)
CALL DGGRQF (m, n, p, a, lda, taua, b, ldb, taub, work, lwork,
info)

INTEGER*4 info, lda, ldb, lwork, m, n, p
COMPLEX*8 a(lda, p), b(ldb, p), taua(min(m,p)), taub(min(n,p)),
work(lwork)
CALL CGGRQF (m, n, p, a, lda, taua, b, ldb, taub, work, lwork,
info)

INTEGER*4 info, lda, ldb, lwork, m, n, p
COMPLEX*16 a(lda, p), b(ldb, p), taua(min(m,p)), taub(min(n,p)),
work(lwork)
CALL ZGGRQF (m, n, p, a, lda, taua, b, ldb, taub, work, lwork,
info)

LAPACK8, available on C Series and Exemplar architectures:

INTEGER*8 info, lda, ldb, lwork, m, n, p
REAL*8 a(lda, p), b(ldb, p), taua(min(m,p)), taub(min(n,p)),
work(lwork)
CALL SGGRQF (m, n, p, a, lda, taua, b, ldb, taub, work, lwork,
info)

INTEGER*8 info, lda, ldb, lwork, m, n, p
COMPLEX*16 a(lda, p), b(ldb, p), taua(min(m,p)), taub(min(n,p)),
work(lwork)
CALL CGGRQF (m, n, p, a, lda, taua, b, ldb, taub, work, lwork,
info)

Input	m	The number of rows of the matrix A . $m \geq 0$.	
	n	The number of rows of the matrix B . $n \geq 0$.	
	p	The number of columns of the matrices A and B . $p \geq 0$.	
	a	The m -by- p matrix A .	
	lda	The leading dimension of array a in the calling program unit. $lda \geq \max(1,m)$.	
	b	The n -by- p matrix B .	
	ldb	The leading dimension of array b in the calling program unit. $ldb \geq \max(1,n)$.	
	lwork	The length of array work . $lwork \geq \max(1,m,n,p)$. For good performance, lwork must generally be larger. The optimum value of lwork for high performance is returned in work(1) .	
	Working Storage	work	An array used for work space. On successful exit, work(1) contains the optimal work space length lwork for high performance.
	Output	a	On successful exit, the factors R and Q from the factorization $A = RQ$, stored as follows:
If $m < p$, the elements on and above the $(p-m)$ -th superdiagonal of a contain the m -by- m upper triangular matrix R .			
If $m = p$, the elements on and above the principal diagonal of a contain the m -by- m upper triangular matrix R .			
taua		If $m > p$, the elements on and above the $(m-p)$ -th subdiagonal of a contain the m -by- p upper trapezoidal matrix R .	
		The remaining elements of a contain components of the v vectors:	
		If $m \leq p$, row i of a , $i = 1, 2, \dots, m$, contains components 1 to $p-m+i-1$ of v_i .	
b		If $m > p$, row $m-p+i$ of a , $i = 2, 3, \dots, p$, contains components 1 to $i-1$ of v_i .	
		On successful exit, the scalar factors, τ_i , $i = 1, 2, \dots, \min(m,p)$, of the elementary reflection matrices, $H_i = I - \tau_i v_i v_i^*$.	
		On successful exit, the factors T and Z from the factorization $B = ZTQ$, stored as follows:	
		If $n \geq p$, the elements on and above the principal diagonal of b contain the p -by- p upper triangular matrix T .	
b	If $n < p$, the elements on and above the principal diagonal of b contain the n -by- p upper trapezoidal matrix T .		
	The elements below the principal diagonal of the j th column of b , $j = 1, 2, \dots, \min(n-1,p)$, contain components $j+1$ to n of \bar{v}_j .		

taub On successful exit, the scalar factors, $\bar{\tau}_i$, $i = 1, 2, \dots, \min(n,p)$, of the elementary reflection matrices, $\bar{H}_i = I - \bar{\tau}_i \bar{v}_i \bar{v}_i^*$.

info Status response:

info = 0: Successful exit.

info < 0: If **info** = $-k$, the k -th argument had an invalid value.

Notes

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

$m < 0$,
 $n < 0$,
 $p < 0$,
 $lda < \max(1,m)$,
 $ldb < \max(1,n)$, and
 $lwork < \max(1,m,n,p)$.

Generate Unfactored Q from an LQ Factorization**SORGLQ/.../ZUNGLQ**

Purpose Given an orthogonal or unitary matrix represented as a product of elementary reflection matrices

$$Q = H_k H_{k-1} \cdots H_1$$

as computed by `_GELQF`, these subprograms generate (expand) the matrix Q into regular, unfactored form.

Each H_i has the form

$$H_i = I - \tau_i v_i v_i^*$$

where τ_i is a scalar and v_i is an m -dimensional vector whose first $i-1$ components are zero and whose i th component is 1.

The unfactored form of Q overwrites the input factored form of Q .

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```
INTEGER*4 info, k, lda, lwork, m, n
REAL*4    a(lda, n), tau(k), work(lwork)
CALL SORGLQ (m, n, k, a, lda, tau, work, lwork, info)
```

```
INTEGER*4 info, k, lda, lwork, m, n
REAL*8    a(lda, n), tau(k), work(lwork)
CALL DORGLQ (m, n, k, a, lda, tau, work, lwork, info)
```

```
INTEGER*4 info, k, lda, lwork, m, n
COMPLEX*8 a(lda, n), tau(k), work(lwork)
CALL CUNGLQ (m, n, k, a, lda, tau, work, lwork, info)
```

```
INTEGER*4 info, k, lda, lwork, m, n
COMPLEX*16 a(lda, n), tau(k), work(lwork)
CALL ZUNGLQ (m, n, k, a, lda, tau, work, lwork, info)
```

LAPACK8, available on C Series and Exemplar architectures:

```
INTEGER*8 info, k, lda, lwork, m, n
REAL*8    a(lda, n), tau(k), work(lwork)
CALL SORGLQ (m, n, k, a, lda, tau, work, lwork, info)
```

```
INTEGER*8 info, k, lda, lwork, m, n
COMPLEX*16 a(lda, n), tau(k), work(lwork)
CALL CUNGLQ (m, n, k, a, lda, tau, work, lwork, info)
```

Input

- m** The number of rows of the matrix Q . $m \geq 0$.
- n** The number of columns of the matrix Q . $n \geq m$.
- k** The number of elementary reflection matrices whose product defines the matrix Q . $k \geq 0$ and $k \leq m$.
- a** The matrix Q , represented as a product of k elementary reflection matrices, as computed by `_GELQF`.

	lda	The leading dimension of array a in the calling program unit. $\text{lda} \geq \max(1, \mathbf{m})$.
	tau	The scalar factors, τ_i , $i = 1, 2, \dots, \mathbf{k}$, of the elementary reflection matrices, $H_i = I - \tau_i v_i v_i^*$, as computed by <code>_GELQF</code> .
	lwork	The length of array work . $\text{lwork} \geq \max(1, \mathbf{m})$. For good performance, lwork must generally be larger. The optimum value of lwork for high performance is returned in work(1) .
Working Storage	work	An array used for work space. On successful exit, work(1) contains the optimal work space length lwork for high performance.
Output	a	On successful exit, the input is overwritten by the m -by- n matrix Q , in unfactored form.
	info	Status response: info = 0: Successful exit. info < 0: If info = $-k$, the k -th argument had an invalid value.

Notes If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

m < 0,
n < **m**,
k < 0,
k > **m**,
lda < $\max(1, \mathbf{m})$, and
lwork < $\max(1, \mathbf{m})$.

Generate Unfactored Q from a QL Factorization**SORGQL.../ZUNGQL**

Purpose Given an orthogonal or unitary matrix represented as a product of elementary reflection matrices

$$Q = H_k H_{k-1} \cdots H_1$$

as computed by `_GEQLF`, these subprograms generate (expand) the matrix Q into regular, unfactored form.

Each H_i has the form

$$H_i = I - \tau_i v_i v_i^*$$

where τ_i is a scalar and v_i is an m -dimensional vector whose first $i-1$ components are zero and whose i th component is 1.

The unfactored form of Q overwrites the input factored form of Q .

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```
INTEGER*4 info, k, lda, lwork, m, n
REAL*4     a(lda, n), tau(k), work(lwork)
CALL SORGQL(m, n, k, a, lda, tau, work, lwork, info)
```

```
INTEGER*4 info, k, lda, lwork, m, n
REAL*8     a(lda, n), tau(k), work(lwork)
CALL DORGQL(m, n, k, a, lda, tau, work, lwork, info)
```

```
INTEGER*4 info, k, lda, lwork, m, n
COMPLEX*8  a(lda, n), tau(k), work(lwork)
CALL CUNGQL(m, n, k, a, lda, tau, work, lwork, info)
```

```
INTEGER*4 info, k, lda, lwork, m, n
COMPLEX*16 a(lda, n), tau(k), work(lwork)
CALL ZUNGQL(m, n, k, a, lda, tau, work, lwork, info)
```

LAPACK8, available on C Series and Exemplar architectures:

```
INTEGER*8 info, k, lda, lwork, m, n
REAL*8     a(lda, n), tau(k), work(lwork)
CALL SORGQL(m, n, k, a, lda, tau, work, lwork, info)
```

```
INTEGER*8 info, k, lda, lwork, m, n
COMPLEX*16 a(lda, n), tau(k), work(lwork)
CALL CUNGQL(m, n, k, a, lda, tau, work, lwork, info)
```

Input

- m** The number of rows of the matrix Q . $m \geq 0$.
- n** The number of columns of the matrix Q . $n \geq 0$ and $n \leq m$.
- k** The number of elementary reflection matrices whose product defines the matrix Q . $k \geq 0$ and $k \leq n$.
- a** The matrix Q , represented as a product of k elementary reflection matrices, as computed by `_GEQLF`.

	lda	The leading dimension of array a in the calling program unit. $\text{lda} \geq \max(1, \mathbf{m})$.
	tau	The scalar factors, τ_i , $i = 1, 2, \dots, k$, of the elementary reflection matrices, $H_i = I - \tau_i v_i v_i^*$, as computed by _GEQLF.
	lwork	The length of array work . $\text{lwork} \geq \max(1, \mathbf{n})$. For good performance, lwork must generally be larger. The optimum value of lwork for high performance is returned in work(1) .
Working Storage	work	An array used for work space. On successful exit, work(1) contains the optimal work space length lwork for high performance.
Output	a	On successful exit, the input is overwritten by the m -by- n matrix Q , in unfactored form.
	info	Status response: info = 0: Successful exit. info < 0: If info = $-k$, the k -th argument had an invalid value.

Notes If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

m < 0,
n < 0,
n > **m**,
k < 0,
k > **n**,
lda < $\max(1, \mathbf{m})$, and
lwork < $\max(1, \mathbf{n})$.

Generate Unfactored Q from a QR Factorization**SORGQR.../ZUNGQR**

Purpose Given an orthogonal or unitary matrix represented as a product of elementary reflection matrices

$$Q = H_1 H_2 \cdots H_k$$

as computed by _GEQRF, _GGQRF, or _GGRQF, these subprograms generate (expand) the matrix Q into regular, unfactored form.

Each H_i has the form

$$H_i = I - \tau_i v_i v_i^*$$

where τ_i is a scalar and v_i is an m -dimensional vector whose first $i-1$ components are zero and whose i th component is 1.

The unfactored form of Q overwrites the input factored form of Q .

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```
INTEGER*4 info, k, lda, lwork, m, n
REAL*4    a(lda, k), tau(k), work(lwork)
CALL SORGQR (m, n, k, a, lda, tau, work, lwork, info)
```

```
INTEGER*4 info, k, lda, lwork, m, n
REAL*8    a(lda, k), tau(k), work(lwork)
CALL DORGQR (m, n, k, a, lda, tau, work, lwork, info)
```

```
INTEGER*4 info, k, lda, lwork, m, n
COMPLEX*8 a(lda, k), tau(k), work(lwork)
CALL CUNGQR (m, n, k, a, lda, tau, work, lwork, info)
```

```
INTEGER*4 info, k, lda, lwork, m, n
COMPLEX*16 a(lda, k), tau(k), work(lwork)
CALL ZUNGQR (m, n, k, a, lda, tau, work, lwork, info)
```

LAPACK8, available on C Series and Exemplar architectures:

```
INTEGER*8 info, k, lda, lwork, m, n
REAL*8    a(lda, k), tau(k), work(lwork)
CALL SORGQR (m, n, k, a, lda, tau, work, lwork, info)
```

```
INTEGER*8 info, k, lda, lwork, m, n
COMPLEX*16 a(lda, k), tau(k), work(lwork)
CALL CUNGQR (m, n, k, a, lda, tau, work, lwork, info)
```

Input

- m** The number of rows of the matrix Q . $m \geq 0$.
- n** The number of columns of the matrix Q . $n \geq 0$ and $n \leq m$.
- k** The number of elementary reflection matrices whose product defines the matrix Q . $k \geq 0$ and $k \leq n$.
- a** The matrix Q , represented as a product of k elementary reflection matrices, as computed by _GEQRF, _GGQRF, or _GGRQF.

	lda	The leading dimension of array a in the calling program unit. $\text{lda} \geq \max(1, \text{m})$.
	tau	The scalar factors, τ_i , $i = 1, 2, \dots, k$, of the elementary reflection matrices, $H_i = I - \tau_i v_i v_i^*$, as computed by <code>_GEQRF</code> , <code>_GGQRF</code> , or <code>_GGRQF</code> .
	lwork	The length of array work . $\text{lwork} \geq \max(1, \text{n})$. For good performance, lwork must generally be larger. The optimum value of lwork for high performance is returned in work(1) .
Working Storage	work	An array used for work space. On successful exit, work(1) contains the optimal work space length lwork for high performance.
Output	a	On successful exit, the input is overwritten by the m -by- n matrix Q , in unfactored form.
	info	Status response: info = 0: Successful exit. info < 0: If info = $-k$, the k -th argument had an invalid value.

Notes If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

m < 0,
n < 0,
n > **m**,
k < 0,
k > **n**,
lda < $\max(1, \text{m})$, and
lwork < $\max(1, \text{n})$.

Generate Unfactored Q from an RQ Factorization**SORGRQ/.../ZUNGRQ**

Purpose Given an orthogonal or unitary matrix represented as a product of elementary reflection matrices

$$Q = H_1 H_2 \cdots H_k$$

as computed by `_GERQF`, `_GGQRF`, or `_GGRQF`, these subprograms generate (expand) the matrix Q into regular, unfactored form.

Each H_i has the form

$$H_i = I - \tau_i v_i v_i^*$$

where τ_i is a scalar and v_i is an m -dimensional vector whose first $i-1$ components are zero and whose i th component is 1.

The unfactored form of Q overwrites the input factored form of Q .

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```
INTEGER*4 info, k, lda, lwork, m, n
REAL*4    a(lda, n), tau(k), work(lwork)
CALL SORGRQ (m, n, k, a, lda, tau, work, lwork, info)
```

```
INTEGER*4 info, k, lda, lwork, m, n
REAL*8    a(lda, n), tau(k), work(lwork)
CALL DORGRQ (m, n, k, a, lda, tau, work, lwork, info)
```

```
INTEGER*4 info, k, lda, lwork, m, n
COMPLEX*8 a(lda, n), tau(k), work(lwork)
CALL CUNGRQ (m, n, k, a, lda, tau, work, lwork, info)
```

```
INTEGER*4 info, k, lda, lwork, m, n
COMPLEX*16 a(lda, n), tau(k), work(lwork)
CALL ZUNGRQ (m, n, k, a, lda, tau, work, lwork, info)
```

LAPACK8, available on C Series and Exemplar architectures:

```
INTEGER*8 info, k, lda, lwork, m, n
REAL*8    a(lda, n), tau(k), work(lwork)
CALL SORGRQ (m, n, k, a, lda, tau, work, lwork, info)
```

```
INTEGER*8 info, k, lda, lwork, m, n
COMPLEX*16 a(lda, n), tau(k), work(lwork)
CALL CUNGRQ (m, n, k, a, lda, tau, work, lwork, info)
```

Input

- m** The number of rows of the matrix Q . $m \geq 0$.
- n** The number of columns of the matrix Q . $n \geq m$.
- k** The number of elementary reflection matrices whose product defines the matrix Q . $k \geq 0$ and $k \leq m$.
- a** The matrix Q , represented as a product of k elementary reflection matrices, as computed by `_GERQF`, `_GGQRF`, or `_GGRQF`.

	lda	The leading dimension of array a in the calling program unit. $\text{lda} \geq \max(1, \mathbf{m})$.
	tau	The scalar factors, τ_i , $i = 1, 2, \dots, k$, of the elementary reflection matrices, $H_i = I - \tau_i v_i v_i^*$, as computed by <code>_GERQF</code> , <code>_GGQRF</code> , or <code>_GGRQF</code> .
	lwork	The length of array work . $\text{lwork} \geq \max(1, \mathbf{m})$. For good performance, lwork must generally be larger. The optimum value of lwork for high performance is returned in <code>work(1)</code> .
Working Storage	work	An array used for work space. On successful exit, <code>work(1)</code> contains the optimal work space length lwork for high performance.
Output	a	On successful exit, the input is overwritten by the m -by- n matrix Q , in unfactored form.
	info	Status response: info = 0: Successful exit. info < 0: If info = $-k$, the k -th argument had an invalid value.

Notes If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

m < 0,
n < **m**,
k < 0,
k > **m**,
lda < $\max(1, \mathbf{m})$, and
lwork < $\max(1, \mathbf{m})$.

Multiply Matrix by Q from an LQ Factorization**SORMLQ/.../ZUNMLQ**

Purpose Given an m -by- n matrix C , these subprograms compute one of the matrix products QC , Q^*C , CQ , or CQ^* , where Q is an orthogonal or unitary matrix computed by `_GELQF`, and Q^* is the conjugate transpose of Q (ordinary transpose if the matrices are real). The matrix product overwrites the input C matrix.

The matrix Q is represented as a product of elementary reflection matrices

$$Q = H_k H_{k-1} \cdots H_1.$$

Each H_i has the form

$$H_i = I - \tau_i v_i v_i^*$$

where τ_i is a scalar and v_i is an m -dimensional vector if QC or Q^*C is requested, or an n -dimensional vector if CQ or CQ^* is requested. The first $i-1$ components of v_i are zero and the i th component is 1.

Usage

LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```
CHARACTER*1 side, trans
INTEGER*4   info, k, lda, ldc, lwork, m, n
REAL*4     a(lda, *), c(ldc, n), tau(k), work(lwork)
CALL SORMLQ (side, trans, m, n, k, a, lda, tau, c, ldc, work,
            lwork, info)
```

```
CHARACTER*1 side, trans
INTEGER*4   info, k, lda, ldc, lwork, m, n
REAL*8     a(lda, *), c(ldc, n), tau(k), work(lwork)
CALL DORMLQ (side, trans, m, n, k, a, lda, tau, c, ldc, work,
            lwork, info)
```

```
CHARACTER*1 side, trans
INTEGER*4   info, k, lda, ldc, lwork, m, n
COMPLEX*8  a(lda, *), c(ldc, n), tau(k), work(lwork)
CALL CUNMLQ (side, trans, m, n, k, a, lda, tau, c, ldc, work,
            lwork, info)
```

```
CHARACTER*1 side, trans
INTEGER*4   info, k, lda, ldc, lwork, m, n
COMPLEX*16 a(lda, *), c(ldc, n), tau(k), work(lwork)
CALL ZUNMLQ (side, trans, m, n, k, a, lda, tau, c, ldc, work,
            lwork, info)
```

LAPACK8, available on C Series and Exemplar architectures:

```
CHARACTER*1 side, trans
INTEGER*8   info, k, lda, ldc, lwork, m, n
REAL*8     a(lda, *), c(ldc, n), tau(k), work(lwork)
CALL SORMLQ (side, trans, m, n, k, a, lda, tau, c, ldc, work,
            lwork, info)
```

```
CHARACTER*1 side, trans
INTEGER*8   info, k, lda, ldc, lwork, m, n
COMPLEX*16 a(lda, *), c(ldc, n), tau(k), work(lwork)
CALL CUNMLQ (side, trans, m, n, k, a, lda, tau, c, ldc, work,
            lwork, info)
```

Input	side	Specifies whether orthogonal or unitary matrix Q is the left or right matrix operand: side = 'L' or 'l' Q is the left matrix operand. side = 'R' or 'r' Q is the right matrix operand.
	trans	Transposition option for Q : trans = 'N' or 'n' Use matrix Q directly trans = 'T' or 't' Use Q^T , the transpose of Q trans = 'C' or 'c' Use Q^* , the conjugate transpose of Q In SORMLQ and DORMLQ, only 'N' and 'n' and 'T' and 't' are valid. In CUNMLQ and ZUNMLQ, only 'N' and 'n' and 'C' and 'c' are valid.
	m	The number of rows of the matrix C . $m \geq 0$.
	n	The number of columns of the matrix C . $n \geq 0$.
	k	The number of elementary reflection matrices whose product defines the matrix Q . $k \geq 0$. If side = 'L' or 'l', $k \leq m$. If side = 'R' or 'r', $k \leq n$.
	a	The orthogonal or unitary matrix Q , represented as a product of elementary reflection matrices as computed by _GELQF. If side = 'L' or 'l', Q is an m -by- m matrix. If side = 'R' or 'r', Q is an n -by- n matrix. a is modified by the subprogram but is restored to its input value on exit.
	lda	The leading dimension of array a in the calling program unit. $lda \geq \max(1,k)$.
	tau	The scalar factors, τ_i , $i = 1, 2, \dots, k$, of the elementary reflection matrices, $H_i = I - \tau_i v_i v_i^*$, as computed by _GELQF.
	c	The m -by- n input matrix C .
	ldc	The leading dimension of array c in the calling program unit. $ldc \geq \max(1,m)$.
lwork	The length of array work . If side = 'L' or 'l', $lwork \geq \max(1,n)$. If side = 'R' or 'r', $lwork \geq \max(1,m)$. For good performance, lwork must generally be larger. The optimum value of lwork for high performance is returned in work (1).	
Working Storage	work	An array used for work space. On successful exit, work (1) contains the optimal work space length lwork for high performance.
Output	c	On successful exit, the input is overwritten by the requested matrix product.
	info	Status response: info = 0: Successful exit. info < 0: If info = $-k$, the k -th argument had an invalid value.

Notes

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

side \neq 'L' or 'l' or 'R' or 'r',
trans invalid,
m < 0,
n < 0,
k < 0,
k too large,
lda < max(1,k),
ldc < max(1,m), and
lwork too small.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the CALL statement may be improved by coding, for example, the side argument as 'Left' for 'L' or 'Right' for 'R'.

Purpose Given an m -by- n matrix C , these subprograms compute one of the matrix products QC , Q^*C , CQ , or CQ^* , where Q is an orthogonal or unitary matrix computed by `_GEQLF`, and Q^* is the conjugate transpose of Q (ordinary transpose if the matrices are real). The matrix product overwrites the input C matrix.

The matrix Q is represented as a product of elementary reflection matrices

$$Q = H_k H_{k-1} \cdots H_1.$$

Each H_i has the form

$$H_i = I - \tau_i v_i v_i^*$$

where τ_i is a scalar and v_i is an m -dimensional vector whose first $i-1$ components are zero and whose i th component is 1.

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

CHARACTER*1 side, trans
INTEGER*4 info, k, lda, ldc, lwork, m, n
REAL*4 a(lda, k), c(ldc, n), tau(k), work(lwork)
CALL SORMQL (side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork, info)

CHARACTER*1 side, trans
INTEGER*4 info, k, lda, ldc, lwork, m, n
REAL*8 a(lda, k), c(ldc, n), tau(k), work(lwork)
CALL DORMQL (side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork, info)

CHARACTER*1 side, trans
INTEGER*4 info, k, lda, ldc, lwork, m, n
COMPLEX*8 a(lda, k), c(ldc, n), tau(k), work(lwork)
CALL CUNMQL (side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork, info)

CHARACTER*1 side, trans
INTEGER*4 info, k, lda, ldc, lwork, m, n
COMPLEX*16 a(lda, k), c(ldc, n), tau(k), work(lwork)
CALL ZUNMQL (side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork, info)

LAPACK8, available on C Series and Exemplar architectures:

CHARACTER*1 side, trans
INTEGER*8 info, k, lda, ldc, lwork, m, n
REAL*8 a(lda, k), c(ldc, n), tau(k), work(lwork)
CALL SORMQL (side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork, info)

CHARACTER*1 side, trans
INTEGER*8 info, k, lda, ldc, lwork, m, n
COMPLEX*16 a(lda, k), c(ldc, n), tau(k), work(lwork)
CALL CUNMQL (side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork, info)

Input	side	Specifies whether orthogonal or unitary matrix Q is the left or right matrix operand: side = 'L' or 'l' Q is the left matrix operand. side = 'R' or 'r' Q is the right matrix operand.
	trans	Transposition option for Q : trans = 'N' or 'n' Use matrix Q directly trans = 'T' or 't' Use Q^T , the transpose of Q trans = 'C' or 'c' Use Q^* , the conjugate transpose of Q In SORMQL and DORMQL, only 'N' and 'n' and 'T' and 't' are valid. In CUNMQL and ZUNMQL, only 'N' and 'n' and 'C' and 'c' are valid.
	m	The number of rows of the matrix C . $m \geq 0$.
	n	The number of columns of the matrix C . $n \geq 0$.
	k	The number of elementary reflection matrices whose product defines the matrix Q . $k \geq 0$. If side = 'L' or 'l' , $k \leq m$. If side = 'R' or 'r' , $k \leq n$.
	a	The orthogonal or unitary matrix Q , represented as a product of elementary reflection matrices as computed by _GEQLF. If side = 'L' or 'l' , Q is an m -by- m matrix. If side = 'R' or 'r' , Q is an n -by- n matrix. a is modified by the subprogram but is restored to its input value on exit.
	lda	The leading dimension of array a in the calling program unit. If side = 'L' or 'l' , lda $\geq \max(1,m)$. If side = 'R' or 'r' , lda $\geq \max(1,n)$.
	tau	The scalar factors, τ_i , $i = 1, 2, \dots, k$, of the elementary reflection matrices, $H_i = I - \tau_i v_i v_i^*$, as computed by _GEQLF.
	c	The m -by- n input matrix C .
	ldc	The leading dimension of array c in the calling program unit. ldc $\geq \max(1,m)$.
	lwork	The length of array work . If side = 'L' or 'l' , lwork $\geq \max(1,n)$. If side = 'R' or 'r' , lwork $\geq \max(1,m)$. For good performance, lwork must generally be larger. The optimum value of lwork for high performance is returned in work(1) .
	Working Storage	work An array used for work space. On successful exit, work(1) contains the optimal work space length lwork for high performance.
	Output	c
info		Status response: info = 0: Successful exit. info < 0: If info = -k , the k -th argument had an invalid value.

Notes

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

side ≠ 'L' or 'l' or 'R' or 'r',
trans invalid,
m < 0,
n < 0,
k < 0,
k too large,
lda too small,
ldc < max(1,**m**), and
lwork too small.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding, for example, the **side** argument as 'Left' for 'L' or 'Right' for 'R'.

Multiply Matrix by Q from a QR Factorization**SORMQR/.../ZUNMQR**

Purpose Given an m -by- n matrix C , these subprograms compute one of the matrix products QC , Q^*C , CQ , or CQ^* , where Q is an orthogonal or unitary matrix computed by `_GEQRF`, `_GGQRF`, or `_GGRQF`, and Q^* is the conjugate transpose of Q (ordinary transpose if the matrices are real). The matrix product overwrites the input C matrix.

The matrix Q is represented as a product of elementary reflection matrices

$$Q = H_1 H_2 \cdots H_k.$$

Each H_i has the form

$$H_i = I - \tau_i v_i v_i^*$$

where τ_i is a scalar and v_i is an m -dimensional vector whose first $i-1$ components are zero and whose i th component is 1.

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```
CHARACTER*1 side, trans
INTEGER*4   info, k, lda, ldc, lwork, m, n
REAL*4      a(lda, k), c(ldc, n), tau(k), work(lwork)
CALL SORMQR (side, trans, m, n, k, a, lda, tau, c, ldc, work,
            lwork, info)
```

```
CHARACTER*1 side, trans
INTEGER*4   info, k, lda, ldc, lwork, m, n
REAL*8      a(lda, k), c(ldc, n), tau(k), work(lwork)
CALL DORMQR (side, trans, m, n, k, a, lda, tau, c, ldc, work,
            lwork, info)
```

```
CHARACTER*1 side, trans
INTEGER*4   info, k, lda, ldc, lwork, m, n
COMPLEX*8   a(lda, k), c(ldc, n), tau(k), work(lwork)
CALL CUNMQR (side, trans, m, n, k, a, lda, tau, c, ldc, work,
            lwork, info)
```

```
CHARACTER*1 side, trans
INTEGER*4   info, k, lda, ldc, lwork, m, n
COMPLEX*16  a(lda, k), c(ldc, n), tau(k), work(lwork)
CALL ZUNMQR (side, trans, m, n, k, a, lda, tau, c, ldc, work,
            lwork, info)
```

LAPACK8, available on C Series and Exemplar architectures:

```
CHARACTER*1 side, trans
INTEGER*8   info, k, lda, ldc, lwork, m, n
REAL*8      a(lda, k), c(ldc, n), tau(k), work(lwork)
CALL SORMQR (side, trans, m, n, k, a, lda, tau, c, ldc, work,
            lwork, info)
```

```
CHARACTER*1 side, trans
INTEGER*8   info, k, lda, ldc, lwork, m, n
COMPLEX*16  a(lda, k), c(ldc, n), tau(k), work(lwork)
CALL CUNMQR (side, trans, m, n, k, a, lda, tau, c, ldc, work,
            lwork, info)
```

Input	side	Specifies whether orthogonal or unitary matrix Q is the left or right matrix operand: side = 'L' or 'l' Q is the left matrix operand. side = 'R' or 'r' Q is the right matrix operand.
	trans	Transposition option for Q : trans = 'N' or 'n' Use matrix Q directly trans = 'T' or 't' Use Q^T , the transpose of Q trans = 'C' or 'c' Use Q^* , the conjugate transpose of Q In SORMQR and DORMQR, only 'N' and 'n' and 'T' and 't' are valid. In CUNMQR and ZUNMQR, only 'N' and 'n' and 'C' and 'c' are valid.
	m	The number of rows of the matrix C . $m \geq 0$.
	n	The number of columns of the matrix C . $n \geq 0$.
	k	The number of elementary reflection matrices whose product defines the matrix Q . $k \geq 0$. If side = 'L' or 'l', $k \leq m$. If side = 'R' or 'r', $k \leq n$.
	a	The orthogonal or unitary matrix Q , represented as a product of elementary reflection matrices as computed by _GEQRF, _GGQRF, or _GGRQF. If side = 'L' or 'l', Q is an m -by- m matrix. If side = 'R' or 'r', Q is an n -by- n matrix. a is modified by the subprogram but is restored to its input value on exit.
	lda	The leading dimension of array a in the calling program unit. If side = 'L' or 'l', lda $\geq \max(1,m)$. If side = 'R' or 'r', lda $\geq \max(1,n)$.
	tau	The scalar factors, τ_i , $i = 1, 2, \dots, k$, of the elementary reflection matrices, $H_i = I - \tau_i v_i v_i^*$, as computed by _GEQRF, _GGQRF, or _GGRQF.
	c	The m -by- n input matrix C .
	ldc	The leading dimension of array c in the calling program unit. ldc $\geq \max(1,m)$.
	lwork	The length of array work . If side = 'L' or 'l', lwork $\geq \max(1,n)$. If side = 'R' or 'r', lwork $\geq \max(1,m)$. For good performance, lwork must generally be larger. The optimum value of lwork for high performance is returned in work (1).
Working Storage	work	An array used for work space. On successful exit, work (1) contains the optimal work space length lwork for high performance.
Output	c	On successful exit, the input is overwritten by the requested matrix product.
	info	Status response: info = 0: Successful exit. info < 0: If info = $-k$, the k -th argument had an invalid value.

Notes

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

side ≠ 'L' or 'l' or 'R' or 'r',
trans invalid,
m < 0,
n < 0,
k < 0,
k too large,
lda too small,
ldc < max(1,m), and
lwork too small.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding, for example, the **side** argument as 'Left' for 'L' or 'Right' for 'R'.

Purpose Given an m -by- n matrix C , these subprograms compute one of the matrix products QC , Q^*C , CQ , or CQ^* , where Q is an orthogonal or unitary matrix computed by `_GERQF`, `_GGQRF`, or `_GGRQF`, and Q^* is the conjugate transpose of Q (ordinary transpose if the matrices are real). The matrix product overwrites the input C matrix.

The matrix Q is represented as a product of elementary reflection matrices

$$Q = H_1 H_2 \cdots H_k.$$

Each H_i has the form

$$H_i = I - \tau_i v_i v_i^*$$

where τ_i is a scalar and v_i is an m -dimensional vector if QC or Q^*C is requested, or an n -dimensional vector if CQ or CQ^* is requested. The first $i-1$ components of v_i are zero and the i th component is 1.

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

CHARACTER*1 side, trans
INTEGER*4 info, k, lda, ldc, lwork, m, n
REAL*4 a(lda, *), c(ldc, n), tau(k), work(lwork)
CALL SORMRQ (side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork, info)

CHARACTER*1 side, trans
INTEGER*4 info, k, lda, ldc, lwork, m, n
REAL*8 a(lda, *), c(ldc, n), tau(k), work(lwork)
CALL DORMRQ (side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork, info)

CHARACTER*1 side, trans
INTEGER*4 info, k, lda, ldc, lwork, m, n
COMPLEX*8 a(lda, *), c(ldc, n), tau(k), work(lwork)
CALL CUNMRQ (side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork, info)

CHARACTER*1 side, trans
INTEGER*4 info, k, lda, ldc, lwork, m, n
COMPLEX*16 a(lda, *), c(ldc, n), tau(k), work(lwork)
CALL ZUNMRQ (side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork, info)

LAPACK8, available on C Series and Exemplar architectures:

CHARACTER*1 side, trans
INTEGER*8 info, k, lda, ldc, lwork, m, n
REAL*8 a(lda, *), c(ldc, n), tau(k), work(lwork)
CALL SORMRQ (side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork, info)

CHARACTER*1 side, trans
INTEGER*8 info, k, lda, ldc, lwork, m, n
COMPLEX*16 a(lda, *), c(ldc, n), tau(k), work(lwork)
CALL CUNMRQ (side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork, info)

Input	side	Specifies whether orthogonal or unitary matrix Q is the left or right matrix operand: side = 'L' or 'l' Q is the left matrix operand. side = 'R' or 'r' Q is the right matrix operand.
	trans	Transposition option for Q : trans = 'N' or 'n' Use matrix Q directly trans = 'T' or 't' Use Q^T , the transpose of Q trans = 'C' or 'c' Use Q^* , the conjugate transpose of Q In SORMRQ and DORMRQ, only 'N' and 'n' and 'T' and 't' are valid. In CUNMRQ and ZUNMRQ, only 'N' and 'n' and 'C' and 'c' are valid.
	m	The number of rows of the matrix C . $m \geq 0$.
	n	The number of columns of the matrix C . $n \geq 0$.
	k	The number of elementary reflection matrices whose product defines the matrix Q . $k \geq 0$. If side = 'L' or 'l', $k \leq m$. If side = 'R' or 'r', $k \leq n$.
	a	The orthogonal or unitary matrix Q , represented as a product of elementary reflection matrices as computed by _GERQF, _GGQRF, or _GGRQF. If side = 'L' or 'l', Q is an m -by- m matrix. If side = 'R' or 'r', Q is an n -by- n matrix. a is modified by the subprogram but is restored to its input value on exit.
	lda	The leading dimension of array a in the calling program unit. $lda \geq \max(1,k)$.
	tau	The scalar factors, τ_i , $i = 1, 2, \dots, k$, of the elementary reflection matrices, $H_i = I - \tau_i v_i v_i^*$, as computed by _GERQF, _GGQRF, or _GGRQF.
	c	The m -by- n input matrix C .
	ldc	The leading dimension of array c in the calling program unit. $ldc \geq \max(1,m)$.
	lwork	The length of array work . If side = 'L' or 'l', $lwork \geq \max(1,n)$. If side = 'R' or 'r', $lwork \geq \max(1,m)$. For good performance, lwork must generally be larger. The optimum value of lwork for high performance is returned in work (1).
Working Storage	work	An array used for work space. On successful exit, work (1) contains the optimal work space length lwork for high performance.
Output	c	On successful exit, the input is overwritten by the requested matrix product.
	info	Status response: info = 0: Successful exit. info < 0: If info = $-k$, the k -th argument had an invalid value.

Notes

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

side \neq 'L' or 'l' or 'R' or 'r',
trans invalid,
m < 0 ,
n < 0 ,
k < 0 ,
k too large,
lda $< \max(1, k)$,
ldc $< \max(1, m)$, and
lwork too small.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding, for example, the **side** argument as 'Left' for 'L' or 'Right' for 'R'.

RQ Factorization of Upper Trapezoidal Matrix**STZRQF/.../ZTZRQF**

Purpose Given an m -by- n matrix A of the form $A = [U \ X]$, where $m \leq n$, U is an m -by- m upper triangular matrix, and X is an m -by- $(n-m)$ general matrix, these subprograms compute the RQ factorization of A .

The factorization has the form $A = [R \ 0] Q$, where R is an m -by- m upper triangular matrix, 0 is an m -by- $(n-m)$ zero matrix, and Q is an n -by- n orthogonal or unitary matrix. The computed matrix Q is represented as a product of elementary reflection matrices

$$Q = H_m H_{m-1} \cdots H_1$$

Each H_i has the form

$$H_i = I - \tau_i v_i v_i^*$$

where τ_i is a scalar and v_i is an n -dimensional vector all of whose components are zero except the i th, which is 1, and the last $n-m$ components, which are computed.

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```
INTEGER*4 info, lda, m, n
REAL*4    a(lda, n), tau(m)
CALL STZRQF (m, n, a, lda, tau, info)
```

```
INTEGER*4 info, lda, m, n
REAL*8    a(lda, n), tau(m)
CALL DTZRQF (m, n, a, lda, tau, info)
```

```
INTEGER*4 info, lda, m, n
COMPLEX*8 a(lda, n), tau(m)
CALL CTZRQF (m, n, a, lda, tau, info)
```

```
INTEGER*4 info, lda, m, n
COMPLEX*16 a(lda, n), tau(m)
CALL ZTZRQF (m, n, a, lda, tau, info)
```

LAPACK8, available on C Series and Exemplar architectures:

```
INTEGER*8 info, lda, m, n
REAL*8    a(lda, n), tau(m)
CALL STZRQF (m, n, a, lda, tau, info)
```

```
INTEGER*8 info, lda, m, n
COMPLEX*16 a(lda, n), tau(m)
CALL CTZRQF (m, n, a, lda, tau, info)
```

Input

- m** The number of rows of the matrix A . $m \geq 0$.
- n** The number of columns of the matrix A . $n \geq m$.
- a** The m -by- n upper trapezoidal matrix A to be factored. The strictly lower triangular part of **a** is not used as input.
- lda** The leading dimension of array **a** in the calling program unit. $lda \geq \max(1, m)$.

- Output**
- a** On successful exit, the factors R and Q from the factorization $A = [R \ 0] Q$, stored as follows:
- The $(m-m)$ upper triangular part of a contains the upper triangular matrix R .
- Columns $m+1$ to n of row i , $i = 1, 2, \dots, m$, of a contain components $m+1$ to n of v_i .
- tau** On successful exit, the scalar factors, τ_i , $i = 1, 2, \dots, m$, of the elementary reflection matrices, $H_i = I - \tau_i v_i v_i^*$.
- info** Status response:
- info** = 0: Successful exit.
info < 0: If **info** = $-k$, the k -th argument had an invalid value.

Notes If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

$m < 0$,
 $n < 0$, and
 $lda < \max(1, m)$.

Simple Drivers for Ordinary Eigenvalue Problems

Overview

This chapter explains how to use LAPACK simple drivers to compute the Schur form, Schur vectors, eigenvalues, or eigenvalues and eigenvectors of matrices. The operations covered are:

- general dense eigenproblems, $Ax = \lambda x$, for arbitrary A
- symmetric and Hermitian dense eigenproblems, $Ax = \lambda x$
- symmetric and Hermitian banded eigenproblems, $Ax = \lambda x$
- symmetric tridiagonal eigenproblems, $Ax = \lambda x$

Chapter 8 describes the LAPACK expert drivers for ordinary eigenvalue problems. Refer to Chapter 9 for software to compute the eigenvalues or eigenvalues and eigenvectors of generalized eigenproblems.

Refer to Chapter 7 of the *ConvexMLIB User's Guide: VECLIB* for software to compute the eigenvalues or eigenvalues and eigenvectors of a real symmetric, sparse, ordinary or generalized eigenproblem.

Chapter Objectives

After reading this chapter you will know how to use the described subprograms.

What You Need to Know to Use These Subprograms

To use these subprograms, it is helpful to know some basic theory of eigensystems and the Schur Form, especially as they relate to your particular application. A few facts discussed in basic textbooks are given here.

If A is an n -by- n matrix, $\det(\lambda I - A)$ is an n -th degree polynomial in λ , called the *characteristic polynomial*. This equation has n zeros, counting algebraic multiplicity, although some or all of them may be complex (with nonzero imaginary part) even if A is real. If λ is one of these zeros, there exists a nonzero vector x such that $Ax = \lambda x$. λ is called an *eigenvalue* of A , and x is called an *eigenvector* belonging to λ .

Alternatively, the definitions can be made without resorting to the characteristic polynomial, as follows: if λ is a scalar for which there exists a nonzero vector x such that $Ax = \lambda x$, then λ is called an *eigenvalue* of A , and x is called an *eigenvector* belonging to λ .

If A is a real symmetric or complex Hermitian matrix, the eigenvalues of A are real, and there exists a complete orthonormal set of eigenvectors, that is, appropriately chosen and scaled eigenvectors form an orthogonal unit-vector basis for n -dimensional Euclidean space. In the real nonsymmetric case, any non-real eigenvalues occur in complex conjugate pairs. In the nonsymmetric, non-Hermitian case, n linearly independent eigenvectors may or may not exist.

When a complete set of eigenvectors does exist, say, the columns of the n -by- n matrix X , then $AX = X\Lambda$, where Λ is the diagonal matrix of eigenvalues of A . X is invertible since its columns are linearly independent. It follows that $X^{-1}AX = \Lambda$, so X is said to *diagonalize* A . In the real symmetric case, X can be made orthogonal, while if A is a complex Hermitian matrix, X can be made unitary.

Even when a complete set of eigenvectors does not exist, a basic theorem due to Schur states that any matrix is unitarily similar to a triangular matrix; that is, there exists a unitary matrix Q and an upper triangular matrix R such that $Q^*AQ = R$. Such an R is called the Schur Form of A , and the columns of Q are called the Schur vectors. Even if A is a real matrix, Q and R may be complex. However, real matrices Q and R do exist, with Q orthogonal and R block-triangular with 1-by-1 and 2-by-2 diagonal blocks, such that $Q^T A Q = R$. It is common to call the block upper triangular R matrix the Schur Form and the real vectors that are the columns of the Q matrix the Schur vectors.

Supplemental Reading

Golub, G.H. and C.F. Van Loan. *Matrix Computations*, 2nd Edition. Baltimore, MD: The Johns Hopkins University Press. 1989.

Parlett, B.N. *The Symmetric Eigenproblem*. Englewood Cliffs, NJ: Prentice-Hall, Inc. 1980.

Wilkinson, J.H. *The Algebraic Eigenproblem*. New York, NY: Oxford University Press. 1965.

Subprogram Descriptions

Eigenvalues and Schur Form of a General Matrix SGEES, DGEES, CGEES, ZGEEES	7-3
Eigenvalues and Eigenvectors of a General Matrix SGEEV, DGEEV, CGEEV, ZGEEV	7-7
Eigenvalues and Eigenvectors of a Symmetric or Hermitian Band Matrix SSBEV, DSBEV, CHBEV, ZHBEV	7-10
Eigenvalues and Eigenvectors of a Symmetric or Hermitian Band Matrix SSBEVD, DSBEVD, CHBEVD, ZHBEVD	7-13
Eigenvalues and Eigenvectors of a Symmetric or Hermitian Packed Matrix SSPEV, DSPEV, CHPEV, ZHPEV	7-17
Eigenvalues and Eigenvectors of a Symmetric or Hermitian Packed Matrix SSPEVD, DSPEVD, CHPEVD, ZHPEVD	7-20
Eigenvalues and Eigenvectors of a Real Symmetric Tridiagonal Matrix SSTEVD, DSSTEVD	7-24
Eigenvalues and Eigenvectors of a Real Symmetric Tridiagonal Matrix SSTEVD, DSSTEVD	7-26
Eigenvalues and Eigenvectors of a Symmetric or Hermitian Matrix SSYEV, DSYEV, CHEEV, ZHEEV	7-29
Eigenvalues and Eigenvectors of a Symmetric or Hermitian Matrix SSYEVD, DSYEVD, CHEEVD, ZHEEVD	7-32

Schur Form of a General Matrix**SGEES/DGEES/CGEES/ZGEES**

Purpose These subprograms compute the eigenvalues and the Schur Form of an n -by- n general matrix A , and optionally compute the Schur vectors of A and order the eigenvalues on the diagonal of the Schur Form so that selected eigenvalues are at the upper left.

Given such a matrix A , the eigenvalues of A are scalars λ_i , $i = 1, 2, \dots, n$, for which there exist corresponding nonzero vectors, z_i , $i = 1, 2, \dots, n$, such that

$$Az_i = \lambda_i z_i.$$

A real matrix is in Schur Form if it is block upper triangular with 1-by-1 and 2-by-2 diagonal blocks. 1-by-1 blocks correspond to real eigenvalues, and 2-by-2 blocks correspond to complex conjugate eigenpairs. 2-by-2 blocks are standardized in the form

$$\begin{bmatrix} a & b \\ c & a \end{bmatrix}$$

where $bc < 0$. The eigenvalues of such a block are $a \pm i\sqrt{|bc|}$.

A complex matrix is in Schur Form if it upper triangular.

If T is the Schur Form of A , then the columns of unitary matrix Q such that

$$A = QTQ^*$$

are known as the Schur vectors of A .

Usage

LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```

CHARACTER*1 jobvs, sort
INTEGER*4 info, lda, ldvs, lwork, n, sdim
LOGICAL*4 bwork(n)
REAL*4 a(lda, n), vs(ldvs, n), wi(n), work(lwork), wr(n)
LOGICAL*4 select
EXTERNAL select
CALL SGEES (jobvs, sort, select, n, a, lda, sdim, wr, wi, vs, ldvs,
work, lwork, bwork, info)

```

```

CHARACTER*1 jobvs, sort
INTEGER*4 info, lda, ldvs, lwork, n, sdim
LOGICAL*4 bwork(n)
REAL*8 a(lda, n), vs(ldvs, n), wi(n), work(lwork), wr(n)
LOGICAL*4 select
EXTERNAL select
CALL DGEES (jobvs, sort, select, n, a, lda, sdim, wr, wi, vs, ldvs,
work, lwork, bwork, info)

```

```

CHARACTER*1 jobvs, sort
INTEGER*4 info, lda, ldvs, lwork, n, sdim
LOGICAL*4 bwork(n)
REAL*4 rwork(n)
COMPLEX*8 a(lda, n), vs(ldvs, n), w(n), work(lwork)
LOGICAL*4 select
EXTERNAL select
CALL CGEES (jobvs, sort, select, n, a, lda, sdim, w, vs, ldvs,
work, lwork, rwork, bwork, info)

```

```

CHARACTER*1 jobvs, sort
INTEGER*4   info, lda, ldvs, lwork, n, sdim
LOGICAL*4   bwork(n)
REAL*8     rwork(n)
COMPLEX*16  a(lda, n), vs(ldvs, n), w(n), work(lwork)
LOGICAL*4   select
EXTERNAL    select
CALL ZGEES (jobvs, sort, select, n, a, lda, sdim, w, vs, ldvs,
           work, lwork, rwork, bwork, info)

```

LAPACK8, available on C Series and Exemplar architectures:

```

CHARACTER*1 jobvs, sort
INTEGER*8   info, lda, ldvs, lwork, n, sdim
LOGICAL*8   bwork(n)
REAL*8     a(lda, n), vs(ldvs, n), wi(n), work(lwork), wr(n)
LOGICAL*8   select
EXTERNAL    select
CALL SGEES (jobvs, sort, select, n, a, lda, sdim, wr, wi, vs, ldvs,
           work, lwork, bwork, info)

```

```

CHARACTER*1 jobvs, sort
INTEGER*8   info, lda, ldvs, lwork, n, sdim
LOGICAL*8   bwork(n)
REAL*8     rwork(n)
COMPLEX*16  a(lda, n), vs(ldvs, n), w(n), work(lwork)
LOGICAL*8   select
EXTERNAL    select
CALL CGEES (jobvs, sort, select, n, a, lda, sdim, w, vs, ldvs,
           work, lwork, rwork, bwork, info)

```

Input	jobvs	Specifies whether the Schur vectors are to be computed, as follows: jobvs = 'N' or 'n': Schur vectors are not computed. jobvs = 'V' or 'v': Schur vectors are computed.
	sort	Specifies whether the eigenvalues on the diagonal of the Schur Form are to be reordered, as follows: sort = 'N' or 'n': The eigenvalues are not specially ordered. sort = 'S' or 's': The eigenvalues are ordered so that the eigenvalues λ_j for which the logical function select(wr(j),wi(j)) (in SGEES and DGEES) or select(w(j)) (in CGEES and ZGEES) is true will appear at the top left corner of the Schur Form. (In SGEES and DGEES, if an eigenvalue is complex and either select(wr(j),wi(j)) or select(wr(j),-wi(j)) is true, both eigenvalues are selected.)

	select	The name of a user-defined function subprogram used if sort = 'S' or 's' to select eigenvalues to reorder to the upper left of the Schur Form. For SGEES and DGEES, select requires two arguments of the same type as wr and wi , while for CGEES and ZGEES, select requires one argument of the same type as w . The eigenvalue λ_j is selected if select(wr(j),wi(j)) or select(w(j)) is true. Note that a selected complex eigenvalue may no longer satisfy select(λ_j) = .TRUE. after ordering, since ordering may perturb the value of complex eigenvalues, and especially ill-conditioned ones; in this case info may be set to a positive value (see info below). select must be declared EXTERNAL in the calling subroutine. select is not referenced if sort = 'N' or 'n'.
	n	The order of the matrix <i>A</i> . $n \geq 0$.
	a	The <i>n</i> -by- <i>n</i> matrix <i>A</i> .
	lda	The leading dimension of array a in the calling program unit. $lda \geq \max(1,n)$.
	ldvs	The leading dimension of array vs in the calling program unit. $ldvs \geq 1$, and if jobvs = 'V' or 'v', then $ldvs \geq n$.
	lwork	The length of array work . $lwork \geq \max(1,3n)$. For good performance, lwork must generally be larger. The optimum value of lwork for high performance is returned in work(1) .
Working Storage	work	An array used for work space. On successful exit, work(1) contains the optimal work space length lwork for high performance.
	rwork	An array used for work space.
	bwork	An array used for work space. Not referenced if sort = 'N' or 'n'.
Output	a	On successful exit, the Schur Form of <i>A</i> overwrites the input. If sort = 'S' or 's', the output Schur Form is $\begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{bmatrix}$ where A_{11} is sdim -by- sdim , A_{12} is sdim -by- $(n-\text{sdim})$, and A_{22} is $(n-\text{sdim})$ -by- $(n-\text{sdim})$, and where the eigenvalues λ_j , $j = 1, 2, \dots, \text{sdim}$, of A_{11} satisfy select(λ_j) = .TRUE. and the eigenvalues λ_j , $j = \text{sdim}+1, \text{sdim}+2, \dots, n$, of A_{22} satisfy select(λ_j) = .FALSE.
	sdim	If sort = 'N' or 'n', sdim = 0. If sort = 'S' or 's', sdim is the number of eigenvalues (after reordering) for which select is true. In SGEES and DGEES, complex conjugate pairs for which select is true for either eigenvalue count as 2.

- wr, wi** On successful exit from SGEES and DGEES, $wr(j)$ and $wi(j)$ contain the real and imaginary parts, respectively, of the computed eigenvalue λ_j , $j = 1, 2, \dots, n$. The eigenvalues are in the same order in which they appear as the eigenvalues of the diagonal 1-by-1 and 2-by-2 blocks of the Schur Form. Complex conjugate pairs of eigenvalues appear consecutively with the eigenvalue having the positive imaginary part first.
- w** On successful exit from CGEES and ZGEES, $w(j)$ contains the computed eigenvalue λ_j , $j = 1, 2, \dots, n$. The eigenvalues are in the same order in which they appear on the diagonal of the Schur Form.
- vs** On successful exit, if **jobvs** = 'V' or 'v', the Schur vectors of A . Not referenced if **jobvs** = 'N' or 'n'.
- info** Status response:
- info** = 0: Successful exit.
info < 0: If **info** = $-k$, the k -th argument had an invalid value.
info > 0: The algorithm terminated before completing the requested computation.

Notes

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

jobvs \neq 'N' or 'n' or 'V' or 'v'
sort \neq 'N' or 'n' or 'S' or 's'
n < 0,
lda < max(1,n),
ldvs too small, and
lwork too small.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding, for example, the **jobvs** argument as 'NoVectors' for 'N' or 'Vectors' for 'V'.

General Matrix Eigenproblem**SGEEV/DGEEV/CGEEV/ZGEEV**

Purpose These subprograms compute all eigenvalues and, optionally, all left and/or right eigenvectors of an n -by- n nonsymmetric matrix A .

Given such a matrix A , the eigenvalues of A are scalars λ_i , $i = 1, 2, \dots, n$, for which there exist corresponding nonzero vectors, z_i , $i = 1, 2, \dots, n$, such that

$$Az_i = \lambda_i z_i.$$

Optionally, the z_i , which are called right eigenvectors, also may be computed. In addition, these subprograms also can compute the left eigenvectors, which satisfy

$$y_i^* A = \lambda_i y_i^*.$$

Usage

LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```

CHARACTER*1 jobvl, jobvr
INTEGER*4 info, lda, ldvl, ldvr, lwork, n
REAL*4 a(lda, n), vl(ldvl, n), vr(ldvr, n), wi(n),
work(lwork), wr(n)
CALL SGEEV (jobvl, jobvr, n, a, lda, wr, wi, vl, ldvl, vr, ldvr,
work, lwork, info)

```

```

CHARACTER*1 jobvl, jobvr
INTEGER*4 info, lda, ldvl, ldvr, lwork, n
REAL*8 a(lda, n), vl(ldvl, n), vr(ldvr, n), wi(n),
work(lwork), wr(n)
CALL DGEEV (jobvl, jobvr, n, a, lda, wr, wi, vl, ldvl, vr, ldvr,
work, lwork, info)

```

```

CHARACTER*1 jobvl, jobvr
INTEGER*4 info, lda, ldvl, ldvr, lwork, n
REAL*4 rwork(2*n)
COMPLEX*8 a(lda, n), vl(ldvl, n), vr(ldvr, n), w(n), work(lwork)
CALL CGEEV (jobvl, jobvr, n, a, lda, w, vl, ldvl, vr, ldvr,
work, lwork, rwork, info)

```

```

CHARACTER*1 jobvl, jobvr
INTEGER*4 info, lda, ldvl, ldvr, lwork, n
REAL*8 rwork(2*n)
COMPLEX*16 a(lda, n), vl(ldvl, n), vr(ldvr, n), w(n), work(lwork)
CALL ZGEEV (jobvl, jobvr, n, a, lda, w, vl, ldvl, vr, ldvr,
work, lwork, rwork, info)

```

LAPACK8, available on C Series and Exemplar architectures:

```

CHARACTER*1 jobvl, jobvr
INTEGER*8 info, lda, ldvl, ldvr, lwork, n
REAL*8 a(lda, n), vl(ldvl, n), vr(ldvr, n), wi(n),
work(lwork), wr(n)
CALL SGEEV (jobvl, jobvr, n, a, lda, wr, wi, vl, ldvl, vr, ldvr,
work, lwork, info)

```

```

CHARACTER*1 jobvl, jobvr
INTEGER*8 info, lda, ldvl, ldvr, lwork, n
REAL*8 rwork(2*n)
COMPLEX*16 a(lda, n), vl(ldvl, n), vr(ldvr, n), w(n), work(lwork)
CALL CGEEV (jobvl, jobvr, n, a, lda, w, vl, ldvl, vr, ldvr,
work, lwork, rwork, info)

```

Input	jobvl	Specifies whether the left eigenvectors of A are to be computed, as follows: jobvl = 'N' or 'n': Do not compute the left eigenvectors. jobvl = 'V' or 'v': Compute the left eigenvectors.
	jobvr	Specifies whether the right eigenvectors of A are to be computed, as follows: jobvr = 'N' or 'n': Do not compute the right eigenvectors. jobvr = 'V' or 'v': Compute the right eigenvectors.
	n	The order of the matrix A . $n \geq 0$.
	a	The n -by- n matrix whose eigenvalues and eigenvectors are desired.
	lda	The leading dimension of array a in the calling program unit. $lda \geq \max(1, n)$.
	ldvl	The leading dimension of array vl in the calling program unit. $ldvl \geq 1$, and if jobvl = 'V' or 'v', then $ldvl \geq n$.
	ldvr	The leading dimension of array vr in the calling program unit. $ldvr \geq 1$, and if jobvr = 'V' or 'v', then $ldvr \geq n$.
	lwork	The length of array work . $lwork \geq \max(1, 3n)$ if jobvl = 'N' or 'n' and jobvr = 'N' or 'n', and $lwork \geq \max(1, 4n)$ otherwise. For good performance, lwork must generally be larger. The optimum value of lwork for high performance is returned in work(1) .
Working Storage	work, rwork	Arrays used for work space. On successful exit, work(1) contains the optimal work space length lwork for high performance.
Output	a	Destroyed.
	wr, wi	On successful exit from SGEEV and DGEEV, wr(j) and wi(j) contain the real and imaginary parts, respectively, of the computed eigenvalue λ_j , $j = 1, 2, \dots, n$. The eigenvalues are not in any particular order, except that complex conjugate pairs of eigenvalues appear consecutively with the eigenvalue having the positive imaginary part first.
	w	On successful exit from CGEEV and ZGEEV, w(j) contains the computed eigenvalue λ_j , $j = 1, 2, \dots, n$. The eigenvalues are not in any particular order.
	vl	On successful exit, if jobvl = 'V' or 'v', the left eigenvectors, y_j , $j = 1, 2, \dots, n$, stored one after another in the columns of vl , in the same order as the eigenvalues. The eigenvectors are normalized to have Euclidean norm equal to 1, and to have the largest component real.

In SGEEV and DGEEV, a left eigenvector corresponding to a real eigenvalue is real and takes up one column. An eigenvector pair corresponding to a complex conjugate pair of eigenvalues is complex and takes up two columns: the first column holds the real part and the second column holds the imaginary part of the eigenvector corresponding to the eigenvalue with positive imaginary part.

Therefore, if $wi(j) > 0$, the left eigenvector corresponding to $wr(j) + i wi(j)$ is $vl(j) + i vl(j+1)$, and if $wi(j) < 0$, the left eigenvector corresponding to $wr(j) + i wi(j)$ is $vl(j-1) - i vl(j)$, where $i = \sqrt{-1}$.

In CGEEV and ZGEEV, each left eigenvector takes up one column.

Not referenced if `jobvl = 'N'` or `'n'`.

vr On successful exit, if `jobvr = 'V'` or `'v'`, the right eigenvectors, z_j , $j = 1, 2, \dots, n$, stored one after another in the columns of `vr`, in the same order as their eigenvalues. The eigenvectors are normalized to have Euclidean norm equal to 1, and to have the largest component real.

In SGEEV and DGEEV, a right eigenvector corresponding to a real eigenvalue is real and takes up one column. An eigenvector pair corresponding to a complex conjugate pair of eigenvalues is complex and takes up two columns: the first column holds the real part and the second column holds the imaginary part of the eigenvector corresponding to the eigenvalue with positive imaginary part.

Therefore, if $w_i(j) > 0$, the right eigenvector corresponding to $wr(j) + i w_i(j)$ is $vr(j) + i vr(j+1)$, and if $w_i(j) < 0$, the right eigenvector corresponding to $wr(j) + i w_i(j)$ is $vr(j-1) - i vr(j)$, where $i = \sqrt{-1}$.

In CGEEV and ZGEEV, each right eigenvector takes up one column.

Not referenced if `jobvr = 'N'` or `'n'`.

info Status response:

info = 0: Successful exit.

info < 0: If `info = -k`, the k -th argument had an invalid value.

info > 0: The algorithm terminated before completing the requested computation.

Notes

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

`jobvl` \neq `'N'` or `'n'` or `'V'` or `'v'`,
`jobvr` \neq `'N'` or `'n'` or `'V'` or `'v'`,
`n` < 0,
`lda` < max(1,n),
`ldvl` too small,
`ldvr` too small, and
`lwork` too small.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding the `jobvl` and `jobvr` arguments as `'NoVectors'` for `'N'` or `'Vectors'` for `'V'`.

Purpose These subprograms compute all eigenvalues and, optionally, all eigenvectors of a real symmetric or complex Hermitian band matrix.

A matrix is symmetric if $A = A^T$, its transpose; a matrix is Hermitian if $A = A^*$, its conjugate transpose.

A matrix is banded if its nonzero elements all lie fairly near the principal diagonal. Specifically, $a_{ij} = 0$ if $|i-j| > kd$ for some integer kd . The smallest such kd for a given matrix is called the half bandwidth, and $2kd+1$ is called the total bandwidth.

Given such a matrix A , the eigenvalues of A are scalars λ_i , $i = 1, 2, \dots, n$, for which there exist corresponding nonzero vectors, z_i , $i = 1, 2, \dots, n$, such that

$$Az_i = \lambda_i z_i.$$

Optionally, the eigenvectors z_i also may be computed.

Matrix Storage Because it is not necessary to store or operate on the zeros outside the band of A , and since either triangle of A may be obtained from the other, you need only provide the band within one triangle of A . Compared to storing the entire matrix, this can save memory in two ways: only the elements within the band are stored, and of them, only the upper or the lower triangle.

The following examples illustrate the storage of real symmetric band matrices. Consider the following matrix A of order $n = 7$ and half bandwidth $kd = 2$:

11	12	13	0	0	0	0
12	22	23	24	0	0	0
13	23	33	34	35	0	0
0	24	34	44	45	46	0
0	0	35	45	55	56	57
0	0	0	46	56	66	67
0	0	0	0	57	67	77

Upper triangular storage. The upper triangle of the matrix A is stored in an array **ab** with at least $kd+1 = 3$ rows and 7 columns as follows:

*	*	13	24	35	46	57
*	12	23	34	45	56	67
11	22	33	44	55	66	77

The asterisks represent elements in the kd -by- kd triangle at the upper left corner of **ab** that are not referenced. Thus, if a_{ij} is an element within the band of the upper triangle of A , it is stored in **ab**($kd+1+i-j, j$). Therefore, the columns of the upper triangle of A are stored in the columns of **ab**, and the diagonals of the upper triangle of A are stored in the rows of **ab**.

Lower triangular storage. The lower triangle of A is stored in the array **ab** as follows:

11	22	33	44	55	66	77
12	23	34	45	56	67	*
13	24	35	46	57	*	*

The asterisks represent elements in the kd -by- kd triangle at the lower right corner of \mathbf{ab} that are not referenced. Thus, if a_{ij} is an element within the band of the lower triangle of A , it is stored in $\mathbf{ab}(1+i-j, j)$. Therefore, the columns of the lower triangle of A are stored in the columns of \mathbf{ab} , and the diagonals of the lower triangle of A are stored in the rows of \mathbf{ab} .

Usage

LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```

CHARACTER*1 jobz, uplo
INTEGER*4   info, kd, ldab, ldz, n
REAL*4     ab(ldab, n), w(n), work(max(1,3*n-2)), z(ldz, n)
CALL SSBEV (jobz, uplo, n, kd, ab, ldab, w, z, ldz, work, info)

```

```

CHARACTER*1 jobz, uplo
INTEGER*4   info, kd, ldab, ldz, n
REAL*8     ab(ldab, n), w(n), work(max(1,3*n-2)), z(ldz, n)
CALL DSBEV (jobz, uplo, n, kd, ab, ldab, w, z, ldz, work, info)

```

```

CHARACTER*1 jobz, uplo
INTEGER*4   info, kd, ldab, ldz, n
REAL*4     rwork(max(1,3*n-2)), w(n)
COMPLEX*8  ab(ldab, n), work(n), z(ldz, n)
CALL CHBEV (jobz, uplo, n, kd, ab, ldab, w, z, ldz, work, rwork,
             info)

```

```

CHARACTER*1 jobz, uplo
INTEGER*4   info, kd, ldab, ldz, n
REAL*8     rwork(max(1,3*n-2)), w(n)
COMPLEX*16 ab(ldab, n), work(n), z(ldz, n)
CALL ZHBEV (jobz, uplo, n, kd, ab, ldab, w, z, ldz, work, rwork,
             info)

```

LAPACK8, available on C Series and Exemplar architectures:

```

CHARACTER*1 jobz, uplo
INTEGER*8   info, kd, ldab, ldz, n
REAL*8     ab(ldab, n), w(n), work(max(1,3*n-2)), z(ldz, n)
CALL SSBEV (jobz, uplo, n, kd, ab, ldab, w, z, ldz, work, info)

```

```

CHARACTER*1 jobz, uplo
INTEGER*8   info, kd, ldab, ldz, n
REAL*8     rwork(max(1,3*n-2)), w(n)
COMPLEX*16 ab(ldab, n), work(n), z(ldz, n)
CALL CHBEV (jobz, uplo, n, kd, ab, ldab, w, z, ldz, work, rwork,
             info)

```

Input

jobz Specifies whether the eigenvectors are to be computed, as follows:

```

jobz = 'N' or 'n': Compute eigenvalues only.
jobz = 'V' or 'v': Compute eigenvectors as well.

```

uplo Specifies whether the upper or lower triangular part of the symmetric or Hermitian matrix A is stored, as follows:

```

uplo = 'U' or 'u': The upper triangular part of  $A$  is stored.
uplo = 'L' or 'l': The lower triangular part of  $A$  is stored.

```

	n	The order of the matrix A . $n \geq 0$.
	kd	The number of super-diagonals of the matrix A if uplo = 'U' or 'u', or the number of subdiagonals if uplo = 'L' or 'l'. $kd \geq 0$.
	ab	The upper or lower triangle of the symmetric or Hermitian band matrix, stored in the first $kd+1$ rows of array ab . The j -th column of A is stored in the j -th column of array ab as follows: If uplo = 'U' or 'u', $ab(kd+1+i-j, j) = A(i, j)$ for $\max(1, j-kd) \leq i \leq j$; If uplo = 'L' or 'l', $ab(1+i-j, j) = A(i, j)$ for $j \leq i \leq \min(n, j+kd)$.
	ldab	The leading dimension of array ab in the calling program unit. $ldab \geq kd+1$.
	ldz	The leading dimension of array z in the calling program unit. $ldz \geq 1$, and if jobz = 'V' or 'v', then $ldz \geq n$.
Working Storage	work, rwork	Arrays used for work space.
Output	ab	Destroyed.
	w	On successful exit, the eigenvalues in ascending order. If an error exit is made, the eigenvalues are correct for indices 1, 2, ..., info -1, but they are unordered and may not be the smallest eigenvalues of the matrix.
	z	On successful exit, if jobz = 'V' or 'v', the orthonormal eigenvectors of the matrix. If an error exit is made, z contains the eigenvectors associated with the stored eigenvalues. Not referenced if jobz = 'N' or 'n'.
	info	Status response: info = 0: Successful exit. info < 0: If info = $-k$, the k -th argument had an invalid value. info > 0: If info = k , the algorithm terminated before finding the k -th eigenvalue.

Notes If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

jobz ≠ 'N' or 'n' or 'V' or 'v',
uplo ≠ 'L' or 'l' or 'U' or 'u',
n < 0,
kd < 0,
ldab < **kd**+1, and
ldz too small.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding, for example, the **jobz** argument as 'NoVectors' for 'N' or 'Vectors' for 'V'.

Symmetric or Hermitian Band Matrix**SSBEVD/DSBEVD/.../ZHBEVD**

Purpose These subprograms compute all eigenvalues and, optionally, all eigenvectors of a real symmetric or complex Hermitian band matrix. If the eigenvectors are desired, the subroutines use a divide-and-conquer algorithm.

A matrix is symmetric if $A = A^T$, its transpose; a matrix is Hermitian if $A = A^*$, its conjugate transpose.

A matrix is banded if its nonzero elements all lie fairly near the principal diagonal. Specifically, $a_{ij} = 0$ if $|i-j| > kd$ for some integer kd . The smallest such kd for a given matrix is called the half bandwidth, and $2kd+1$ is called the total bandwidth.

Given such a matrix A , the eigenvalues of A are scalars λ_i , $i = 1, 2, \dots, n$, for which there exist corresponding nonzero vectors, z_i , $i = 1, 2, \dots, n$, such that

$$Az_i = \lambda_i z_i.$$

Optionally, the eigenvectors z_i also may be computed.

Matrix Storage Because it is not necessary to store or operate on the zeros outside the band of A , and since either triangle of A may be obtained from the other, you need only provide the band within one triangle of A . Compared to storing the entire matrix, this can save memory in two ways: only the elements within the band are stored, and of them, only the upper or the lower triangle.

The following examples illustrate the storage of real symmetric band matrices. Consider the following matrix A of order $n = 7$ and half bandwidth $kd = 2$:

11	12	13	0	0	0	0
12	22	23	24	0	0	0
13	23	33	34	35	0	0
0	24	34	44	45	46	0
0	0	35	45	55	56	57
0	0	0	46	56	66	67
0	0	0	0	57	67	77

Upper triangular storage. The upper triangle of the matrix A is stored in an array **ab** with at least $kd+1 = 3$ rows and 7 columns as follows:

*	*	13	24	35	46	57
*	12	23	34	45	56	67
11	22	33	44	55	66	77

The asterisks represent elements in the kd -by- kd triangle at the upper left corner of **ab** that are not referenced. Thus, if a_{ij} is an element within the band of the upper triangle of A , it is stored in **ab**($kd+1+i-j, j$). Therefore, the columns of the upper triangle of A are stored in the columns of **ab**, and the diagonals of the upper triangle of A are stored in the rows of **ab**.

Lower triangular storage. The lower triangle of A is stored in the array **ab** as follows:

11	22	33	44	55	66	77
12	23	34	45	56	67	*
13	24	35	46	57	*	*

The asterisks represent elements in the kd -by- kd triangle at the lower right corner of \mathbf{ab} that are not referenced. Thus, if a_{ij} is an element within the band of the lower triangle of A , it is stored in $\mathbf{ab}(1+i-j,j)$. Therefore, the columns of the lower triangle of A are stored in the columns of \mathbf{ab} , and the diagonals of the lower triangle of A are stored in the rows of \mathbf{ab} .

Usage

LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```

CHARACTER*1  jobz, uplo
INTEGER*4    info, kd, ldab, ldz, liwork, lwork, n
INTEGER*4    iwork(liwork)
REAL*4      ab(ldab, n), w(n), work(lwork), z(ldz, n)
CALL SSBEVD (jobz, uplo, n, kd, ab, ldab, w, z, ldz, work, lwork,
              iwork, liwork, info)

```

```

CHARACTER*1  jobz, uplo
INTEGER*4    info, kd, ldab, ldz, liwork, lwork, n
INTEGER*4    iwork(liwork)
REAL*8      ab(ldab, n), w(n), work(lwork), z(ldz, n)
CALL DSBEVD (jobz, uplo, n, kd, ab, ldab, w, z, ldz, work, lwork,
              iwork, liwork, info)

```

```

CHARACTER*1  jobz, uplo
INTEGER*4    info, kd, ldab, ldz, liwork, lrwork, lwork, n
INTEGER*4    iwork(liwork)
REAL*4      rwork(lrwork), w(n)
COMPLEX*8   ab(ldab, n), work(n), z(ldz, n)
CALL CHBEVD (jobz, uplo, n, kd, ab, ldab, w, z, ldz, work, lwork,
              rwork, lrwork, iwork, liwork, info)

```

```

CHARACTER*1  jobz, uplo
INTEGER*4    info, kd, ldab, ldz, liwork, lrwork, lwork, n
INTEGER*4    iwork(liwork)
REAL*8      rwork(lrwork), w(n)
COMPLEX*16  ab(ldab, n), work(n), z(ldz, n)
CALL ZHBEVD (jobz, uplo, n, kd, ab, ldab, w, z, ldz, work, lwork,
              rwork, lrwork, iwork, liwork, info)

```

LAPACK8, available on C Series and Exemplar architectures:

```

CHARACTER*1  jobz, uplo
INTEGER*8    info, kd, ldab, ldz, liwork, lwork, n
INTEGER*8    iwork(liwork)
REAL*8      ab(ldab, n), w(n), work(lwork), z(ldz, n)
CALL SSBEVD (jobz, uplo, n, kd, ab, ldab, w, z, ldz, work, lwork,
              iwork, liwork, info)

```

```

CHARACTER*1  jobz, uplo
INTEGER*8    info, kd, ldab, ldz, liwork, lrwork, lwork, n
INTEGER*8    iwork(liwork)
REAL*8      rwork(lrwork), w(n)
COMPLEX*16  ab(ldab, n), work(n), z(ldz, n)
CALL CHBEVD (jobz, uplo, n, kd, ab, ldab, w, z, ldz, work, lwork,
              rwork, lrwork, iwork, liwork, info)

```

Continued

Input	jobz	Specifies whether the eigenvectors are to be computed, as follows: jobz = 'N' or 'n': Compute eigenvalues only. jobz = 'V' or 'v': Compute eigenvectors as well.
	uplo	Specifies whether the upper or lower triangular part of the symmetric or Hermitian matrix <i>A</i> is stored, as follows: uplo = 'U' or 'u': The upper triangular part of <i>A</i> is stored. uplo = 'L' or 'l': The lower triangular part of <i>A</i> is stored.
	n	The order of the matrix <i>A</i> . $n \geq 0$.
	kd	The number of super-diagonals of the matrix <i>A</i> if uplo = 'U' or 'u', or the number of subdiagonals if uplo = 'L' or 'l'. $kd \geq 0$.
	ab	The upper or lower triangle of the symmetric or Hermitian band matrix, stored in the first $kd+1$ rows of array ab . The <i>j</i> -th column of <i>A</i> is stored in the <i>j</i> -th column of array ab as follows: If uplo = 'U' or 'u', $ab(kd+1+i-j, j) = A(i, j)$ for $\max(1, j-kd) \leq i \leq j$; If uplo = 'L' or 'l', $ab(1+i-j, j) = A(i, j)$ for $j \leq i \leq \min(n, j+kd)$.
	ldab	The leading dimension of array ab in the calling program unit. $ldab \geq kd+1$.
	ldz	The leading dimension of array z in the calling program unit. $ldz \geq 1$, and if jobz = 'V' or 'v', then $ldz \geq n$.
	lwork	The length of array work . For SSBEVD and DSBEVD, $lwork \geq \max(1, 2*n)$ if jobz = 'N' or 'n', and $lwork \geq 3*n**2+4*n+2*n*lg(n)+1$ if jobz = 'V' or 'v', where $lg(n)$ is the smallest integer $k \geq 0$ such that $2^k \geq n$. For CHBEVD and ZHBEVD, $lwork \geq \max(1, n)$ if jobz = 'N' or 'n', and $lwork \geq \max(1, 2*n**2)$ if jobz = 'V' or 'v'. For good performance, lwork must generally be larger. The optimum value of lwork for high performance is returned in work(1) .
	lrwork	The length of array rwork . $lrwork \geq \max(1, n)$ if jobz = 'N' or 'n', and $lrwork \geq 3*n**2+4*n+2*n*lg(n)+1$ if jobz = 'V' or 'v', where $lg(n)$ is the smallest integer $k \geq 0$ such that $2^k \geq n$. For good performance, lrwork must generally be larger. The optimum value of lrwork for high performance is returned in rwork(1) .
	liwork	The length of array iwork . $liwork \geq 1$ if jobz = 'N' or 'n', and $liwork \geq 5*n+2$ if jobz = 'V' or 'v'. For good performance, liwork must generally be larger. The optimum value of liwork for high performance is returned in iwork(1) .
Working Storage	work	Array used for work space. On successful exit, if $lwork > 0$, work(1) contains the optimal work space length lwork for high performance.

	rwork	Array used for work space. On successful exit, rwork (1) contains the optimal work space length lrwork for high performance.
	iwork	Array used for work space. On successful exit, if liwork > 0, iwork (1) contains the optimal work space length liwork for high performance.
Output	ab	Destroyed.
	w	On successful exit, the eigenvalues in ascending order.
	z	On successful exit, if jobz = 'V' or 'v', the orthonormal eigenvectors of the matrix. Not referenced if jobz = 'N' or 'n'.
	info	Status response: info = 0: Successful exit. info < 0: If info = - <i>k</i> , the <i>k</i> -th argument had an invalid value. info > 0: If info = <i>k</i> , the algorithm failed to converge; <i>k</i> off-diagonal elements of an intermediate tridiagonal form did not converge to zero.

Notes

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

jobz ≠ 'N' or 'n' or 'V' or 'v',
uplo ≠ 'L' or 'l' or 'U' or 'u',
n < 0,
kd < 0,
ldab < **kd**+1,
ldz too small,
lwork too small,
lrwork too small, and
liwork too small.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding, for example, the **jobz** argument as 'NoVectors' for 'N' or 'Vectors' for 'V'.

Symmetric or Hermitian Packed Matrix**SSPEV/DSPEV/CHPEV/ZHPEV**

Purpose These subprograms compute all eigenvalues and, optionally, all eigenvectors of a real symmetric or complex Hermitian matrix in packed storage.

A matrix is symmetric if $A = A^T$, its transpose; a matrix is Hermitian if $A = A^*$, its conjugate transpose.

Given such a matrix A , the eigenvalues of A are scalars λ_i , $i = 1, 2, \dots, n$, for which there exist corresponding nonzero vectors, z_i , $i = 1, 2, \dots, n$, such that

$$Az_i = \lambda_i z_i.$$

Optionally, the eigenvectors z_i also may be computed.

Matrix Storage Because either triangle of A may be obtained from the other, you need only provide one triangle of A , either the upper or the lower triangle. Compared to storing the entire matrix, you save memory by supplying that triangle stored column-by-column in packed form in a 1-dimensional array.

The following examples illustrate the packed storage of symmetric or Hermitian matrices.

Upper triangular storage. If the upper triangle of A is

$$\begin{array}{cccc} 11 & 12 & 13 & 14 \\ & 22 & 23 & 24 \\ & & 33 & 34 \\ & & & 44 \end{array}$$

then A is packed column-by-column into an array **ap** as follows:

k	1	2	3	4	5	6	7	8	9	10
ap(k)	11	12	22	13	23	33	14	24	34	44

Upper triangular matrix element a_{ij} is stored in array element **ap**($i+j \times (j-1)/2$).

Lower triangular storage. If the lower triangle of A is

$$\begin{array}{cccc} 11 & & & \\ 21 & 22 & & \\ 31 & 32 & 33 & \\ 41 & 42 & 43 & 44 \end{array}$$

then A is packed column-by-column into an array **ap** as follows:

k	1	2	3	4	5	6	7	8	9	10
ap(k)	11	21	31	41	22	32	42	33	43	44

Lower triangular matrix element a_{ij} is stored in array element **ap**($i+(j-1) \times (2n-j)/2$).

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

CHARACTER*1 jobz, uplo
INTEGER*4 info, ldz, n
REAL*4 ap((n*(n+1))/2), w(n), work(max(1,3*n)), z(ldz, n)
CALL SSPEV (jobz, uplo, n, ap, w, z, ldz, work, info)

CHARACTER*1 jobz, uplo
INTEGER*4 info, ldz, n
REAL*8 ap((n*(n+1))/2), w(n), work(max(1,3*n)), z(ldz, n)
CALL DSPEV (jobz, uplo, n, ap, w, z, ldz, work, info)

CHARACTER*1 jobz, uplo
INTEGER*4 info, ldz, n
REAL*4 rwork(max(1,3*n-1)), w(n)
COMPLEX*8 ap((n*(n+1))/2), work(max(1,2*n-1)), z(ldz, n)
CALL CHPEV (jobz, uplo, n, ap, w, z, ldz, work, rwork, info)

CHARACTER*1 jobz, uplo
INTEGER*4 info, ldz, n
REAL*8 rwork(max(1,3*n-1)), w(n)
COMPLEX*16 ap((n*(n+1))/2), work(max(1,2*n-1)), z(ldz, n)
CALL ZHPEV (jobz, uplo, n, ap, w, z, ldz, work, rwork, info)

LAPACK8, available on C Series and Exemplar architectures:

CHARACTER*1 jobz, uplo
INTEGER*8 info, ldz, n
REAL*8 ap((n*(n+1))/2), w(n), work(max(1,3*n)), z(ldz, n)
CALL SSPEV (jobz, uplo, n, ap, w, z, ldz, work, info)

CHARACTER*1 jobz, uplo
INTEGER*8 info, ldz, n
REAL*8 rwork(max(1,3*n-1)), w(n)
COMPLEX*16 ap((n*(n+1))/2), work(max(1,2*n-1)), z(ldz, n)
CALL CHPEV (jobz, uplo, n, ap, w, z, ldz, work, rwork, info)

Input **jobz** Specifies whether eigenvectors are to be computed, as follows:

jobz = 'N' or 'n': Compute eigenvalues only.
jobz = 'V' or 'v': Compute eigenvectors as well.

uplo Specifies whether the upper or lower triangular part of the symmetric or Hermitian matrix A is stored in array **ap**, as follows:

uplo = 'U' or 'u': The upper triangular part of A is stored.
uplo = 'L' or 'l': The lower triangular part of A is stored.

n The order of the matrix A . $n \geq 0$.

ap The upper or lower triangular part of the symmetric or Hermitian matrix A , packed columnwise in a linear array as follows:

If **uplo** = 'U' or 'u', $ap(i + (j-1) \times j/2) = A(i, j)$ for $1 \leq i \leq j$;
 If **uplo** = 'L' or 'l', $ap(i + (j-1) \times (2n-j)/2) = A(i, j)$ for $j \leq i \leq n$.

	ldz	The leading dimension of array z in the calling program unit. $\text{ldz} \geq 1$. If eigenvectors are desired, then $\text{ldz} \geq \max(1, n)$.
Working Storage	work, rwork	Arrays used for work space.
Output	ap	Destroyed.
	w	On successful exit, the eigenvalues in ascending order. If an error exit is made, the eigenvalues are correct for indices 1, 2, ..., info -1, but they are unordered and may not be the smallest eigenvalues of the matrix.
	z	On successful exit, if jobz = 'V' or 'v', the orthonormal eigenvectors of the matrix. If an error exit is made, z contains the eigenvectors associated with the stored eigenvalues. Not referenced if jobz = 'N' or 'n'.
	info	Status response: info = 0: Successful exit. info < 0: If info = - <i>k</i> , the <i>k</i> -th argument had an invalid value. info > 0: If info = <i>k</i> , the algorithm terminated before finding the <i>k</i> -th eigenvalue.

Notes

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

jobz ≠ 'N' or 'n' or 'V' or 'v',
uplo ≠ 'L' or 'l' or 'U' or 'u',
n < 0,
ldz too small.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding, for example, the **jobz** argument as 'NoVectors' for 'N' or 'Vectors' for 'V'.

Usage

LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

CHARACTER*1 jobz, uplo
INTEGER*4 info, ldz, liwork, lwork, n
INTEGER*4 iwork(liwork)
REAL*4 ap((n*(n+1))/2), w(n), work(lwork), z(ldz, n)
CALL SSPEVD (jobz, uplo, n, ap, w, z, ldz, work, lwork, iwork,
 liwork, info)

CHARACTER*1 jobz, uplo
INTEGER*4 info, ldz, liwork, lwork, n
INTEGER*4 iwork(liwork)
REAL*8 ap((n*(n+1))/2), w(n), work(lwork), z(ldz, n)
CALL DSPEVD (jobz, uplo, n, ap, w, z, ldz, work, lwork, iwork,
 liwork, info)

CHARACTER*1 jobz, uplo
INTEGER*4 info, ldz, liwork, lrwork, lwork, n
INTEGER*4 iwork(liwork)
REAL*4 rwork(lrwork), w(n)
COMPLEX*8 ap((n*(n+1))/2), work(lwork), z(ldz, n)
CALL CHPEVD (jobz, uplo, n, ap, w, z, ldz, work, lwork, rwork,
 lrwork, iwork, liwork, info)

CHARACTER*1 jobz, uplo
INTEGER*4 info, ldz, liwork, lrwork, lwork, n
INTEGER*4 iwork(liwork)
REAL*8 rwork(lrwork), w(n)
COMPLEX*16 ap((n*(n+1))/2), work(lwork), z(ldz, n)
CALL ZHPEVD (jobz, uplo, n, ap, w, z, ldz, work, lwork, rwork,
 lrwork, iwork, liwork, info)

LAPACK8, available on C Series and Exemplar architectures:

CHARACTER*1 jobz, uplo
INTEGER*8 info, ldz, liwork, lwork, n
INTEGER*8 iwork(liwork)
REAL*8 ap((n*(n+1))/2), w(n), work(lwork), z(ldz, n)
CALL SSPEVD (jobz, uplo, n, ap, w, z, ldz, work, lwork, iwork,
 liwork, info)

CHARACTER*1 jobz, uplo
INTEGER*8 info, ldz, liwork, lrwork, lwork, n
INTEGER*8 iwork(liwork)
REAL*8 rwork(lrwork), w(n)
COMPLEX*16 ap((n*(n+1))/2), work(lwork), z(ldz, n)
CALL CHPEVD (jobz, uplo, n, ap, w, z, ldz, work, lwork, rwork,
 lrwork, iwork, liwork, info)

Input	jobz	Specifies whether eigenvectors are to be computed, as follows: jobz = 'N' or 'n': Compute eigenvalues only. jobz = 'V' or 'v': Compute eigenvectors as well.	
	uplo	Specifies whether the upper or lower triangular part of the symmetric or Hermitian matrix A is stored in array ap , as follows: uplo = 'U' or 'u': The upper triangular part of A is stored. uplo = 'L' or 'l': The lower triangular part of A is stored.	
	n	The order of the matrix A . $n \geq 0$.	
	ap	The upper or lower triangular part of the symmetric or Hermitian matrix A , packed columnwise in a linear array as follows: If uplo = 'U' or 'u', $ap(i + (j-1) \times j/2) = A(i, j)$ for $1 \leq i \leq j$; If uplo = 'L' or 'l', $ap(i + (j-1) \times (2n-j)/2) = A(i, j)$ for $j \leq i \leq n$.	
	ldz	The leading dimension of array z in the calling program unit. $ldz \geq 1$. If eigenvectors are desired, then $ldz \geq \max(1, n)$.	
	lwork	The length of array work . For SSPEVD and DSPEVD, $lwork \geq \max(1, 2*n)$ if jobz = 'N' or 'n', and $lwork \geq 2*n**2+5*n+2*n*lg(n)+1$ if jobz = 'V' or 'v', where $lg(n)$ is the smallest integer $k \geq 0$ such that $2^k \geq n$. For CHPEVD and ZHPEVD, $lwork \geq \max(1, n)$ if jobz = 'N' or 'n', and $lwork \geq \max(1, 2*n**2)$ if jobz = 'V' or 'v'. For good performance, lwork must generally be larger. The optimum value of lwork for high performance is returned in work(1) .	
	lrwork	The length of array rwork . $lrwork \geq \max(1, n)$ if jobz = 'N' or 'n', and $lrwork \geq 3*n**2+4*n+2*n*lg(n)+1$ if jobz = 'V' or 'v', where $lg(n)$ is the smallest integer $k \geq 0$ such that $2^k \geq n$. For good performance, lrwork must generally be larger. The optimum value of lrwork for high performance is returned in rwork(1) .	
	liwork	The length of array iwork . $liwork \geq 1$ if jobz = 'N' or 'n', and $liwork \geq 5*n+2$ if jobz = 'V' or 'v'. For good performance, liwork must generally be larger. The optimum value of liwork for high performance is returned in iwork(1) .	
	Working Storage	work	Array used for work space. On successful exit, if $lwork > 0$, work(1) contains the optimal work space length lwork for high performance.
		rwork	Array used for work space. On successful exit, rwork(1) contains the optimal work space length lrwork for high performance.
iwork		Array used for work space. On successful exit, if $liwork > 0$, iwork(1) contains the optimal work space length liwork for high performance.	

Output	ap	Destroyed.
	w	On successful exit, the eigenvalues in ascending order.
	z	On successful exit, if jobz = 'V' or 'v', the orthonormal eigenvectors of the matrix. Not referenced if jobz = 'N' or 'n'.
	info	Status response: info = 0: Successful exit. info < 0: If info = - <i>k</i> , the <i>k</i> -th argument had an invalid value. info > 0: If info = <i>k</i> , the algorithm failed to converge; <i>k</i> off-diagonal elements of an intermediate tridiagonal form did not converge to zero.

Notes If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

jobz ≠ 'N' or 'n' or 'V' or 'v',
uplo ≠ 'L' or 'l' or 'U' or 'u',
n < 0,
ldz too small,
lwork too small,
lrwork too small, and
liwork too small.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding, for example, the **jobz** argument as 'NoVectors' for 'N' or 'Vectors' for 'V'.

Purpose These subprograms compute all eigenvalues and, optionally, all eigenvectors of a real symmetric tridiagonal matrix.

A matrix is symmetric if $A = A^T$, its transpose.

A matrix $A = (a_{ij})$ is tridiagonal if its nonzero elements lie only on the principal diagonal ($i = j$), the subdiagonal ($i = j+1$), and the superdiagonal ($i = j-1$) of the matrix.

Given such a matrix A , the eigenvalues of A are scalars λ_i , $i = 1, 2, \dots, n$, for which there exist corresponding nonzero vectors, z_i , $i = 1, 2, \dots, n$, such that

$$Az_i = \lambda_i z_i.$$

Optionally, the eigenvectors z_i also may be computed.

Matrix Storage The following example illustrates the storage of symmetric tridiagonal matrices. Consider the following symmetric tridiagonal matrix of order $n = 7$:

11	21	0	0	0	0	0
21	22	32	0	0	0	0
0	32	33	43	0	0	0
0	0	43	44	54	0	0
0	0	0	54	55	65	0
0	0	0	0	65	66	76
0	0	0	0	0	76	77

The subdiagonal is stored in array **e** and the principal diagonal is stored in array **d**, as follows:

<i>i</i>	e (<i>i</i>)	d (<i>i</i>)
1	21	11
2	32	22
3	43	33
4	54	44
5	65	55
6	76	66
7		77

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```
CHARACTER*1 jobz
INTEGER*4    info, ldz, n
REAL*4      d(n), e(n-1), work(max(1,2*n-2)), z(ldz, n)
CALL SSTEVD (jobz, n, d, e, z, ldz, work, info)
```

```
CHARACTER*1 jobz
INTEGER*4    info, ldz, n
REAL*8      d(n), e(n-1), work(max(1,2*n-2)), z(ldz, n)
CALL DSTEVD (jobz, n, d, e, z, ldz, work, info)
```

LAPACK8, available on C Series and Exemplar architectures:

```
CHARACTER*1 jobz
INTEGER*8    info, ldz, n
REAL*8      d(n), e(n-1), work(max(1,2*n-2)), z(ldz, n)
CALL SSTEVD (jobz, n, d, e, z, ldz, work, info)
```

Input	jobz	Specifies whether eigenvectors are to be computed, as follows: jobz = 'N' or 'n': Compute eigenvalues only. jobz = 'V' or 'v': Compute eigenvectors as well.
	n	The order in the matrix A . $n \geq 0$.
	d	The n diagonal elements of the tridiagonal matrix.
	e	The $n-1$ subdiagonal elements of the tridiagonal matrix.
	ldz	The leading dimension of array z in the calling program unit. $ldz \geq 1$, and if jobz = 'V' or 'v', then $ldz \geq n$.
Working Storage	work	An array used for work space. Not referenced if jobz = 'N' or 'n'.
Output	d	On successful exit, the eigenvalues, in ascending order. If an error exit is made, the eigenvalues are correct for indices 1, 2, ..., info -1, but they are unordered and may not be the smallest eigenvalues of the matrix.
	e	Destroyed.
	z	On successful exit, if jobz = 'V' or 'v', the orthonormal eigenvectors of the matrix. If an error exit is made, z contains the eigenvectors associated with the stored eigenvalues. Not referenced if jobz = 'N' or 'n'.
	info	Status response: info = 0: Successful exit. info < 0: If info = $-k$, the k -th argument had an invalid value. info > 0: If info = k , the algorithm terminated before finding the k -th eigenvalue.
Notes		If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are jobz \neq 'N' or 'n' or 'V' or 'v', n < 0, ldz too small. Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the CALL statement may be improved by coding, the jobz argument as 'NoVectors' for 'N' or 'Vectors' for 'V'.

Purpose These subprograms compute all eigenvalues and, optionally, all eigenvectors of a real symmetric tridiagonal matrix. If the eigenvectors are desired, the subroutines use a divide-and-conquer algorithm.

A matrix is symmetric if $A = A^T$, its transpose.

A matrix $A = (a_{ij})$ is tridiagonal if its nonzero elements lie only on the principal diagonal ($i = j$), the subdiagonal ($i = j+1$), and the superdiagonal ($i = j-1$) of the matrix.

Given such a matrix A , the eigenvalues of A are scalars λ_i , $i = 1, 2, \dots, n$, for which there exist corresponding nonzero vectors, z_i , $i = 1, 2, \dots, n$, such that

$$Az_i = \lambda_i z_i.$$

Optionally, the eigenvectors z_i also may be computed.

Matrix Storage The following example illustrates the storage of symmetric tridiagonal matrices. Consider the following symmetric tridiagonal matrix of order $n = 7$:

11	21	0	0	0	0	0
21	22	32	0	0	0	0
0	32	33	43	0	0	0
0	0	43	44	54	0	0
0	0	0	54	55	65	0
0	0	0	0	65	66	76
0	0	0	0	0	76	77

The subdiagonal is stored in array **e** and the principal diagonal is stored in array **d**, as follows:

<i>i</i>	e (<i>i</i>)	d (<i>i</i>)
1	21	11
2	32	22
3	43	33
4	54	44
5	65	55
6	76	66
7		77

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```

CHARACTER*1 jobz
INTEGER*4 info, ldz, liwork, lwork, n
INTEGER*4 iwork(liwork)
REAL*4 d(n), e(n-1), work(lwork), z(ldz, n)
CALL SSTEVD (jobz, n, d, e, z, ldz, work, lwork, iwork, liwork,
info)

```

```

CHARACTER*1 jobz
INTEGER*4 info, ldz, liwork, lwork, n
INTEGER*4 iwork(liwork)
REAL*8 d(n), e(n-1), work(lwork), z(ldz, n)
CALL DSTEVD (jobz, n, d, e, z, ldz, work, lwork, iwork, liwork,
info)

```

LAPACK8, available on C Series and Exemplar architectures:

```

CHARACTER*1 jobz
INTEGER*8    info, ldz, liwork, lwork, n
INTEGER*8    iwork(liwork)
REAL*8      d(n), e(n-1), work(lwork), z(ldz, n)
CALL SSTEVD (jobz, n, d, e, z, work, lwork, iwork, liwork,
             info)

```

Input	jobz	Specifies whether eigenvectors are to be computed, as follows: jobz = 'N' or 'n': Compute eigenvalues only. jobz = 'V' or 'v': Compute eigenvectors as well.
	n	The order in the matrix A . $n \geq 0$.
	d	The n diagonal elements of the tridiagonal matrix.
	e	The $n-1$ subdiagonal elements of the tridiagonal matrix.
	ldz	The leading dimension of array z in the calling program unit. $ldz \geq 1$, and if jobz = 'V' or 'v', then $ldz \geq n$.
	lwork	The length of array work . $lwork \geq 1$ if jobz = 'N' or 'n', and $lwork \geq 2*n**2+3*n+2*n*lg(n)+1$ if jobz = 'V' or 'v', where $lg(n)$ is the smallest integer $k \geq 0$ such that $2^k \geq n$. The optimal value of lwork is returned in work(1) .
	liwork	The length of array iwork . $liwork \geq 1$ if jobz = 'N' or 'n', and $liwork \geq 5*n+2$ if jobz = 'V' or 'v'. The optimal value of liwork is returned in iwork(1) .
Working Storage	work	Array used for work space. On successful exit, if $lwork > 0$, work(1) contains the optimal work space length lwork .
	iwork	Array used for work space. On successful exit, if $liwork > 0$, iwork(1) contains the optimal work space length liwork .
Output	d	On successful exit, the eigenvalues, in ascending order.
	e	Destroyed.
	z	On successful exit, if jobz = 'V' or 'v', the orthonormal eigenvectors of the matrix. Not referenced if jobz = 'N' or 'n'.
	info	Status response: info = 0: Successful exit. info < 0: If info = $-k$, the k -th argument had an invalid value. info > 0: If info = k , the algorithm failed to converge; k off-diagonal elements of e did not converge to zero.

Notes

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

jobz ≠ 'N' or 'n' or 'V' or 'v',
n < 0,
ldz too small,
lwork too small, and
liwork too small.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding, the **jobz** argument as 'NoVectors' for 'N' or 'Vectors' for 'V'.

Symmetric or Hermitian Matrix**SSYEV/DSYEV/CHEEV/ZHEEV**

Purpose These subprograms compute all eigenvalues and, optionally, all eigenvectors of a real symmetric or complex Hermitian matrix.

A matrix is symmetric if $A = A^T$, its transpose; a matrix is Hermitian if $A = A^*$, its conjugate transpose.

Given such a matrix A , the eigenvalues of A are scalars λ_i , $i = 1, 2, \dots, n$, for which there exist corresponding nonzero vectors, z_i , $i = 1, 2, \dots, n$, such that

$$Az_i = \lambda_i z_i.$$

Optionally, the eigenvectors z_i also may be computed.

Matrix Storage Because either triangle of A may be obtained from the other, you need only provide one triangle of A . You may supply either the upper or the lower triangle of A , in a two-dimensional array large enough to hold the entire matrix. The other triangle of the array is not referenced.

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```
CHARACTER*1 jobz, uplo
INTEGER*4   info, lda, lwork, n
REAL*4      a(lda, n), w(n), work(lwork)
CALL SSYEV (jobz, uplo, n, a, lda, w, work, lwork, info)
```

```
CHARACTER*1 jobz, uplo
INTEGER*4   info, lda, lwork, n
REAL*8      a(lda, n), w(n), work(lwork)
CALL DSYEV (jobz, uplo, n, a, lda, w, work, lwork, info)
```

```
CHARACTER*1 jobz, uplo
INTEGER*4   info, lda, lwork, n
REAL*4      rwork(max(1,3*n-2)), w(n)
COMPLEX*8   a(lda, n), work(lwork)
CALL CHEEV (jobz, uplo, n, a, lda, w, work, lwork, rwork, info)
```

```
CHARACTER*1 jobz, uplo
INTEGER*4   info, lda, lwork, n
REAL*8      rwork(max(1,3*n-2)), w(n)
COMPLEX*16  a(lda, n), work(lwork)
CALL ZHEEV (jobz, uplo, n, a, lda, w, work, lwork, rwork, info)
```

LAPACK8, available on C Series and Exemplar architectures:

```
CHARACTER*1 jobz, uplo
INTEGER*8   info, lda, lwork, n
REAL*8      a(lda, n), w(n), work(lwork)
CALL S8YEV (jobz, uplo, n, a, lda, w, work, lwork, info)
```

```
CHARACTER*1 jobz, uplo
INTEGER*8   info, lda, lwork, n
REAL*8      rwork(max(1,3*n-2)), w(n)
COMPLEX*16  a(lda, n), work(lwork)
CALL C8EEV (jobz, uplo, n, a, lda, w, work, lwork, rwork, info)
```

Input	jobz	Specifies whether eigenvectors are to be computed, as follows: jobz = 'N' or 'n': Compute eigenvalues only. jobz = 'V' or 'v': Compute eigenvectors as well.	
	uplo	Specifies whether the upper or lower triangular part of the symmetric or Hermitian matrix <i>A</i> is stored, as follows: uplo = 'U' or 'u': The upper triangular part of <i>A</i> is stored. uplo = 'L' or 'l': The lower triangular part of <i>A</i> is stored.	
	n	The order of the matrix <i>A</i> . $n \geq 0$.	
	a	The symmetric or Hermitian matrix <i>A</i> . If uplo = 'U' or 'u', only the upper triangular part of <i>a</i> is used to define the elements of the matrix and the strict lower triangular part of <i>a</i> is not used for input. If uplo = 'L' or 'l', only the lower triangular part of <i>a</i> is used to define the elements of the matrix and the strict upper triangular part of <i>a</i> is not used for input.	
	lda	The leading dimension of array <i>a</i> in the calling program unit. $lda \geq \max(1, n)$.	
	lwork	The length of array <i>work</i> . $lwork \geq \max(1, 3n-1)$. For good performance, <i>lwork</i> must generally be larger. The optimum value of <i>lwork</i> for high performance is returned in <i>work</i> (1).	
	Working Storage	work, rwork	Arrays used for work space. On successful exit, <i>work</i> (1) contains the optimal work space length <i>lwork</i> for high performance.
	Output	a	On successful exit, if jobz = 'V' or 'v', the orthonormal eigenvectors of the matrix. If an error exit is made, <i>a</i> contains the eigenvectors associated with the stored eigenvalues. If jobz = 'N' or 'n', then the triangle of <i>a</i> specified by uplo , including the diagonal, has been destroyed.
		w	On successful exit, the eigenvalues in ascending order. If an error exit is made, the eigenvalues are correct for indices 1, 2, ..., <i>info</i> -1, but they are unordered and may not be the smallest eigenvalues of the matrix.
		info	Status response: info = 0: Successful exit. info < 0: If info = - <i>k</i> , the <i>k</i> -th argument had an invalid value. info > 0: If info = <i>k</i> , the algorithm terminated before finding the <i>k</i> -th eigenvalue.

Notes

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

jobz ≠ 'N' or 'n' or 'V' or 'v',
uplo ≠ 'L' or 'l' or 'U' or 'u',
n < 0,
lda < max(1,n), and
lwork < max(1,3n-1).

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding the **jobz** argument as 'NoVectors' for 'N' or 'Vectors' for 'V'.

Purpose These subprograms compute all eigenvalues and, optionally, all eigenvectors of a real symmetric or complex Hermitian matrix. If the eigenvectors are desired, the subroutines use a divide-and-conquer algorithm.

A matrix is symmetric if $A = A^T$, its transpose; a matrix is Hermitian if $A = A^*$, its conjugate transpose.

Given such a matrix A , the eigenvalues of A are scalars λ_i , $i = 1, 2, \dots, n$, for which there exist corresponding nonzero vectors, z_i , $i = 1, 2, \dots, n$, such that

$$Az_i = \lambda_i z_i.$$

Optionally, the eigenvectors z_i also may be computed.

Matrix Storage Because either triangle of A may be obtained from the other, you need only provide one triangle of A . You may supply either the upper or the lower triangle of A , in a two-dimensional array large enough to hold the entire matrix. The other triangle of the array is not referenced.

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```

CHARACTER*1 jobz, uplo
INTEGER*4 info, lda, liwork, lwork, n
INTEGER*4 iwork(liwork)
REAL*4 a(lda, n), w(n), work(lwork)
CALL SSYEVD (jobz, uplo, n, a, lda, w, work, lwork, iwork,
              liwork, info)

```

```

CHARACTER*1 jobz, uplo
INTEGER*4 info, lda, liwork, lwork, n
INTEGER*4 iwork(liwork)
REAL*8 a(lda, n), w(n), work(lwork)
CALL DSYEVD (jobz, uplo, n, a, lda, w, work, lwork, iwork,
              liwork, info)

```

```

CHARACTER*1 jobz, uplo
INTEGER*4 info, lda, liwork, lrwork, lwork, n
INTEGER*4 iwork(liwork)
REAL*4 rwork(lrwork), w(n)
COMPLEX*8 a(lda, n), work(lwork)
CALL CHEEVD (jobz, uplo, n, a, lda, w, work, lwork, rwork,
              lrwork, iwork, liwork, info)

```

```

CHARACTER*1 jobz, uplo
INTEGER*4 info, lda, liwork, lrwork, lwork, n
INTEGER*4 iwork(liwork)
REAL*8 rwork(lrwork), w(n)
COMPLEX*16 a(lda, n), work(lwork)
CALL ZHEEVD (jobz, uplo, n, a, lda, w, work, lwork, rwork,
              lrwork, iwork, liwork, info)

```

LAPACK8, available on C Series and Exemplar architectures:

```

CHARACTER*1 jobz, uplo
INTEGER*8 info, lda, liwork, lwork, n
INTEGER*8 iwork(liwork)
REAL*8 a(lda, n), w(n), work(lwork)
CALL SSYEVD (jobz, uplo, n, a, lda, w, work, lwork, iwork,
              liwork, info)
    
```

```

CHARACTER*1 jobz, uplo
INTEGER*8 info, lda, liwork, lrwork, lwork, n
INTEGER*8 iwork(liwork)
REAL*8 rwork(lrwork), w(n)
COMPLEX*16 a(lda, n), work(lwork)
CALL CHEEVD (jobz, uplo, n, a, lda, w, work, lwork, rwork,
              lrwork, iwork, liwork, info)
    
```

Input

jobz Specifies whether eigenvectors are to be computed, as follows:

jobz = 'N' or 'n': Compute eigenvalues only.
jobz = 'V' or 'v': Compute eigenvectors as well.

uplo Specifies whether the upper or lower triangular part of the symmetric or Hermitian matrix *A* is stored, as follows:

uplo = 'U' or 'u': The upper triangular part of *A* is stored.
uplo = 'L' or 'l': The lower triangular part of *A* is stored.

n The order of the matrix *A*. $n \geq 0$.

a The symmetric or Hermitian matrix *A*.

If **uplo = 'U' or 'u'**, only the upper triangular part of **a** is used to define the elements of the matrix and the strict lower triangular part of **a** is not used for input.

If **uplo = 'L' or 'l'**, only the lower triangular part of **a** is used to define the elements of the matrix and the strict upper triangular part of **a** is not used for input.

lda The leading dimension of array **a** in the calling program unit. $lda \geq \max(1, n)$.

lwork The length of array **work**.

For SSYEVD and DSYEVD, $lwork \geq 2*n+1$ if **jobz = 'N' or 'n'**, and $lwork \geq 3*n**2+5*n+2*n*lg(n)+1$ if **jobz = 'V' or 'v'**, where $lg(n)$ is the smallest integer $k \geq 0$ such that $2^k \geq n$.

For CHEEVD and ZHEEVD, $lwork \geq n+1$ if **jobz = 'N' or 'n'**, and $lwork \geq \max(1, 2*n+n**2)$ if **jobz = 'V' or 'v'**.

For good performance, **lwork** must generally be larger. The optimum value of **lwork** for high performance is returned in **work(1)**.

	lrwork	The length of array rwork . $\text{lrwork} \geq \max(1, n)$ if jobz = 'N' or 'n', and $\text{lrwork} \geq 3*n**2+4*n+2*n*\lg(n)+1$ if jobz = 'V' or 'v', where $\lg(n)$ is the smallest integer $k \geq 0$ such that $2^k \geq n$. For good performance, lrwork must generally be larger. The optimum value of lrwork for high performance is returned in rwork(1) .
	liwork	The length of array iwork . $\text{liwork} \geq 1$ if jobz = 'N' or 'n', and $\text{liwork} \geq 5*n+2$ if jobz = 'V' or 'v'. For good performance, liwork must generally be larger. The optimum value of liwork for high performance is returned in iwork(1) .
Working Storage	work	Array used for work space. On successful exit, if liwork > 0, work(1) contains the optimal work space length liwork for high performance.
	rwork	Array used for work space. On successful exit, rwork(1) contains the optimal work space length lrwork for high performance.
	iwork	Array used for work space. On successful exit, if liwork > 0, iwork(1) contains the optimal work space length liwork for high performance.
Output	a	On successful exit, if jobz = 'V' or 'v', the orthonormal eigenvectors of the matrix. If jobz = 'N' or 'n', then the triangle of a specified by uplo , including the diagonal, has been destroyed.
	w	On successful exit, the eigenvalues in ascending order.
	info	Status response: info = 0: Successful exit. info < 0: If info = - k , the k -th argument had an invalid value. info > 0: If info = k , the algorithm failed to converge; k off-diagonal elements of an intermediate tridiagonal form did not converge to zero.
Notes	If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are	
	<p style="margin-left: 40px;"> jobz ≠ 'N' or 'n' or 'V' or 'v', uplo ≠ 'L' or 'I' or 'U' or 'u', n < 0, lda < max(1,n), liwork too small, lrwork too small, and liwork too small. </p>	
	Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the CALL statement may be improved by coding the jobz argument as 'NoVectors' for 'N' or 'Vectors' for 'V'.	

Expert Drivers for Ordinary Eigenvalue Problems

Overview

This chapter explains how to use LAPACK expert drivers to compute the Schur form, Schur vectors, eigenvalues, or eigenvalues and eigenvectors of matrices. The problems covered are:

- general dense eigenproblems, $Ax = \lambda x$, for arbitrary A
- symmetric and Hermitian dense eigenproblems, $Ax = \lambda x$, with $A = A^T$ or $A = A^*$
- symmetric and Hermitian banded eigenproblems, $Ax = \lambda x$, with $A = A^T$ or $A = A^*$
- symmetric tridiagonal eigenproblems, $Ax = \lambda x$, with $A = A^T$

Chapter 7 describes the LAPACK simple drivers for ordinary eigenvalue problems. Use the simple drivers when you want to compute all of the eigenvalues of a matrix, instead of only some of them. Refer to Chapter 9 for software to compute the eigenvalues or eigenvalues and eigenvectors of generalized eigenproblems.

Refer to Chapter 7 of the *ConvexMLIB User's Guide: VECLIB* for software to compute the eigenvalues or eigenvalues and eigenvectors of a real symmetric, sparse, ordinary or generalized eigenproblem.

Chapter Objectives

After reading this chapter you will know how to use the described subprograms.

What You Need to Know to Use These Subprograms

To use these subprograms, it is helpful to know some basic theory of eigensystems and the Schur Form, especially as they relate to your particular application. A few facts discussed in basic textbooks are given in the introduction to Chapter 7.

The eigenvalues for real symmetric or complex Hermitian matrices are real. The subprograms in this chapter which deal with this type of matrix allow you to select only some of the eigenvalues. For example, you can select all the eigenvalues in a half-open interval $a < \lambda_j \leq b$, or you can choose a range of indices of ordered eigenvalues, such as the five smallest, the seven largest, or the twelfth smallest through the seventeenth smallest.

The eigenvalues of a general nonsymmetric matrix may change radically as a result of small perturbations in the elements of the matrix. The subprograms in this chapter which solve the eigenproblem for a general matrix optionally compute a condition number which measures the sensitivity of eigenvalues. An estimate of a condition number for the eigenvectors also can be computed.

Since real symmetric and complex Hermitian eigenvalues are always well conditioned, no condition numbers are estimated by the subprograms for these types of matrices.

Supplemental Reading

- Anderson, E. *et al.* *LAPACK Users' Guide*, Second Edition. Philadelphia, PA: Society for Industrial and Applied Mathematics. 1995.
- Golub, G.H. and C.F. Van Loan. *Matrix Computations*, 2nd Edition. Baltimore, MD: The Johns Hopkins University Press. 1989.
- Parlett, B.N. *The Symmetric Eigenproblem*. Englewood Cliffs, NJ: Prentice-Hall, Inc. 1980.
- Wilkinson, J.H. *The Algebraic Eigenproblem*. New York, NY: Oxford University Press. 1965.

Subprogram Descriptions

Eigenvalues and Schur Form of a General Matrix SGEESX, DGEESX, CGEESX, ZGEESX	8-3
Eigenvalues and Eigenvectors of a General Matrix SGEEVX, DGEEVX, CGEEVX, ZGEEVX	8-9
Eigenvalues and Eigenvectors of a Symmetric or Hermitian Band Matrix SSBEVX, DSBEVX, CHBEVX, ZHBEVX	8-15
Eigenvalues and Eigenvectors of a Symmetric or Hermitian Packed Matrix SSPEVX, DSPEVX, CHPEVX, ZHPEVX	8-20
Eigenvalues and Eigenvectors of a Real Symmetric Tridiagonal Matrix SSTEVS, DSTEVX	8-25
Eigenvalues and Eigenvectors of a Symmetric or Hermitian Matrix SSYEVS, DSYEVX, CHEEVX, ZHEEVX	8-28

Schur Form of a General Matrix**SGEESX/DGEESX/CGEESX/ZGEESX**

Purpose These subprograms compute the eigenvalues and the Schur Form of an n -by- n general matrix A , and optionally compute the Schur vectors of A and order the eigenvalues on the diagonal of the Schur Form so that selected eigenvalues are at the upper left.

Given a general matrix A , the eigenvalues of A are scalars λ_i , $i = 1, 2, \dots, n$, for which there exist corresponding nonzero vectors, z_i , $i = 1, 2, \dots, n$, such that

$$Az_i = \lambda_i z_i.$$

A real matrix is in Schur Form if it is block upper triangular with 1-by-1 and 2-by-2 blocks. 1-by-1 blocks correspond to real eigenvalues, and 2-by-2 blocks correspond to complex conjugate eigenpairs. 2-by-2 blocks are standardized in the form

$$\begin{bmatrix} a & b \\ c & a \end{bmatrix}$$

where $bc < 0$. The eigenvalues of such a block are $a \pm i\sqrt{|bc|}$.

A complex matrix is in Schur Form if it upper triangular.

If T is the Schur Form of A , then the columns of unitary matrix Q such that

$$A = QTQ^*$$

are known as the Schur vectors of A .

Also, optionally, two condition numbers may be estimated; one measures the sensitivity of the average of the eigenvalues to errors in the matrix or to small roundoff errors introduced during the computation, and the other estimates the conditioning of the right invariant subspace corresponding to the selected eigenvalues.

Usage

LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```

CHARACTER*1 jobvs, sense, sort
INTEGER*4 info, lda, ldvs, liwork, lwork, n, sdim
REAL*4 rconde, rcondv
LOGICAL*4 bwork(n)
INTEGER*4 iwork(liwork)
REAL*4 a(lda, n), vs(ldvs, n), wi(n), work(lwork), wr(n)
LOGICAL*4 select
EXTERNAL select
CALL SGEESX (jobvs, sort, select, sense, n, a, lda, sdim, wr, wi,
vs, ldvs, rconde, rcondv, work, lwork, iwork,
liwork, bwork, info)

```

CHARACTER*1 jobvs, sense, sort
INTEGER*4 info, lda, ldvs, liwork, lwork, n, sdim
REAL*8 rconde, rcondv
LOGICAL*4 bwork(n)
INTEGER*4 iwork(liwork)
REAL*8 a(lda, n), vs(ldvs, n), wi(n), work(lwork), wr(n)
LOGICAL*4 select
EXTERNAL select
CALL DGEESX (jobvs, sort, select, sense, n, a, lda, sdim, wr, wi, vs, ldvs, rconde, rcondv, work, lwork, iwork, liwork, bwork, info)

CHARACTER*1 jobvs, sense, sort
INTEGER*4 info, lda, ldvs, lwork, n, sdim
REAL*4 rconde, rcondv
LOGICAL*4 bwork(n)
REAL*4 rwork(n)
COMPLEX*8 a(lda, n), vs(ldvs, n), w(n), work(lwork)
LOGICAL*4 select
EXTERNAL select
CALL CGEESX (jobvs, sort, select, sense, n, a, lda, sdim, w, vs, ldvs, rconde, rcondv, work, lwork, rwork, bwork, info)

CHARACTER*1 jobvs, sense, sort
INTEGER*4 info, lda, ldvs, lwork, n, sdim
REAL*8 rconde, rcondv
LOGICAL*4 bwork(n)
REAL*8 rwork(n)
COMPLEX*16 a(lda, n), vs(ldvs, n), w(n), work(lwork)
LOGICAL*4 select
EXTERNAL select
CALL ZGEEEX (jobvs, sort, select, sense, n, a, lda, sdim, w, vs, ldvs, rconde, rcondv, work, lwork, rwork, bwork, info)

LAPACK8, available on C Series and Exemplar architectures:

CHARACTER*1 jobvs, sense, sort
INTEGER*8 info, lda, ldvs, liwork, lwork, n, sdim
REAL*8 rconde, rcondv
LOGICAL*8 bwork(n)
INTEGER*4 iwork(liwork)
REAL*8 a(lda, n), vs(ldvs, n), wi(n), work(lwork), wr(n)
LOGICAL*8 select
EXTERNAL select
CALL SGEESX (jobvs, sort, select, sense, n, a, lda, sdim, wr, wi, vs, ldvs, rconde, rcondv, work, lwork, iwork, liwork, bwork, info)

CHARACTER*1 jobvs, sense, sort
INTEGER*8 info, lda, ldvs, lwork, n, sdim
REAL*8 rconde, rcondv
LOGICAL*8 bwork(n)
REAL*8 rwork(n)
COMPLEX*16 a(lda, n), vs(ldvs, n), w(n), work(lwork)
LOGICAL*8 select
EXTERNAL select
CALL CGEESX (jobvs, sort, select, sense, n, a, lda, sdim, w,
vs, ldvs, rconde, rcondv, work, lwork, rwork,
bwork, info)

Input

jobvs Specifies whether the Schur vectors are to be computed, as follows:

jobvs = 'N' or 'n': Schur vectors are not computed.
jobvs = 'V' or 'v': Schur vectors are computed.

sort Specifies whether the eigenvalues on the diagonal of the Schur Form are to be reordered, as follows:

sort = 'N' or 'n': The eigenvalues are not specially ordered.
sort = 'S' or 's': The eigenvalues are ordered so that the eigenvalues λ_j for which the logical function **select**(**wr**(*j*),**wi**(*j*)) (in SGEESX and DGEESX) or **select**(**w**(*j*)) (in CGEESX and ZGEESX) is true will appear at the top left corner of the Schur Form. (In SGEESX and DGEESX, if an eigenvalue is complex and either **select**(**wr**(*j*),**wi**(*j*)) or **select**(**wr**(*j*),**-wi**(*j*)) is true, both eigenvalues are selected.)

select The name of a user-defined function subprogram used if **sort = 'S' or 's'** to select eigenvalues to reorder to the upper left of the Schur Form. For SGEESX and DGEESX, **select** requires two arguments of the same type as **wr** and **wi**, while for CGEESX and ZGEESX, **select** requires one argument of the same type as **w**. The eigenvalue λ_j is selected if **select**(**wr**(*j*),**wi**(*j*)) or **select**(**w**(*j*)) is true. Note that a selected complex eigenvalue may no longer satisfy **select**(λ_j) = .TRUE. after ordering, since ordering may perturb the value of complex eigenvalues, and especially ill-conditioned ones; in this case **info** may be set to a positive value (see **info** below). **select** must be declared **EXTERNAL** in the calling subroutine. **select** is not referenced if **sort = 'N' or 'n'**.

sense Specifies which reciprocal condition numbers are to be computed, as follows:

sense = 'N' or 'n': None.
sense = 'E' or 'e': Compute **rconde** only.
sense = 'V' or 'v': Compute **rcondv** only.
sense = 'B' or 'b': Compute both **rconde** and **rcondv**.

If **sense = 'E' or 'e' or 'V' or 'v' or 'B' or 'b'**, then **sort** must be **'S' or 's'**.

n The order of the matrix *A*. $n \geq 0$.

a The *n*-by-*n* matrix *A*.

	lda	The leading dimension of array a in the calling program unit. lda $\geq \max(1, n)$.
	ldvs	The leading dimension of array vs in the calling program unit. ldvs ≥ 1 , and if jobvs = 'V' or 'v', then ldvs $\geq n$.
	lwork	The length of array work . lwork $\geq \max(1, 3n)$. If sense = 'E' or 'e' or 'V' or 'v' or 'B' or 'b', then lwork $\geq n + 2\text{sdim} \times (n - \text{sdim})$ where sdim is the total number of eigenvalues selected. Note that $n + 2\text{sdim} \times (n - \text{sdim}) \leq n \times (1 + n/2)$. For good performance, lwork must generally be larger. The optimum value of lwork for high performance is returned in work(1) .
	liwork	The length of array iwork . If sense = 'V' or 'v' or 'B' or 'b', then liwork $\geq \text{sdim} \times (n - \text{sdim})$ where sdim is the total number of eigenvalues selected. Not referenced if sense = 'N' or 'n' or 'E' or 'e'.
Working Storage	work	An array used for work space. On successful exit, work(1) contains the optimal work space length lwork for high performance.
	iwork	An array used for work space. Not referenced if sense = 'N' or 'n' or 'E' or 'e'.
	rwork	An array used for work space.
	bwork	An array used for work space. Not referenced if sort = 'N' or 'n'.
Output	a	On successful exit, the Schur Form of A overwrites the input. If sort = 'S' or 's', the output Schur Form is $\begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{bmatrix}$ where A_{11} is sdim -by- sdim , A_{12} is sdim -by- $(n - \text{sdim})$, and A_{22} is $(n - \text{sdim})$ -by- $(n - \text{sdim})$, and where the eigenvalues λ_j , $j = 1, 2, \dots, \text{sdim}$, of A_{11} satisfy select (λ_j) = .TRUE. and the eigenvalues λ_j , $j = \text{sdim} + 1, \text{sdim} + 2, \dots, n$, of A_{22} satisfy select (λ_j) = .FALSE.
	sdim	If sort = 'N' or 'n', sdim = 0. If sort = 'S' or 's', sdim is the number of eigenvalues (after reordering) for which select is true. In SGEESX and DGEESX, complex conjugate pairs for which select is true for either eigenvalue count as 2.
	wr, wi	On successful exit from SGEESX and DGEESX, wr(j) and wi(j) contain the real and imaginary parts, respectively, of the computed eigenvalue λ_j , $j = 1, 2, \dots, n$. The eigenvalues are in the same order in which they appear as the eigenvalues of the diagonal 1-by-1 and 2-by-2 blocks of the Schur Form. Complex conjugate pairs of eigenvalues appear consecutively with the eigenvalue having the positive imaginary part first.
	w	On successful exit from CGEESX and ZGEESX, w(j) contains the computed eigenvalue λ_j , $j = 1, 2, \dots, n$. The eigenvalues are in the same order in which they appear on the diagonal of the Schur Form.

vs On successful exit, if **jobvs** = 'V' or 'v', the Schur vectors of A . Not referenced if **jobvs** = 'N' or 'n'.

rconde On successful exit, if **sense** = 'E' or 'e' or 'B' or 'b', the reciprocal condition number for the average of the selected eigenvalues. **rconde** measures the sensitivity of the average of the eigenvalues of A_{11} :

$$(\lambda_1 + \lambda_2 + \cdots + \lambda_{\text{sdim}})/\text{sdim}$$

$0 \leq \text{rconde} \leq 1$, with **rconde** ≈ 0 indicating very poor conditioning, and **rconde** ≈ 1 indicating very good conditioning. **rconde** is computed as follows. First, we compute R so that

$$P = \begin{bmatrix} I & R \\ 0 & 0 \end{bmatrix}$$

is the projector on the invariant subspace associated with A_{11} . R is the solution of the Sylvester equation

$$A_{11}R - RA_{22} = A_{12}.$$

Then **rconde** $= (1 + \|R\|_F^2)^{-1/2}$, where $\|\cdot\|_F$ is the Frobenius norm. **rconde** underestimates the true reciprocal condition number, but not by more than a factor of \sqrt{n} .

An approximate bound on the error between the computed average of the eigenvalues of A_{11} is $\epsilon \|A\|/\text{rconde}$, where ϵ is the machine precision.

rcondv On successful exit, if **sense** = 'V' or 'v' or 'B' or 'b', the reciprocal condition number for the average of the selected right invariant subspace, that is, the subspace spanned by A_{11} . **rcondv** is defined as the separation of A_{11} and A_{22} :

$$\text{sep}(A_{11}, A_{22}) = \sigma_{\min}(K)$$

where σ_{\min} is the smallest singular value of the $\text{sdim} \times (\text{n} - \text{sdim})$ -by- $\text{sdim} \times (\text{n} - \text{sdim})$ matrix

$$K = I_{\text{n} - \text{sdim}} \otimes A_{11} - A_{22}^T \otimes I_{\text{sdim}}$$

where I_m is an m -by- m identity matrix and \otimes denotes the Kronecker product. The smallest singular value is approximated by the reciprocal of an estimate of $\|K^{-1}\|_1$. **rcondv** cannot differ from the true reciprocal condition number by more than a factor of $(\text{sdim} \times (\text{n} - \text{sdim}))^{1/2}$. When **rcondv** is small, small changes in A can cause large changes in the invariant subspace spanned by A_{11} . An approximate bound on the maximum angular error in the computed invariant subspace is $\epsilon \|A\|/\text{rcondv}$, where ϵ is the machine precision.

info Status response:

info = 0: Successful exit.

info < 0: If **info** = $-k$, the k -th argument had an invalid value.

info > 0: The algorithm terminated before completing the requested computation.

Notes

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

jobvs ≠ 'N' or 'n' or 'V' or 'v'
sort ≠ 'N' or 'n' or 'S' or 's'
sense ≠ 'N' or 'n' or 'E' or 'e' or 'V' or 'v' or 'B' or 'b',
sense = 'E' or 'e' or 'V' or 'v' or 'B' or 'b' and **sort** ≠ 'S' or 's',
n < 0,
lda < max(1,n),
ldvs too small, and
lwork too small.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding, for example, the **jobvs** argument as 'NoVectors' for 'N' or 'Vectors' for 'V'.

General Matrix Eigenproblem**SGEEVX/DGEEVX/CGEEVX/ZGEEVX****Purpose**

These subprograms compute all eigenvalues and, optionally, all left and/or right eigenvectors of an n -by- n nonsymmetric matrix A . Optionally, A may be balanced to improve the conditioning of its eigenvalues and eigenvectors. Balancing a matrix means permuting its rows and columns into a more nearly upper triangular form, and/or computing a similar matrix DAD^{-1} , where D is a diagonal scaling matrix, to make its rows and columns closer in norm and the condition numbers of its eigenvalues and eigenvectors smaller.

Given such a matrix A , the eigenvalues of A are scalars λ_i , $i = 1, 2, \dots, n$, for which there exist corresponding nonzero vectors, z_i , $i = 1, 2, \dots, n$, such that

$$Az_i = \lambda_i z_i.$$

Optionally, the z_i , which are called right eigenvectors, also may be computed. In addition, these subprograms also can compute the left eigenvectors, which satisfy

$$y_i^* A = \lambda_i y_i^*.$$

Also optionally, two sets of condition numbers may be estimated; one set measures the sensitivity of the eigenvalues to errors in the matrix or to small roundoff errors introduced during the computation, and the other set estimates the conditioning of the eigenvectors.

Usage

LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```

CHARACTER*1 balanc, jobvl, jobvr, sense
INTEGER*4   ihi, ilo, info, lda, ldvl, ldvr, lwork, n
REAL*4     abnrm
INTEGER*4   iwork(max(1,2*n-2))
REAL*4     a(lda, n), rconde(n), rcondv(n), scale(n),
             vl(ldvl, n), vr(ldvr, n), wi(n), work(lwork), wr(n)
CALL 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*1 balanc, jobvl, jobvr, sense
INTEGER*4   ihi, ilo, info, lda, ldvl, ldvr, lwork, n
REAL*8     abnrm
INTEGER*4   iwork(max(1,2*n-2))
REAL*8     a(lda, n), rconde(n), rcondv(n), scale(n),
             vl(ldvl, n), vr(ldvr, n), wi(n), work(lwork), wr(n)
CALL DGEEVX (balanc, jobvl, jobvr, sense, n, a, lda, wr, wi,
             vl, ldvl, vr, ldvr, ilo, ihi, scale, abnrm, rconde,
             rcondv, work, lwork, iwork, info)

```

```

CHARACTER*1 balanc, jobvl, jobvr, sense
INTEGER*4   ihi, ilo, info, lda, ldvl, ldvr, lwork, n
REAL*4     abnrm
REAL*4     rconde(n), rcondv(n), rwork(2*n), scale(n)
COMPLEX*8  a(lda, n), vl(ldvl, n), vr(ldvr, n), w(n),
             work(lwork)
CALL CGEEVX (balanc, jobvl, jobvr, sense, n, a, lda, w, vl, ldvl,
             vr, ldvr, ilo, ihi, scale, abnrm, rconde, rcondv,
             work, lwork, rwork, info)

```

CHARACTER*1 balanc, jobvl, jobvr, sense
INTEGER*4 ihi, ilo, info, lda, ldvl, ldvr, lwork, n
REAL*8 abnrm
REAL*8 rconde(n), rcondv(n), rwork(2*n), scale(n)
COMPLEX*16 a(lda, n), vl(ldvl, n), vr(ldvr, n), w(n),
work(lwork)
CALL ZGEEVX (balanc, jobvl, jobvr, sense, n, a, lda, w, vl, ldvl,
vr, ldvr, ilo, ihi, scale, abnrm, rconde, rcondv,
work, lwork, rwork, info)

LAPACK8, available on C Series and Exemplar architectures:

CHARACTER*1 balanc, jobvl, jobvr, sense
INTEGER*8 ihi, ilo, info, lda, ldvl, ldvr, lwork, n
REAL*8 abnrm
INTEGER*8 iwork(max(1,2*n-2))
REAL*8 a(lda, n), rconde(n), rcondv(n), scale(n),
vl(ldvl, n), vr(ldvr, n), wi(n), work(lwork), wr(n)
CALL SGEEVX (balanc, jobvl, jobvr, sense, n, a, lda, w, vl,
vl, ldvl, vr, ldvr, ilo, ihi, scale, abnrm, rconde,
rcondv, work, lwork, iwork, info)

CHARACTER*1 balanc, jobvl, jobvr, sense
INTEGER*8 ihi, ilo, info, lda, ldvl, ldvr, lwork, n
REAL*8 abnrm
REAL*8 rconde(n), rcondv(n), rwork(2*n), scale(n)
COMPLEX*16 a(lda, n), vl(ldvl, n), vr(ldvr, n), w(n),
work(lwork)
CALL CGEEVX (balanc, jobvl, jobvr, sense, n, a, lda, w, vl, ldvl,
vr, ldvr, ilo, ihi, scale, abnrm, rconde, rcondv,
work, lwork, rwork, info)

Input

balanc Specifies how A should be permuted and/or diagonally scaled to improve the conditioning of its eigenvalues and eigenvectors, as follows:

balanc = 'N' or 'n': Do not permute or diagonally scale A .
balanc = 'P' or 'p': Permute A into a more nearly upper triangular form, but do not diagonally scale A .
balanc = 'S' or 's': Scale A , that is, replace A by DAD^{-1} where D is a diagonal scaling matrix chosen to make the rows and columns of DAD^{-1} more equal in norm, but do not permute A .
balanc = 'B' or 'b': Both permute and diagonally scale A .

jobvl Specifies whether the left eigenvectors of A are to be computed, as follows:

jobvl = 'N' or 'n': Do not compute the left eigenvectors.
jobvl = 'V' or 'v': Compute the left eigenvectors.

jobvr Specifies whether the right eigenvectors of A are to be computed, as follows:

jobvr = 'N' or 'n': Do not compute the right eigenvectors.
jobvr = 'V' or 'v': Compute the right eigenvectors.

	sense	Specifies which reciprocal condition numbers are to be computed, as follows: sense = 'N' or 'n': None. sense = 'E' or 'e': Compute rconde only. sense = 'V' or 'v': Compute rcondv only. sense = 'B' or 'b': Compute both rconde and rcondv . If sense = 'E' or 'e' or 'B' or 'b' , both the left and right eigenvectors must also be computed, that is, jobvl = 'V' or 'v' and jobvr = 'V' or 'v' .
	n	The order of the matrix A . $n \geq 0$.
	a	The n -by- n matrix whose eigenvalues and eigenvectors are desired.
	lda	The leading dimension of array a in the calling program unit. $lda \geq \max(1, n)$.
	ldvl	The leading dimension of array vl in the calling program unit. $ldvl \geq 1$, and if jobvl = 'V' or 'v' , then $ldvl \geq n$.
	ldvr	The leading dimension of array vr in the calling program unit. $ldvr \geq 1$, and if jobvr = 'V' or 'v' , then $ldvr \geq n$.
	lwork	The length of array work . For SGEEVX and DGEEVX, if sense = 'N' or 'n' or 'E' or 'e' , $lwork \geq \max(1, 2n)$, and if jobvl = 'V' or 'v' or jobvr = 'V' or 'v' , $lwork \geq \max(1, 3n)$. If sense = 'V' or 'v' or 'B' or 'b' , $lwork \geq n \times (n+6)$. For CGEEVX and ZGEEVX, if sense = 'N' or 'n' or 'E' or 'e' , $lwork \geq \max(1, 2n)$. If sense = 'V' or 'v' or 'B' or 'b' , $lwork \geq n \times (n+2)$. For good performance, lwork must generally be larger. The optimum value of lwork for high performance is returned in work(1) .
Working Storage	work, iwork, rwork	Arrays used for work space. On successful exit, work(1) contains the optimal work space length lwork for high performance. iwork is not referenced if sense = 'N' or 'n' or 'E' or 'e' .
Output	a	Destroyed.
	wr, wi	On successful exit from SGEEVX and DGEEVX, wr(j) and wi(j) contain the real and imaginary parts, respectively, of the computed eigenvalue λ_j , $j = 1, 2, \dots, n$. The eigenvalues are not in any particular order, except that complex conjugate pairs of eigenvalues appear consecutively with the eigenvalue having the positive imaginary part first.
	w	On successful exit from CGEEVX and ZGEEVX, w(j) contains the computed eigenvalue λ_j , $j = 1, 2, \dots, n$. The eigenvalues are not in any particular order.

- vl** On successful exit, if **jobvl** = 'V' or 'v', the left eigenvectors, y_j , $j = 1, 2, \dots, n$, stored one after another in the columns of **vl**, in the same order as the eigenvalues. The eigenvectors are normalized to have Euclidean norm equal to 1, and to have the largest component real.

In SGEEVX and DGEEVX, a left eigenvector corresponding to a real eigenvalue is real and takes up one column. An eigenvector pair corresponding to a complex conjugate pair of eigenvalues is complex and takes up two columns: the first column holds the real part and the second column holds the imaginary part of the eigenvector corresponding to the eigenvalue with positive imaginary part.

Therefore, if $wl(j) > 0$, the left eigenvector corresponding to $wr(j) + iwl(j)$ is $vl(j) + i vl(j+1)$, and if $wl(j) < 0$, the left eigenvector corresponding to $wr(j) + iwl(j)$ is $vl(j-1) - i vl(j)$, where $i = \sqrt{-1}$.

In CGEEVX and ZGEEVX, each left eigenvector takes up one column.

Not referenced if **jobvl** = 'N' or 'n'.

- vr** On successful exit, if **jobvr** = 'V' or 'v', the right eigenvectors, z_j , $j = 1, 2, \dots, n$, stored one after another in the columns of **vr**, in the same order as their eigenvalues. The eigenvectors are normalized to have Euclidean norm equal to 1, and to have the largest component real.

In SGEEVX and DGEEVX, a right eigenvector corresponding to a real eigenvalue is real and takes up one column. An eigenvector pair corresponding to a complex conjugate pair of eigenvalues is complex and takes up two columns: the first column holds the real part and the second column holds the imaginary part of the eigenvector corresponding to the eigenvalue with positive imaginary part.

Therefore, if $wl(j) > 0$, the right eigenvector corresponding to $wr(j) + iwl(j)$ is $vr(j) + i vr(j+1)$, and if $wl(j) < 0$, the right eigenvector corresponding to $wr(j) + iwl(j)$ is $vr(j-1) - i vr(j)$, where $i = \sqrt{-1}$.

In CGEEVX and ZGEEVX, each right eigenvector takes up one column.

Not referenced if **jobvr** = 'N' or 'n'.

- ilo, ihi, scale** On successful exit, information about the permutations and diagonal scaling factors used in balancing. The permutations consist of row and column interchanges which put the matrix in the form

$$PAP = \begin{bmatrix} T_{11} & X & Y \\ 0 & B & Z \\ 0 & 0 & T_{33} \end{bmatrix}$$

where T_{11} and T_{33} are upper triangular matrices. The indices **ilo** and **ihi** are the beginning and ending rows and columns of submatrix B . P is a product of transpositions:

$$P = (ilo-1, P_{ilo-1}) \cdots (2, P_2) (1, P_1) (ihi+1, P_{ihi+1}) \cdots (n-1, P_{n-1}) (n, P_n)$$

where (j, P_j) denotes the transposition that interchanges j with P_j .

Scaling consists of applying a diagonal similarity transformation, $D^{-1}BD$ to make the 1-norms of each row of B and its corresponding column nearly equal.

The contents of **scale** is as follows:

$$\text{scale}(j) = \begin{cases} P_j & \text{for } j = 1, 2, \dots, \text{ilo}-1 \\ D_{jj} & \text{for } j = \text{ilo}, \text{ilo}+1, \dots, \text{ihi} \\ P_j & \text{for } j = \text{ihi}+1, \text{ihi}+2, \dots, \text{n}. \end{cases}$$

abnorm On successful exit, the one-norm of the balanced matrix.

rconde On successful exit, **rconde**(j) is the reciprocal condition number of eigenvalue λ_j with respect to the balanced matrix, defined as

$$\text{rconde}(j) = |y_j^* z_j|$$

where y_j and z_j are left and right unit eigenvectors, respectively, corresponding to λ_j . $0 \leq \text{rconde}(j) \leq 1$, with **rconde**(j) ≈ 0 indicating that λ_j is very poorly conditioned, and **rconde**(j) ≈ 1 indicating that λ_j is very well conditioned.

An approximate bound on the error between the computed eigenvalue $\tilde{\lambda}_j$ and the correct eigenvalue λ_j is given by

$$|\tilde{\lambda}_j - \lambda_j| \leq \varepsilon \|A\|_1 / \text{rconde}(j)$$

where ε is the machine precision.

rcondv On successful exit, **rcondv**(j) is an approximation to the reciprocal condition number of the right eigenvector z_j corresponding to λ_j , defined as follows. Suppose T is the Schur Form of A (see SGEESX) with $T_{11} = \lambda_j$:

$$Q^* A Q = \begin{bmatrix} \lambda_j & T_{12} \\ 0 & T_{22} \end{bmatrix}$$

where T_{12} is 1-by-($n-1$) and T_{22} is ($n-1$)-by-($n-1$). Then

$$\text{rcondv}(j) = \sigma_{\min}(T_{22} - \lambda_j I)$$

where σ_{\min} denotes the smallest singular value. The smallest singular value is approximated by the reciprocal of an estimate of $\|(T_{22} - \lambda_j I)^{-1}\|_1$.

When **rcondv**(j) is small, small changes in the matrix can cause large changes in z_j . If **balanc** = 'N' or 'n' or 'P' or 'p', an approximate bound for a computed right eigenvector z_j is given by

$$\theta(\tilde{z}_j, z_j) \leq \varepsilon \text{abnorm} / \text{rcondv}(j)$$

where $\theta(x, y)$ is the angle between vectors x and y , and where ε is the machine precision.

The interpretation of **rcondv**(j) is more complicated when **balanc** = 'S' or 's' or 'B' or 'b'. See (Anderson, *et al.*) for details.

info Status response:

info = 0: Successful exit.
info < 0: If **info** = $-k$, the k -th argument had an invalid value.
info > 0: The *QR* algorithm failed to converge on all the eigenvalues; if **info** = k , elements 1, 2, ..., $ilo-1$ and $k+1, k+2, \dots, n$ of **wr** and **wi** or **w** contain eigenvalues which have converged. No eigenvectors have been computed.

Notes

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

balanc \neq 'N' or 'n' or 'P' or 'p' or 'S' or 's' or 'B' or 'b',
jobvl \neq 'N' or 'n' or 'V' or 'v',
jobvr \neq 'N' or 'n' or 'V' or 'v',
sense \neq 'N' or 'n' or 'E' or 'e' or 'V' or 'v' or 'B' or 'b',
sense = 'E' or 'e' or 'B' or 'b' and **jobvl** \neq 'V' or 'v',
sense = 'E' or 'e' or 'B' or 'b' and **jobvr** \neq 'V' or 'v',
n < 0,
lda < max(1,n),
ldvl too small,
ldvr too small, and
lwork too small.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding, for example, the **jobvl** and **jobvr** arguments as 'NoVectors' for 'N' or 'Vectors' for 'V'.

Symmetric or Hermitian Band Matrix**SSBEVX/DSBEVX/.../ZHBEVX**

Purpose These subprograms compute selected eigenvalues and, optionally, corresponding eigenvectors of a real symmetric or complex Hermitian matrix band matrix. The desired eigenvalues can be selected by specifying either a range of values or a range of indices.

A matrix is symmetric if $A = A^T$, its transpose; a matrix is Hermitian if $A = A^*$, its conjugate transpose.

Given such a matrix A , the eigenvalues of A are scalars λ_i , $i = 1, 2, \dots, n$, for which there exist corresponding nonzero vectors, z_i , $i = 1, 2, \dots, n$, such that

$$Az_i = \lambda_i z_i.$$

Matrix Storage Because it is not necessary to store or operate on the zeros outside the band of A , and since either triangle of A may be obtained from the other, you need only provide the band within one triangle of A . Compared to storing the entire matrix, this can save memory in two ways: only the elements within the band are stored, and of them, only the upper or the lower triangle.

The following examples illustrate the storage of real symmetric band matrices. Consider the following matrix A of order $n = 7$ and half bandwidth $kd = 2$:

11	12	13	0	0	0	0
12	22	23	24	0	0	0
13	23	33	34	35	0	0
0	24	34	44	45	46	0
0	0	35	45	55	56	57
0	0	0	46	56	66	67
0	0	0	0	57	67	77

Upper triangular storage. The upper triangle of the matrix A is stored in an array \mathbf{ab} with at least $kd+1 = 3$ rows and 7 columns as follows:

*	*	13	24	35	46	57
*	12	23	34	45	56	67
11	22	33	44	55	66	77

The asterisks represent elements in the kd -by- kd triangle at the upper left corner of \mathbf{ab} that are not referenced. Thus, if a_{ij} is an element within the band of the upper triangle of A , it is stored in $\mathbf{ab}(kd+1+i-j, j)$. Therefore, the columns of the upper triangle of A are stored in the columns of \mathbf{ab} , and the diagonals of the upper triangle of A are stored in the rows of \mathbf{ab} .

Lower triangular storage. The lower triangle of A is stored in the array \mathbf{ab} as follows:

11	22	33	44	55	66	77
12	23	34	45	56	67	*
13	24	35	46	57	*	*

The asterisks represent elements in the kd -by- kd triangle at the lower right corner of \mathbf{ab} that are not referenced. Thus, if a_{ij} is an element within the band of the lower triangle of A , it is stored in $\mathbf{ab}(1+i-j, j)$. Therefore, the columns of the lower triangle of A are stored in the columns of \mathbf{ab} , and the diagonals of the lower triangle of A are stored in the rows of \mathbf{ab} .

Usage

LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

CHARACTER*1 jobz, range, uplo
INTEGER*4 il, info, iu, kd, ldab, ldq, ldz, m, n
REAL*4 abstol, vl, vu
INTEGER*4 ifail(n), iwork(5*n)
REAL*4 ab(ldab, n), q(ldq, n), w(n), work(6*n), z(ldz, n)
CALL SSBEVX (jobz, range, uplo, n, kd, ab, ldab, q, ldq, vl, vu,
 il, iu, abstol, m, w, z, ldz, work, iwork, ifail, info)

CHARACTER*1 jobz, range, uplo
INTEGER*4 il, info, iu, kd, ldab, ldq, ldz, m, n
REAL*8 abstol, vl, vu
INTEGER*4 ifail(n), iwork(5*n)
REAL*8 ab(ldab, n), q(ldq, n), w(n), work(6*n), z(ldz, n)
CALL DSBEVX (jobz, range, uplo, n, kd, ab, ldab, q, ldq, vl, vu,
 il, iu, abstol, m, w, z, ldz, work, iwork, ifail, info)

CHARACTER*1 jobz, range, uplo
INTEGER*4 il, info, iu, kd, ldab, ldq, ldz, m, n
REAL*4 abstol, vl, vu
INTEGER*4 ifail(n), iwork(5*n)
REAL*4 rwork(6*n), w(n)
COMPLEX*8 ab(ldab, n), q(ldq, n), work(n), z(ldz, n)
CALL 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*1 jobz, range, uplo
INTEGER*4 il, info, iu, kd, ldab, ldq, ldz, m, n
REAL*8 abstol, vl, vu
INTEGER*4 ifail(n), iwork(5*n)
REAL*8 rwork(6*n), w(n)
COMPLEX*16 ab(ldab, n), q(ldq, n), work(n), z(ldz, n)
CALL ZHBEVX (jobz, range, uplo, n, kd, ab, ldab, q, ldq, vl, vu,
 il, iu, abstol, m, w, z, ldz, work, rwork, iwork,
 ifail, info)

LAPACK8, available on C Series and Exemplar architectures:

CHARACTER*1 jobz, range, uplo
INTEGER*8 il, info, iu, kd, ldab, ldq, ldz, m, n
REAL*8 abstol, vl, vu
INTEGER*8 ifail(n), iwork(5*n)
REAL*8 ab(ldab, n), q(ldq, n), w(n), work(6*n), z(ldz, n)
CALL SSBEVX (jobz, range, uplo, n, kd, ab, ldab, q, ldq, vl, vu,
 il, iu, abstol, m, w, z, ldz, work, iwork, ifail, info)

CHARACTER*1 jobz, range, uplo
INTEGER*8 il, info, iu, kd, ldab, ldq, ldz, m, n
REAL*8 abstol, vl, vu
INTEGER*8 ifail(n), iwork(5*n)
REAL*8 rwork(6*n), w(n)
COMPLEX*16 ab(ldab, n), q(ldq, n), work(n), z(ldz, n)
CALL CHBEVX (jobz, range, uplo, n, kd, ab, ldab, q, ldq, vl, vu,
 il, iu, abstol, m, w, z, ldz, work, rwork, iwork,
 ifail, info)

Continued

Input	jobz	<p>Specifies whether eigenvectors are to be computed, as follows:</p> <p>jobz = 'N' or 'n': Compute eigenvalues only. jobz = 'V' or 'v': Compute eigenvectors as well.</p>
	range	<p>Specifies which eigenvalues are to be computed, as follows:</p> <p>range = 'A' or 'a': All eigenvalues are to be computed. range = 'V' or 'v': Eigenvalues λ with $vl < \lambda \leq vu$ are to be computed. range = 'I' or 'i': Eigenvalues λ_{il} through λ_{iu} are to be computed.</p>
	uplo	<p>Specifies whether the upper or lower triangular part of the symmetric or Hermitian matrix A is stored, as follows:</p> <p>uplo = 'U' or 'u': The upper triangular part of A is stored. uplo = 'L' or 'l': The lower triangular part of A is stored.</p>
	n	The order of the matrix A . $n \geq 0$.
	kd	The number of super-diagonals of the matrix A if uplo = 'U' or 'u', or the number of subdiagonals if uplo = 'L' or 'l'. $kd \geq 0$.
	ab	<p>The upper or lower triangle of the symmetric or Hermitian band matrix, stored in the first $kd+1$ rows of array ab. The j-th column of A is stored in the j-th column of array ab as follows:</p> <p>If uplo = 'U' or 'u', $ab(kd+1+i-j, j) = A(i, j)$ for $\max(1, j-kd) \leq i \leq j$; If uplo = 'L' or 'l', $ab(1+i-j, j) = A(i, j)$ for $j \leq i \leq \min(n, j+kd)$.</p>
	ldab	The leading dimension of array ab in the calling program unit. $ldab \geq \max(1, n)$.
	ldq	The leading dimension of array q in the calling program unit. If jobz = 'V' or 'v', then $ldq \geq \max(1, n)$.
	vl	<p>If range = 'V' or 'v', the left endpoint of the interval of selected eigenvalues. Only eigenvalues $\lambda > vl$ are returned. $vl < vu$.</p> <p>Not referenced if range = 'A' or 'a' or 'I' or 'i'.</p>
	vu	<p>If range = 'V' or 'v', the right endpoint of the interval of selected eigenvalues. Only eigenvalues $\lambda \leq vu$ are returned. $vu > vl$.</p> <p>Not referenced if range = 'A' or 'a' or 'I' or 'i'.</p>
	il	<p>If range = 'I' or 'i', the index of the smallest eigenvalue to be computed. If the eigenvalues are ordered so that $\lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$, only eigenvalues λ_i with $i \geq il$ are returned. $il \geq 1$.</p> <p>Not referenced if range = 'A' or 'a' or 'V' or 'v'.</p>
	iu	<p>If range = 'I' or 'i', the index of the largest eigenvalue to be computed. If the eigenvalues are ordered so that $\lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$, only eigenvalues λ_i with $i \leq iu$ are returned. $\min(il, n) \leq iu \leq n$.</p> <p>Not referenced if range = 'A' or 'a' or 'V' or 'v'.</p>

abstol 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

$$b - a \leq \mathbf{abstol} + \varepsilon \max(|a|, |b|),$$

where ε is the machine precision. If $\mathbf{abstol} \leq 0$, then $\varepsilon \|T\|_1$ is used in its place, where T is the matrix obtained by reducing A to tridiagonal form.

Eigenvalues will be computed most accurately when \mathbf{abstol} is set to twice the underflow threshold $2\mathbf{slamch}$ ('Safe minimum'), not zero. If this subprogram returns with $\mathbf{info} > 0$, indicating that some eigenvectors did not converge, try setting \mathbf{abstol} to $2\mathbf{slamch}$ ('Safe minimum').

ldz The leading dimension of array z in the calling program unit. $\mathbf{ldz} \geq \max(1, n)$.

**Working
Storage**

work,
iwork,
rwork Arrays used for work space.

Output

ab Destroyed.

q If $\mathbf{jobz} = 'V'$ or $'v'$, the n -by- n orthogonal or unitary matrix that reduces A to tridiagonal form.

m The total number of selected eigenvalues found. $0 \leq \mathbf{m} \leq n$.

w On successful exit, the first \mathbf{m} elements contain the selected eigenvalues in ascending order.

z On successful exit, if $\mathbf{jobz} = 'V'$ or $'v'$, the first \mathbf{m} columns of z contain the orthonormal eigenvectors of the matrix, corresponding to the selected eigenvalues. 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**.

Not referenced if $\mathbf{jobz} = 'N'$ or $'n'$.

ifail Eigenvector convergence status response. On successful exit, if $\mathbf{jobz} = 'V'$ or $'v'$, the first \mathbf{m} elements are zero. If $\mathbf{info} = k > 0$, then the first k elements of **ifail** contain the indices of the eigenvectors that failed to converge.

Not referenced if $\mathbf{jobz} = 'N'$ or $'n'$.

info Status response:

info = 0: Successful exit.

info < 0: If $\mathbf{info} = -k$, the k -th argument had an invalid value.

info > 0: If $\mathbf{info} = k$, then k eigenvectors failed to converge. Their indices are stored in the first k elements of **ifail**.

Notes

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

jobz ≠ 'N' or 'n' or 'V' or 'v',
range ≠ 'A' or 'a' or 'V' or 'v' or 'I' or 'i',
uplo ≠ 'L' or 'l' or 'U' or 'u',
n < 0,
ldab < **kd**+1,
range = 'V' or 'v' and **vu** ≤ **vl**,
range = 'I' or 'i' and **il** < 1,
range = 'I' or 'i' and **iu** < min(**il**,**n**) or **iu** > **n**, and
ldz < max(1,**n**).

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding, for example, the **jobz** argument as 'NoVectors' for 'N' or 'Vectors' for 'V'.

Purpose These subprograms compute selected eigenvalues and, optionally, corresponding eigenvectors of a real symmetric or complex Hermitian matrix in packed storage. The desired eigenvalues can be selected by specifying either a range of values or a range of indices.

A matrix is symmetric if $A = A^T$, its transpose; a matrix is Hermitian if $A = A^*$, its conjugate transpose.

Given such a matrix A , the eigenvalues of A are scalars λ_i , $i = 1, 2, \dots, n$, for which there exist corresponding nonzero vectors, z_i , $i = 1, 2, \dots, n$, such that

$$Az_i = \lambda_i z_i.$$

Matrix Storage Because either triangle of A may be obtained from the other, you need only provide one triangle of A , either the upper or the lower triangle. Compared to storing the entire matrix, you save memory by supplying that triangle stored column-by-column in packed form in a 1-dimensional array.

The following examples illustrate the packed storage of symmetric or Hermitian matrices.

Upper triangular storage. If the upper triangle of A is

$$\begin{array}{cccc} 11 & 12 & 13 & 14 \\ & 22 & 23 & 24 \\ & & 33 & 34 \\ & & & 44 \end{array}$$

then A is packed column-by-column into an array **ap** as follows:

k	1	2	3	4	5	6	7	8	9	10
ap(k)	11	12	22	13	23	33	14	24	34	44

Upper triangular matrix element a_{ij} is stored in array element **ap**($i+j \times (j-1)/2$).

Lower triangular storage. If the lower triangle of A is

$$\begin{array}{cccc} 11 & & & \\ 21 & 22 & & \\ 31 & 32 & 33 & \\ 41 & 42 & 43 & 44 \end{array}$$

then A is packed column-by-column into an array **ap** as follows:

k	1	2	3	4	5	6	7	8	9	10
ap(k)	11	21	31	41	22	32	42	33	43	44

Lower triangular matrix element a_{ij} is stored in array element **ap**($i+(j-1) \times (2n-j)/2$).

Usage

LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

CHARACTER*1 jobz, range, uplo
 INTEGER*4 il, info, iu, ldz, m, n
 REAL*4 abstol, vl, vu
 INTEGER*4 ifail(n), iwork(5*n)
 REAL*4 ap((n*(n+1))/2), w(n), work(7*n), z(ldz, n)
 CALL SSPEVX (jobz, range, uplo, n, ap, vl, vu, il, iu, abstol, m,
 w, z, ldz, work, iwork, ifail, info)

CHARACTER*1 jobz, range, uplo
 INTEGER*4 il, info, iu, ldz, m, n
 REAL*8 abstol, vl, vu
 INTEGER*4 ifail(n), iwork(5*n)
 REAL*8 ap((n*(n+1))/2), w(n), work(7*n), z(ldz, n)
 CALL DSPEVX (jobz, range, uplo, n, ap, vl, vu, il, iu, abstol, m,
 w, z, ldz, work, iwork, ifail, info)

CHARACTER*1 jobz, range, uplo
 INTEGER*4 il, info, iu, ldz, m, n
 REAL*4 abstol, vl, vu
 INTEGER*4 ifail(n), iwork(5*n)
 REAL*4 rwork(6*n), w(n)
 COMPLEX*8 ap((n*(n+1))/2), work(2*n), z(ldz, n)
 CALL CHPEVX (jobz, range, uplo, n, ap, vl, vu, il, iu, abstol, m,
 w, z, ldz, work, rwork, iwork, ifail, info)

CHARACTER*1 jobz, range, uplo
 INTEGER*4 il, info, iu, ldz, m, n
 REAL*8 abstol, vl, vu
 INTEGER*4 ifail(n), iwork(5*n)
 REAL*8 rwork(6*n), w(n)
 COMPLEX*16 ap((n*(n+1))/2), work(2*n), z(ldz, n)
 CALL ZHPEVX (jobz, range, uplo, n, ap, vl, vu, il, iu, abstol, m,
 w, z, ldz, work, rwork, iwork, ifail, info)

LAPACK8, available on C Series and Exemplar architectures:

CHARACTER*1 jobz, range, uplo
 INTEGER*8 il, info, iu, ldz, m, n
 REAL*8 abstol, vl, vu
 INTEGER*8 ifail(n), iwork(5*n)
 REAL*8 ap((n*(n+1))/2), w(n), work(7*n), z(ldz, n)
 CALL SSPEVX (jobz, range, uplo, n, ap, vl, vu, il, iu, abstol, m,
 w, z, ldz, work, iwork, ifail, info)

CHARACTER*1 jobz, range, uplo
 INTEGER*8 il, info, iu, ldz, m, n
 REAL*8 abstol, vl, vu
 INTEGER*8 ifail(n), iwork(5*n)
 REAL*8 rwork(6*n), w(n)
 COMPLEX*16 ap((n*(n+1))/2), work(2*n), z(ldz, n)
 CALL CHPEVX (jobz, range, uplo, n, ap, vl, vu, il, iu, abstol, m,
 w, z, ldz, work, rwork, iwork, ifail, info)

Input	jobz	Specifies whether eigenvectors are to be computed, as follows: jobz = 'N' or 'n': Compute eigenvalues only. jobz = 'V' or 'v': Compute eigenvectors as well.
	range	Specifies which eigenvalues are to be computed, as follows: range = 'A' or 'a': All eigenvalues are to be computed. range = 'V' or 'v': Eigenvalues λ with $vl < \lambda \leq vu$ are to be computed. range = 'I' or 'i': Eigenvalues λ_{i1} through λ_{iu} are to be computed.
	uplo	Specifies whether the upper or lower triangular part of the symmetric or Hermitian matrix A is stored, as follows: uplo = 'U' or 'u': The upper triangular part of A is stored. uplo = 'L' or 'l': The lower triangular part of A is stored.
	n	The order of the matrix A . $n \geq 0$.
	ap	The upper or lower triangular part of the symmetric or Hermitian matrix A , packed columnwise in a linear array as follows: If uplo = 'U' or 'u', $ap(i + (j-1) \times j/2) = A(i, j)$ for $1 \leq i \leq j$; If uplo = 'L' or 'l', $ap(i + (j-1) \times (2n-j)/2) = A(i, j)$ for $j \leq i \leq n$.
	vl	If range = 'V' or 'v', the left endpoint of the interval of selected eigenvalues. Only eigenvalues $\lambda > vl$ are returned. $vl < vu$. Not referenced if range = 'A' or 'a' or 'I' or 'i'.
	vu	If range = 'V' or 'v', the right endpoint of the interval of selected eigenvalues. Only eigenvalues $\lambda \leq vu$ are returned. $vu > vl$. Not referenced if range = 'A' or 'a' or 'I' or 'i'.
	il	If range = 'I' or 'i', the index of the smallest eigenvalue to be computed. If the eigenvalues are ordered so that $\lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$, only eigenvalues λ_i with $i \geq il$ are returned. $il \geq 1$. Not referenced if range = 'A' or 'a' or 'V' or 'v'.
	iu	If range = 'I' or 'i', the index of the largest eigenvalue to be computed. If the eigenvalues are ordered so that $\lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$, only eigenvalues λ_i with $i \leq iu$ are returned. $\min(il, n) \leq iu \leq n$. Not referenced if range = 'A' or 'a' or 'V' or 'v'.

Continued

	abstol	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 $b - a \leq \text{abstol} + \epsilon \max(a , b),$ where ϵ is the machine precision. If abstol ≤ 0 , then $\epsilon \ T\ _1$ is used in its place, where T is the matrix obtained by reducing A to tridiagonal form. Eigenvalues will be computed most accurately when abstol is set to twice the underflow threshold 2slamch ('Safe minimum'), not zero. If this subprogram returns with info > 0 , indicating that some eigenvectors did not converge, try setting abstol to 2slamch ('Safe minimum').
	ldz	The leading dimension of array z in the calling program unit. ldz $\geq \max(1, n)$.
Working Storage	work, iwork, rwork	Arrays used for work space.
Output	ap	Destroyed.
	m	The total number of selected eigenvalues found. $0 \leq m \leq n$.
	w	On successful exit, the first m elements contain the selected eigenvalues in ascending order.
	z	On successful exit, if jobz = 'V' or 'v', the first m columns of z contain the orthonormal eigenvectors of the matrix, corresponding to the selected eigenvalues. 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 . Not referenced if jobz = 'N' or 'n'.
	ifail	Eigenvector convergence status response. On successful exit, if jobz = 'V' or 'v', the first m elements are zero. If info = $k > 0$, then the first k elements of ifail contain the indices of the eigenvectors that failed to converge. Not referenced if jobz = 'N' or 'n'.
	info	Status response: info = 0: Successful exit. info < 0: If info = $-k$, the k -th argument had an invalid value. info > 0: If info = k , then k eigenvectors failed to converge. Their indices are stored in the first k elements of ifail .

Notes

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

jobz ≠ 'N' or 'n' or 'V' or 'v',
range ≠ 'A' or 'a' or 'V' or 'v' or 'I' or 'i',
uplo ≠ 'L' or 'l' or 'U' or 'u',
n < 0,
range = 'V' or 'v' and **vu** ≤ **vl**,
range = 'I' or 'i' and **il** < 1,
range = 'I' or 'i' and **iu** < min(**il**,**n**) or **iu** > **n**, and
ldz < max(1,**n**).

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding, for example, the **jobz** argument as 'NoVectors' for 'N' or 'Vectors' for 'V'.

Real Symmetric Tridiagonal Matrix**SSTEVMX/DSTEVMX**

Purpose These subprograms compute selected eigenvalues and, optionally, corresponding eigenvectors of a real symmetric tridiagonal matrix. The desired eigenvalues can be selected by specifying either a range of values or a range of indices.

A matrix is symmetric if $A = A^T$, its transpose.

A matrix $A = (a_{ij})$ is tridiagonal if its nonzero elements lie only on the principal diagonal ($i = j$), the subdiagonal ($i = j+1$), and the superdiagonal ($i = j-1$) of the matrix.

Given such a matrix A , the eigenvalues of A are scalars λ_i , $i = 1, 2, \dots, n$, for which there exist corresponding nonzero vectors, z_i , $i = 1, 2, \dots, n$, such that

$$Az_i = \lambda_i z_i.$$

Matrix Storage The following example illustrates the storage of symmetric tridiagonal matrices. Consider the following symmetric tridiagonal matrix of order $n = 7$:

11	21	0	0	0	0	0
21	22	32	0	0	0	0
0	32	33	43	0	0	0
0	0	43	44	54	0	0
0	0	0	54	55	65	0
0	0	0	0	65	66	76
0	0	0	0	0	76	77

The subdiagonal is stored in array **e** and the principal diagonal is stored in array **d**, as follows:

<i>i</i>	e (<i>i</i>)	d (<i>i</i>)
1	21	11
2	32	22
3	43	33
4	54	44
5	65	55
6	76	66
7		77

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```

CHARACTER*1 jobz, range
INTEGER*4    il, info, iu, ldz, m, n
REAL*4      abstol, vl, vu
INTEGER*4    ifail(n), iwork(5*n)
REAL*4      d(n), e(n-1), w(n), work(4*n), z(ldz, n)
CALL SSTEVMX (jobz, range, n, d, e, vl, vu, il, iu, abstol, m, w,
               z, ldz, work, iwork, ifail, info)

```

```

CHARACTER*1 jobz, range
INTEGER*4    il, info, iu, ldz, m, n
REAL*8      abstol, vl, vu
INTEGER*4    ifail(n), iwork(5*n)
REAL*8      d(n), e(n-1), w(n), work(4*n), z(ldz, n)
CALL DSTEVMX (jobz, range, n, d, e, vl, vu, il, iu, abstol, m, w,
               z, ldz, work, iwork, ifail, info)

```

LAPACK8, available on C Series and Exemplar architectures:

```

CHARACTER*1  jobz, range
INTEGER*8    il, info, iu, ldz, m, n
REAL*8       abstol, vl, vu
INTEGER*8    ifail(n), iwork(5*n)
REAL*8       d(n), e(n-1), w(n), work(4*n), z(ldz, n)
CALL SSTEVMX (jobz, range, n, d, e, vl, vu, il, iu, abstol, m, w,
               z, ldz, work, iwork, ifail, info)

```

- Input**
- jobz** Specifies whether eigenvectors are to be computed, as follows:
- jobz** = 'N' or 'n': Compute eigenvalues only.
jobz = 'V' or 'v': Compute eigenvectors as well.
- range** Specifies which eigenvalues are to be computed, as follows:
- range** = 'A' or 'a': All eigenvalues are to be computed.
range = 'V' or 'v': Eigenvalues λ with $vl < \lambda \leq vu$ are to be computed.
range = 'I' or 'i': Eigenvalues λ_{i_1} through λ_{i_u} are to be computed.
- n** The order in the matrix A . $n \geq 0$.
- d** The n diagonal elements of the tridiagonal matrix.
- e** The $n-1$ subdiagonal elements of the tridiagonal matrix.
- vl** If **range** = 'V' or 'v', the left endpoint of the interval of selected eigenvalues. Only eigenvalues $\lambda > vl$ are returned. $vl < vu$.
 Not referenced if **range** = 'A' or 'a' or 'I' or 'i'.
- vu** If **range** = 'V' or 'v', the right endpoint of the interval of selected eigenvalues. Only eigenvalues $\lambda \leq vu$ are returned. $vu > vl$.
 Not referenced if **range** = 'A' or 'a' or 'I' or 'i'.
- il** If **range** = 'I' or 'i', the index of the smallest eigenvalue to be computed. If the eigenvalues are ordered so that $\lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$, only eigenvalues λ_i with $i \geq il$ are returned. $il \geq 1$.
 Not referenced if **range** = 'A' or 'a' or 'V' or 'v'.
- iu** If **range** = 'I' or 'i', the index of the largest eigenvalue to be computed. If the eigenvalues are ordered so that $\lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$, only eigenvalues λ_i with $i \leq iu$ are returned. $\min(il, n) \leq iu \leq n$.
- abstol** 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

$$b - a \leq \text{abstol} + \epsilon \max(|a|, |b|),$$

where ϵ is the machine precision. If **abstol** ≤ 0 , then $\epsilon \|A\|_1$ is used in its place.

Eigenvalues will be computed most accurately when **abstol** is set to twice the underflow threshold `2slamch('Safe minimum')`, not zero. If this subprogram returns with **info** > 0 , indicating that some eigenvectors did not converge, try

Continued

		setting abstol to 2slamch ('Safe minimum').
	ldz	The leading dimension of array z in the calling program unit. $ldz \geq 1$, and if jobz = 'V' or 'v', then $ldz \geq n$.
Working Storage	work , iwork	Arrays used for work space.
Output	d	Destroyed.
	e	Destroyed.
	m	The total number of selected eigenvalues found. $0 \leq m \leq n$.
	w	On successful exit, the first m elements contain the selected eigenvalues in ascending order.
	z	On successful exit, if jobz = 'V' or 'v', the orthonormal eigenvectors of the matrix. If an error exit is made, z contains the eigenvectors associated with the stored eigenvalues.
		Not referenced if jobz = 'N' or 'n'.
	ifail	Eigenvector convergence status response. On successful exit, if jobz = 'V' or 'v', the first m elements are zero. If info = $k > 0$, then the first k elements of ifail contain the indices of the eigenvectors that failed to converge.
		Not referenced if jobz = 'N' or 'n'.
	info	Status response: info = 0: Successful exit. info < 0: If info = $-k$, the k -th argument had an invalid value. info > 0: If info = k , then k eigenvectors failed to converge. Their indices are stored in the first k elements of ifail .

Notes

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

jobz ≠ 'N' or 'n' or 'V' or 'v',
range ≠ 'A' or 'a' or 'V' or 'v' or 'I' or 'i',
n < 0,
range = 'V' or 'v' and **vu** ≤ **vl**,
range = 'I' or 'i' and **il** < 1,
range = 'I' or 'i' and **iu** < min(**il**,**n**) or **iu** > **n**,
ldz too small.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding, for example, the **jobz** argument as 'NoVectors' for 'N' or 'Vectors' for 'V'.

Purpose These subprograms compute selected eigenvalues and, optionally, corresponding eigenvectors of a real symmetric or complex Hermitian matrix. The desired eigenvalues can be selected by specifying either a range of values or a range of indices.

A matrix is symmetric if $A = A^T$, its transpose; a matrix is Hermitian if $A = A^*$, its conjugate transpose.

Given such a matrix A , the eigenvalues of A are scalars λ_i , $i = 1, 2, \dots, n$, for which there exist corresponding nonzero vectors, z_i , $i = 1, 2, \dots, n$, such that

$$Az_i = \lambda_i z_i.$$

Matrix Storage Because either triangle of A may be obtained from the other, you need only provide one triangle of A . You may supply either the upper or the lower triangle of A , in a two-dimensional array large enough to hold the entire matrix. The other triangle of the array is not referenced.

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```

CHARACTER*1 jobz, range, uplo
INTEGER*4   il, info, iu, lda, ldz, lwork, m, n
REAL*4     abstol, vl, vu
INTEGER*4   ifail(n), iwork(5*n)
REAL*4     a(lda, n), w(n), work(lwork), z(ldz, n)
CALL SSYEVX (jobz, range, uplo, n, a, lda, vl, vu, il, iu, abstol, m,
              w, z, ldz, work, lwork, iwork, ifail, info)

```

```

CHARACTER*1 jobz, range, uplo
INTEGER*4   il, info, iu, lda, ldz, lwork, m, n
REAL*8     abstol, vl, vu
INTEGER*4   ifail(n), iwork(5*n)
REAL*8     a(lda, n), w(n), work(lwork), z(ldz, n)
CALL DSYEVX (jobz, range, uplo, n, a, lda, vl, vu, il, iu, abstol, m,
              w, z, ldz, work, lwork, iwork, ifail, info)

```

```

CHARACTER*1 jobz, range, uplo
INTEGER*4   il, info, iu, lda, ldz, lwork, m, n
REAL*4     abstol, vl, vu
INTEGER*4   ifail(n), iwork(5*n)
REAL*4     rwork(6*n), w(n)
COMPLEX*8  a(lda, n), work(lwork), z(ldz, n)
CALL CHEEVX (jobz, range, uplo, n, a, lda, vl, vu, il, iu, abstol, m,
              w, z, ldz, work, lwork, rwork, iwork, ifail, info)

```

```

CHARACTER*1 jobz, range, uplo
INTEGER*4   il, info, iu, lda, ldz, lwork, m, n
REAL*8     abstol, vl, vu
INTEGER*4   ifail(n), iwork(5*n)
REAL*8     rwork(6*n), w(n)
COMPLEX*16 a(lda, n), work(lwork), z(ldz, n)
CALL ZHEEVX (jobz, range, uplo, n, a, lda, vl, vu, il, iu, abstol, m,
              w, z, ldz, work, lwork, rwork, iwork, ifail, info)

```

Continued

LAPACK8, available on C Series and Exemplar architectures:

```

CHARACTER*1 jobz, range, uplo
INTEGER*8    il, info, iu, lda, ldz, lwork, m, n
REAL*8      abstol, vl, vu
INTEGER*8    ifail(n), iwork(5*n)
REAL*8      a(lda, n), w(n), work(lwork), z(ldz, n)
CALL SSYEVX (jobz, range, uplo, n, a, lda, vl, vu, il, iu, abstol, m,
              w, z, ldz, work, lwork, iwork, ifail, info)
    
```

```

CHARACTER*1 jobz, range, uplo
INTEGER*8    il, info, iu, lda, ldz, lwork, m, n
REAL*8      abstol, vl, vu
INTEGER*8    ifail(n), iwork(5*n)
REAL*8      rwork(6*n), w(n)
COMPLEX*16  a(lda, n), work(lwork), z(ldz, n)
CALL CHEEVX (jobz, range, uplo, n, a, lda, vl, vu, il, iu, abstol, m,
              w, z, ldz, work, lwork, rwork, iwork, ifail, info)
    
```

Input	jobz	Specifies whether eigenvectors are to be computed, as follows: jobz = 'N' or 'n': Compute eigenvalues only. jobz = 'V' or 'v': Compute eigenvectors as well.
	range	Specifies which eigenvalues are to be computed, as follows: range = 'A' or 'a': All eigenvalues are to be computed. range = 'V' or 'v': Eigenvalues λ with $vl < \lambda \leq vu$ are to be computed. range = 'I' or 'i': Eigenvalues λ_{i1} through λ_{in} are to be computed.
	uplo	Specifies whether the upper or lower triangular part of the symmetric or Hermitian matrix <i>A</i> is stored, as follows: uplo = 'U' or 'u': The upper triangular part of <i>A</i> is stored. uplo = 'L' or 'l': The lower triangular part of <i>A</i> is stored.
	n	The order of the matrix <i>A</i> . $n \geq 0$.
	a	The symmetric or Hermitian matrix <i>A</i> . If uplo = 'U' or 'u' , only the upper triangular part of a is used to define the elements of the matrix and the strict lower triangular part of a is not used for input. If uplo = 'L' or 'l' , only the lower triangular part of a is used to define the elements of the matrix and the strict upper triangular part of a is not used for input.
	lda	The leading dimension of array a in the calling program unit. $lda \geq \max(1, n)$.
	vl	If range = 'V' or 'v' , the left endpoint of the interval of selected eigenvalues. Only eigenvalues $\lambda > vl$ are returned. $vl < vu$. Not referenced if range = 'A' or 'a' or 'I' or 'i' .

	vu	If range = 'V' or 'v', the right endpoint of the interval of selected eigenvalues. Only eigenvalues $\lambda \leq \mathbf{vu}$ are returned. $\mathbf{vu} > \mathbf{vl}$. Not referenced if range = 'A' or 'a' or 'I' or 'i'.
	il	If range = 'I' or 'i', the index of the smallest eigenvalue to be computed. If the eigenvalues are ordered so that $\lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$, only eigenvalues λ_i with $i \geq \mathbf{il}$ are returned. $\mathbf{il} \geq 1$. Not referenced if range = 'A' or 'a' or 'V' or 'v'.
	iu	If range = 'I' or 'i', the index of the largest eigenvalue to be computed. If the eigenvalues are ordered so that $\lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$, only eigenvalues λ_i with $i \leq \mathbf{iu}$ are returned. $\min(\mathbf{il}, \mathbf{n}) \leq \mathbf{iu} \leq \mathbf{n}$. Not referenced if range = 'A' or 'a' or 'V' or 'v'.
	abstol	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 $b - a \leq \mathbf{abstol} + \epsilon \max(a , b),$ where ϵ is the machine precision. If $\mathbf{abstol} \leq 0$, then $\epsilon \ T\ _1$ is used in its place, where T is the matrix obtained by reducing A to tridiagonal form. Eigenvalues will be computed most accurately when abstol is set to twice the underflow threshold 2slamch ('Safe minimum'), not zero. If this subprogram returns with info > 0, indicating that some eigenvectors did not converge, try setting abstol to 2slamch ('Safe minimum').
	ldz	The leading dimension of array z in the calling program unit. $\mathbf{ldz} \geq \max(1, \mathbf{n})$.
	lwork	The length of array work . For subprograms SSYEVS and DSYEVS, $\mathbf{lwork} \geq \max(1, 7\mathbf{n})$. For subprograms CHEEVX and ZHEEVX, $\mathbf{lwork} \geq \max(1, 2\mathbf{n}-1)$. For good performance, lwork must generally be larger. The optimum value of lwork for high performance is returned in work (1).
Working Storage	work, iwork, rwork	Arrays used for work space. On successful exit, work (1) contains the optimal work space length lwork for high performance.
Output	a	The triangle of a specified by uplo , including the diagonal, has been destroyed.
	m	The total number of selected eigenvalues found. $0 \leq \mathbf{m} \leq \mathbf{n}$.
	w	On successful exit, the first m elements contain the selected eigenvalues in ascending order.
	z	On successful exit, if jobz = 'V' or 'v', the first m columns of z contain the orthonormal eigenvectors of the matrix, corresponding to the selected eigenvalues. 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 . Not referenced if jobz = 'N' or 'n'.

ifail Eigenvector convergence status response. On successful exit, if **jobz** = 'V' or 'v', the first **m** elements are zero. If **info** = $k > 0$, then the first k elements of **ifail** contain the indices of the eigenvectors that failed to converge.

Not referenced if **jobz** = 'N' or 'n'.

info Status response:

info = 0: Successful exit.

info < 0: If **info** = $-k$, the k -th argument had an invalid value.

info > 0: If **info** = k , then k eigenvectors failed to converge. Their indices are stored in the first k elements of **ifail**.

Notes

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

jobz ≠ 'N' or 'n' or 'V' or 'v',
range ≠ 'A' or 'a' or 'V' or 'v' or 'I' or 'i',
uplo ≠ 'L' or 'l' or 'U' or 'u',
n < 0,
lda < max(1,n),
range = 'V' or 'v' and **vu** ≤ **vl**,
range = 'I' or 'i' and **il** < 1,
range = 'I' or 'i' and **iu** < min(**il**,n) or **iu** > n,
ldz < max(1,n), and
lwork too small.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding, for example, the **jobz** argument as 'NoVectors' for 'N' or 'Vectors' for 'V'.

Drivers for Generalized Eigenvalue Problems

Overview

This chapter explains how to use LAPACK subprograms to compute the eigenvalues or eigenvalues and eigenvectors of

- generalized symmetric or Hermitian definite eigenproblems of the forms

$$Ax = \lambda Bx \quad \text{or} \quad ABx = \lambda x \quad \text{or} \quad BAx = \lambda x$$

for real symmetric or complex Hermitian matrices A and B , with B positive definite

- generalized symmetric or Hermitian definite banded eigenproblems of the form

$$Ax = \lambda Bx$$

for real symmetric or complex Hermitian band matrices A and B , with B positive definite

- generalized general eigenproblems of the form

$$Ax = \lambda Bx$$

for real or complex general matrices A and B

Refer to Chapters 7 and 8 for software to compute the eigenvalues or eigenvectors and eigenvectors of a real symmetric or complex Hermitian ordinary eigenproblem.

Refer to Chapter 7 of the *ConvexMLIB User's Guide: VECLIB* for software to compute the eigenvalues or eigenvalues and eigenvectors of a real symmetric, sparse, ordinary or generalized eigenproblem.

Chapter Objectives

After reading this chapter you will:

- understand the forms of the generalized eigenproblems solved by LAPACK
- know how to use the described subprograms

What You Need to Know to Use These Subprograms

To use these subprograms, it is helpful to know some basic theory of generalized eigensystems. A few facts discussed in basic textbooks are given here.

If A and B are n -by- n matrices, the set of all matrices of the form $A - \lambda B$ with $\lambda \in \mathbf{C}$ is called a *matrix pencil*. If λ is such that $\det(A - \lambda B) = 0$ then λ is called an *eigenvalue* of the pencil. If λ is an eigenvalue and $Ax = \lambda Bx$ with $x \neq 0$, then x is called an *eigenvector* belonging to λ .

Alternatively, the definitions can be made without resorting to the determinant of the matrix pencil, as follows: if λ is a scalar for which there exists a nonzero vector x such that $Ax = \lambda Bx$, then λ is called an *eigenvalue* and x is called an *eigenvector* belonging to λ .

There are exactly n eigenvalues, counting multiplicity, if and only if $\text{rank}(B) = n$. If B is rank deficient, there may be zero, fewer than n , or an infinite number of eigenvalues. If A and B are real symmetric or complex Hermitian matrices and B is positive definite, then n eigenvalues exist, they are real, and the problem can be reduced to an ordinary symmetric or Hermitian eigenvalue problem as follows:

- Compute the Cholesky factorization, $B = LL^*$.
- Set $C = L^{-1}AL^{-*}$, where L^{-*} is the conjugate transpose of L^{-1} .
- Solve the ordinary eigenvalue problem $Cy_i = \lambda_i y_i$ for λ_i and y_i .
- Set $x_i = L^{-*}y_i$ for $i = 1, 2, \dots, n$.

Then λ_i is an eigenvalue and x_i is an eigenvector of the generalized eigenproblem $Ax = \lambda Bx$. The eigenvectors are B -orthogonal: $x_i^* B x_i = 1$ and $x_i^* B x_j = 0$ if $i \neq j$.

Similar transformations reduce the other forms of generalized eigenvalue problems, $ABx = \lambda x$ and $BAx = \lambda x$, to ordinary symmetric or Hermitian eigenproblems when A and B are real symmetric or complex Hermitian matrices and B is positive definite.

If B is invertible, the general generalized eigenproblem can be reduced to a general ordinary eigenproblem by noting that $B^{-1}Ax = \lambda x$ has the same eigenvalues and eigenvectors as the original problem. This approach will not produce generalized eigenvalues accurately if B is ill-conditioned, so LAPACK uses a robust alternative approach that avoids dealing with B^{-1} .

Supplemental Reading

Golub, G.H. and C.F. Van Loan. *Matrix Computations*, 2nd Edition. Baltimore, MD: The Johns Hopkins University Press. 1989.

Parlett, B.N. *The Symmetric Eigenproblem*. Englewood Cliffs, NJ: Prentice-Hall, Inc. 1980.

Wilkinson, J.H. *The Algebraic Eigenproblem*. New York, NY: Oxford University Press. 1965.

Subprogram Descriptions

Eigenvalues and Generalized Schur Form with General Matrices SGEYS, DGEYS, CGEYS, ZGEYS	9-3
Eigenvalues and Eigenvectors with General Matrices SGEGV, DGEGV, CGEGV, ZGEGV	9-7
Eigenvalues and Eigenvectors with Symmetric or Hermitian Band Matrices SSBGV, DSBGV, CHBGV, ZHBGV	9-11
Eigenvalues and Eigenvectors with Symmetric or Hermitian Packed Matrices SSPGV, DSPGV, CHPGV, ZHPGV	9-15
Eigenvalues and Eigenvectors with Symmetric or Hermitian Packed Matrices SSYGV, DSYGV, CHEGV, ZHEGV	9-19

Generalized Schur Form**SGEGS/DGEGS/CGEGS/ZGEGS****Purpose**

These subprograms compute the eigenvalues and the Schur Form of a generalized eigenproblem of the form $Az = \lambda Bz$, where A and B are n -by- n real or complex general matrices. Optionally, the left and/or right generalized Schur vectors of A and B also are computed. If you need only the generalized eigenvalues, use SGEGV, DGEGV, CGEGV, or ZGEGV, documented elsewhere in this chapter, instead of one of these subprograms.

Given such matrices A and B , the generalized eigenvalues are scalars λ_i , $i = 1, 2, \dots, n$, for which there exist corresponding nonzero vectors, z_i , $i = 1, 2, \dots, n$, such that

$$Az_i = \lambda_i Bz_i.$$

λ_i is usually represented as a pair, (α_i, β_i) , such that $\lambda_i = \alpha_i/\beta_i$ if the ratio is defined. There is a reasonable interpretation for $\beta_i = 0$ and even for $\alpha_i = \beta_i = 0$. See (Golub and Van Loan) for details.

A pair of real matrices T and S is in generalized real Schur Form if T is block upper triangular with 1-by-1 and 2-by-2 diagonal blocks and S is upper triangular with non-negative diagonal elements. 2-by-2 blocks diagonal blocks of T are standardized so that the corresponding diagonal blocks of S have the form

$$\begin{bmatrix} a & 0 \\ 0 & c \end{bmatrix}.$$

1-by-1 blocks correspond to real generalized eigenvalues, while 2-by-2 blocks correspond to complex conjugate generalized eigenpairs.

A pair of complex matrices T and S is in complex Schur Form if T is upper triangular and S is upper triangular with non-negative real diagonal elements.

If T and S are the generalized Schur Form of A and B , then the columns of the orthogonal or unitary matrices Q and Z such that

$$T = Q^*AZ \quad \text{and} \quad S = Q^*BZ$$

are known as the left and right generalized Schur vectors of A and B , respectively.

Usage

LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```

CHARACTER*1 jobvsl, jobvsr
INTEGER*4 info, lda, ldb, ldvsl, ldvsr, lwork, n
REAL*4 a(lda, n), alphas(n), alphasr(n), b(ldb, n), beta(n),
vsl(ldvsl, n), vsr(ldvsr, n), work(lwork)
CALL SGEGS (jobvsl, jobvsr, n, a, lda, b, ldb, alphas, alphasr, beta,
vsl, ldvsl, vsr, ldvsr, work, lwork, info)

```

```

CHARACTER*1 jobvsl, jobvsr
INTEGER*4 info, lda, ldb, ldvsl, ldvsr, lwork, n
REAL*8 a(lda, n), alphas(n), alphasr(n), b(ldb, n), beta(n),
vsl(ldvsl, n), vsr(ldvsr, n), work(lwork)
CALL DGEGS (jobvsl, jobvsr, n, a, lda, b, ldb, alphas, alphasr, beta,
vsl, ldvsl, vsr, ldvsr, work, lwork, info)

```

```

CHARACTER*1 jobvsl, jobvsr
INTEGER*4   info, lda, ldb, ldvsl, ldvsr, lwork, n
REAL*4     rwork(3*n)
COMPLEX*8  a(lda, n), alpha(n), b(ldb, n), beta(n),
           vsl(ldvsl, n), vsr(ldvsr, n), work(lwork)
CALL CGEGS (jobvsl, jobvsr, n, a, lda, b, ldb, alpha, beta,
           vsl, ldvsl, vsr, ldvsr, work, lwork, rwork, info)

```

```

CHARACTER*1 jobvsl, jobvsr
INTEGER*4   info, lda, ldb, ldvsl, ldvsr, lwork, n
REAL*8     rwork(3*n)
COMPLEX*16 a(lda, n), alpha(n), b(ldb, n), beta(n),
           vsl(ldvsl, n), vsr(ldvsr, n), work(lwork)
CALL ZGEGS (jobvsl, jobvsr, n, a, lda, b, ldb, alpha, beta,
           vsl, ldvsl, vsr, ldvsr, work, lwork, rwork, info)

```

LAPACK8, available on C Series and Exemplar architectures:

```

CHARACTER*1 jobvsl, jobvsr
INTEGER*8   info, lda, ldb, ldvsl, ldvsr, lwork, n
REAL*8     a(lda, n), alphas(n), alphasr(n), b(ldb, n), beta(n),
           vsl(ldvsl, n), vsr(ldvsr, n), work(lwork)
CALL SGEGS (jobvsl, jobvsr, n, a, lda, b, ldb, alphas, alphasr, beta,
           vsl, ldvsl, vsr, ldvsr, work, lwork, info)

```

```

CHARACTER*1 jobvsl, jobvsr
INTEGER*8   info, lda, ldb, ldvsl, ldvsr, lwork, n
REAL*8     rwork(3*n)
COMPLEX*16 a(lda, n), alpha(n), b(ldb, n), beta(n),
           vsl(ldvsl, n), vsr(ldvsr, n), work(lwork)
CALL CGEGS (jobvsl, jobvsr, n, a, lda, b, ldb, alpha, beta,
           vsl, ldvsl, vsr, ldvsr, work, lwork, rwork, info)

```

- Input**
- jobvsl** Specifies whether the left generalized Schur vectors are to be computed, as follows:
- jobvsl** = 'N' or 'n': Do not compute the left generalized Schur vectors.
jobvsl = 'V' or 'v': Compute the left generalized Schur vectors.
- jobvsr** Specifies whether the right generalized Schur vectors are to be computed, as follows:
- jobvsr** = 'N' or 'n': Do not compute the right generalized Schur vectors.
jobvsr = 'V' or 'v': Compute the right generalized Schur vectors.
- n** The order of the matrices *A* and *B*. $n \geq 0$.
- a** The *n*-by-*n* matrix *A*.
- lda** The leading dimension of array *a* in the calling program unit. $lda \geq \max(1, n)$.
- b** The *n*-by-*n* matrix *B*.

	ldb	The leading dimension of array b in the calling program unit. $\text{ldb} \geq \max(1, n)$.
	ldvsl	The leading dimension of array vsl in the calling program unit. $\text{ldvsl} \geq 1$, and if jobvsl = 'V' or 'v', then $\text{ldvsl} \geq n$.
	ldvsr	The leading dimension of array vsr in the calling program unit. $\text{ldvsr} \geq 1$, and if jobvsr = 'V' or 'v', then $\text{ldvsr} \geq n$.
	lwork	The length of array work . For SGEGS and DGEGS, $\text{lwork} \geq \max(1, 4n)$, while for CGEGS and ZGEGS, $\text{lwork} \geq \max(1, 2n)$. For good performance, lwork must generally be larger. The optimum value of lwork for high performance is returned in work(1) .
Working Storage	work, rwork	Arrays used for work space. On successful exit, work(1) contains the optimal work space length lwork for high performance.
Output	a	On successful exit, the generalized Schur Form of <i>A</i> overwrites the input.
	b	On successful exit, the generalized Schur Form of <i>B</i> overwrites the input.
	alpha, alphai, beta	On successful exit from SGEGS and DGEGS, $\text{alpha}(j)/\text{beta}(j)$ and $\text{alphai}(j)/\text{beta}(j)$ are the real and imaginary parts, respectively, of the generalized eigenvalue λ_j , $j = 1, 2, \dots, n$. The eigenvalues are not in any particular order, except that if $\text{alphai}(j) > 0$, then the j th and $j+1$ st eigenvalues are a complex conjugate pair, with $\text{alphai}(j+1) < 0$. Note: the quotients $\text{alpha}(j)/\text{beta}(j)$ and $\text{alphai}(j)/\text{beta}(j)$ may easily overflow or underflow, and $\text{beta}(j)$ may even be zero. Thus, you should avoid computing the ratios directly. However, $\text{alpha}(j)$ and $\text{alphai}(j)$ will be always less than and usually comparable in magnitude to $\ A\ $, and $\text{beta}(j)$ will be always less than and usually comparable to $\ B\ $.
	alpha, beta	On successful exit from CGEGS and ZGEGS, $\text{alpha}(j)/\text{beta}(j)$ is the generalized eigenvalue λ_j , $j = 1, 2, \dots, n$. The eigenvalues are not in any particular order. Note: the quotients $\text{alpha}(j)/\text{beta}(j)$ may easily overflow or underflow, and $\text{beta}(j)$ may even be zero. Thus, you should avoid computing the ratio directly. However, $\text{alpha}(j)$ will be always less than and usually comparable in magnitude to $\ A\ $, and $\text{beta}(j)$ will be always less than and usually comparable to $\ B\ $.
	vsl	On successful exit, if jobvsl = 'V' or 'v', the left generalized Schur vectors of <i>A</i> and <i>B</i> . Not referenced if jobvsl = 'N' or 'n'.
	vsr	On successful exit, if jobvsr = 'V' or 'v', the right generalized Schur vectors of <i>A</i> and <i>B</i> . Not referenced if jobvsr = 'N' or 'n'.
	info	Status response: info = 0: Successful exit. info < 0: If info = $-k$, the k -th argument had an invalid value. info > 0: The algorithm terminated before completing the requested computation.

Notes

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

`jobvsl` \neq 'N' or 'n' or 'V' or 'v'
`jobvsr` \neq 'N' or 'n' or 'V' or 'v'
`n` < 0 ,
`lda` $< \max(1,n)$,
`ldb` $< \max(1,n)$,
`ldvsl` too small, and
`ldvsr` too small, and
`lwork` too small.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding, for example, the `jobvsl` argument as 'NoVectors' for 'N' or 'Vectors' for 'V'.

General Matrix Eigenproblem**SGEGV/DGEGV/CGEGV/ZGEGV**

Purpose These subprograms compute the eigenvalues and, optionally, the eigenvectors of a generalized eigenproblem of the form $Az = \lambda Bz$, where A and B are n -by- n real or complex general matrices.

Given such matrices A and B , the generalized eigenvalues are scalars λ_i , $i = 1, 2, \dots, n$, for which there exist corresponding nonzero vectors, z_i , $i = 1, 2, \dots, n$, such that

$$Az_i = \lambda_i Bz_i.$$

λ_i is usually represented as a pair, (α_i, β_i) , such that $\lambda_i = \alpha_i/\beta_i$ if the ratio is defined. There is a reasonable interpretation for $\beta_i = 0$ and even for $\alpha_i = \beta_i = 0$. See (Golub and Van Loan) for details.

Optionally, the z_i , which are called right generalized eigenvectors, also may be computed. In addition, these subprograms also can compute the left generalized eigenvectors, which satisfy

$$y_i^*A = \lambda_i y_i^*B.$$

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```

CHARACTER*1 jobvl, jobvr
INTEGER*4 info, lda, ldb, ldvl, ldvr, lwork, n
REAL*4 a(lda, n), alphas(n), alphasr(n), beta(n), b(ldb, n),
vl(ldvl, n), vr(ldvr, n), work(lwork)
CALL SGEGV (jobvl, jobvr, n, a, lda, b, ldb, alphas, alphasr, beta,
vl, ldvl, vr, ldvr, work, lwork, info)

```

```

CHARACTER*1 jobvl, jobvr
INTEGER*4 info, lda, ldb, ldvl, ldvr, lwork, n
REAL*8 a(lda, n), alphas(n), alphasr(n), beta(n), b(ldb, n), beta(n),
vl(ldvl, n), vr(ldvr, n), work(lwork)
CALL DGEGV (jobvl, jobvr, n, a, lda, b, ldb, alphas, alphasr, beta,
vl, ldvl, vr, ldvr, work, lwork, info)

```

```

CHARACTER*1 jobvl, jobvr
INTEGER*4 info, lda, ldb, ldvl, ldvr, lwork, n
REAL*4 rwork(8*n)
COMPLEX*8 a(lda, n), alpha(n), b(ldb, n), beta(n),
vl(ldvl, n), vr(ldvr, n), work(lwork)
CALL CGEGV (jobvl, jobvr, n, a, lda, b, ldb, alpha, beta,
vl, ldvl, vr, ldvr, work, lwork, rwork, info)

```

```

CHARACTER*1 jobvl, jobvr
INTEGER*4 info, lda, ldb, ldvl, ldvr, lwork, n
REAL*8 rwork(8*n)
COMPLEX*16 a(lda, n), alpha(n), b(ldb, n), beta(n),
vl(ldvl, n), vr(ldvr, n), work(lwork)
CALL ZGEGV (jobvl, jobvr, n, a, lda, b, ldb, alpha, beta,
vl, ldvl, vr, ldvr, work, lwork, rwork, info)

```

LAPACK8, available on C Series and Exemplar architectures:

```

CHARACTER*1 jobvl, jobvr
INTEGER*8 info, lda, ldb, ldvl, ldvr, lwork, n
REAL*8 a(lda, n), alphas(n), alphasr(n), beta(n), b(ldb, n), beta(n),
vl(ldvl, n), vr(ldvr, n), work(lwork)
CALL SGEGV (jobvl, jobvr, n, a, lda, b, ldb, alphas, alphasr, beta,
vl, ldvl, vr, ldvr, work, lwork, info)

```

	CHARACTER*1	jobvl, jobvr	
	INTEGER*8	info, lda, ldb, ldvl, ldvr, lwork, n	
	REAL*8	rwork(8*n)	
	COMPLEX*16	a(lda, n), alpha(n), b(ldb, n), beta(n), vl(ldvl, n), vr(ldvr, n), work(lwork)	
	CALL CGEGV	(jobvl, jobvr, n, a, lda, b, ldb, alpha, beta, vl, ldvl, vr, ldvr, work, lwork, rwork, info)	
Input	jobvl	Specifies whether the left generalized eigenvectors of A and B are to be computed, as follows: jobvl = 'N' or 'n': Do not compute the left generalized eigenvectors. jobvl = 'V' or 'v': Compute the left generalized eigenvectors.	
	jobvr	Specifies whether the right generalized eigenvectors of A and B are to be computed, as follows: jobvr = 'N' or 'n': Do not compute the right generalized eigenvectors. jobvr = 'V' or 'v': Compute the right generalized eigenvectors.	
	n	The order of the matrices A and B . $n \geq 0$.	
	a	The n -by- n matrix A .	
	lda	The leading dimension of array a in the calling program unit. $lda \geq \max(1, n)$.	
	b	The n -by- n matrix B .	
	ldb	The leading dimension of array b in the calling program unit. $ldb \geq \max(1, n)$.	
	ldvl	The leading dimension of array vl in the calling program unit. $ldvl \geq 1$, and if jobvl = 'V' or 'v', then $ldvl \geq n$.	
	ldvr	The leading dimension of array vr in the calling program unit. $ldvr \geq 1$, and if jobvr = 'V' or 'v', then $ldvr \geq n$.	
	lwork	The length of array work . For SGEGV and DGEGV, $lwork \geq \max(1, 8n)$, while for CGEGV and ZGEGV, $lwork \geq \max(1, 2n)$. For good performance, lwork must generally be larger. The optimum value of lwork for high performance is returned in work(1) .	
Working Storage	work, rwork	Arrays used for work space. On successful exit, work(1) contains the optimal work space length lwork for high performance.	
Output	a, b	Destroyed.	
	alphar, alphai, beta	On successful exit from SGEGV and DGEGV, alphar(j)/beta(j) and alphai(j)/beta(j) are the real and imaginary parts, respectively, of the generalized eigenvalue λ_j , $j = 1, 2, \dots, n$. The eigenvalues are not in any particular order, except that if alphai(j) > 0 , then the j th and $j+1$ st eigenvalues are a complex conjugate pair, with alphai(j+1) < 0 .	

Note: the quotients $\mathbf{alpha}(j)/\mathbf{beta}(j)$ and $\mathbf{alpha}(j)/\mathbf{beta}(j)$ may easily overflow or underflow, and $\mathbf{beta}(j)$ may even be zero. Thus, you should avoid computing the ratios directly. However, $\mathbf{alpha}(j)$ and $\mathbf{alpha}(j)$ will be always less than and usually comparable in magnitude to $\|A\|$, and $\mathbf{beta}(j)$ will be always less than and usually comparable to $\|B\|$.

alpha, beta On successful exit from CGEGV and ZGEGV, $\mathbf{alpha}(j)/\mathbf{beta}(j)$ is the generalized eigenvalue λ_j , $j = 1, 2, \dots, n$. The eigenvalues are not in any particular order.

Note: the quotients $\mathbf{alpha}(j)/\mathbf{beta}(j)$ may easily overflow or underflow, and $\mathbf{beta}(j)$ may even be zero. Thus, you should avoid computing the ratio directly. However, $\mathbf{alpha}(j)$ will be always less than and usually comparable in magnitude to $\|A\|$, and $\mathbf{beta}(j)$ will be always less than and usually comparable to $\|B\|$.

vl On successful exit, if $\mathbf{jobvl} = 'V'$ or $'v'$, the left generalized eigenvectors, y_j , $j = 1, 2, \dots, n$, stored one after another in the columns of **vl**, in the same order as the generalized eigenvalues. The eigenvectors are normalized so the largest component will have $|\text{real part}| + |\text{imaginary part}| = 1$, except that for eigenvalues with $\mathbf{alpha}(j) = \mathbf{beta}(j) = 0$ or $\mathbf{alpha}(j) = \mathbf{alpha}(j) = \mathbf{beta}(j) = 0$, a zero vector will be returned as the corresponding eigenvector.

In SGEGV and DGEGV, a left generalized eigenvector corresponding to a real generalized eigenvalue is real and takes up one column. An eigenvector pair corresponding to a complex conjugate pair of eigenvalues is complex and takes up two columns: the first column holds the real part and the second column holds the imaginary part of the eigenvector corresponding to the eigenvalue with positive imaginary part.

In CGEGV and ZGEGV, each left generalized eigenvector takes up one column.

Not referenced if $\mathbf{jobvl} = 'N'$ or $'n'$.

vr On successful exit, if $\mathbf{jobvr} = 'V'$ or $'v'$, the right generalized eigenvectors, z_j , $j = 1, 2, \dots, n$, stored one after another in the columns of **vr**, in the same order as their generalized eigenvalues. The eigenvectors are normalized so the largest component will have $|\text{real part}| + |\text{imaginary part}| = 1$, except that for eigenvalues with $\mathbf{alpha}(j) = \mathbf{beta}(j) = 0$ or $\mathbf{alpha}(j) = \mathbf{alpha}(j) = \mathbf{beta}(j) = 0$, a zero vector will be returned as the corresponding eigenvector.

In SGEGV and DGEGV, a right generalized eigenvector corresponding to a real generalized eigenvalue is real and takes up one column. An eigenvector pair corresponding to a complex conjugate pair of eigenvalues is complex and takes up two columns: the first column holds the real part and the second column holds the imaginary part of the eigenvector corresponding to the eigenvalue with positive imaginary part.

In CGEGV and ZGEGV, each right generalized eigenvector takes up one column.

Not referenced if $\mathbf{jobvr} = 'N'$ or $'n'$.

info Status response:

info = 0: Successful exit.

info < 0: If **info** = $-k$, the k -th argument had an invalid value.

info > 0: The algorithm terminated before completing the requested computation.

Notes

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

jobvl ≠ 'N' or 'n' or 'V' or 'v',
jobvr ≠ 'N' or 'n' or 'V' or 'v',
n < 0,
lda < max(1,n),
ldb < max(1,n),
ldvl too small,
ldvr too small, and
lwork too small.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding the **jobvl** and **jobvr** arguments as 'NoVectors' for 'N' or 'Vectors' for 'V'.

Symmetric or Hermitian Band Matrices**SSBGV/DSBGV/CHBGV/ZHBGV**

Purpose These subprograms compute all eigenvalues and, optionally, all eigenvectors of a generalized eigenproblem of the form $Az = \lambda Bz$, where A and B are n -by- n real symmetric or complex Hermitian band matrices and B is positive definite.

A real matrix is symmetric if $A = A^T$, its transpose; a complex matrix is Hermitian if $A = A^*$, its conjugate transpose.

A matrix is banded if its nonzero elements all lie fairly near the principal diagonal. Specifically, $a_{ij} = 0$ if $|i-j| > ka$ for some integer ka . The smallest such ka for a given matrix is called the half bandwidth, and $2ka+1$ is called the total bandwidth.

A real symmetric matrix B is positive definite if the quadratic form $x^T Bx$ is positive for all nonzero real vectors x ; a complex Hermitian matrix B is positive definite if the quadratic form $x^* Bx$ is positive for all nonzero complex vectors x .

Given such matrices A and B , the generalized eigenvalues are scalars λ_i , $i = 1, 2, \dots, n$, for which there exist corresponding nonzero vectors, z_i , $i = 1, 2, \dots, n$, such that

$$Ax = \lambda Bx$$

Optionally, the generalized eigenvectors z_i also may be computed.

Matrix Storage Because it is not necessary to store or operate on the zeros outside the band of A or B , and since either triangle of A or B may be obtained from the other, you need only provide the band within one triangle of A and B . Compared to storing the entire matrix, this can save memory in two ways: only the elements within the bands are stored, and of them, only the upper or the lower triangles.

The following examples illustrate the storage of real symmetric band matrices. Consider the following matrix A of order $n = 7$ and half bandwidth $ka = 2$:

11	12	13	0	0	0	0
12	22	23	24	0	0	0
13	23	33	34	35	0	0
0	24	34	44	45	46	0
0	0	35	45	55	56	57
0	0	0	46	56	66	67
0	0	0	0	57	67	77

Upper triangular storage. The upper triangle of the matrix A is stored in an array **ab** with at least $ka+1 = 3$ rows and 7 columns as follows:

*	*	13	24	35	46	57
*	12	23	34	45	56	67
11	22	33	44	55	66	77

The asterisks represent elements in the ka -by- ka triangle at the upper left corner of **ab** that are not referenced. Thus, if a_{ij} is an element within the band of the upper triangle of A , it is stored in **ab**($ka+1+i-j, j$). Therefore, the columns of the upper triangle of A are stored in the columns of **ab**, and the diagonals of the upper triangle of A are stored in the rows of **ab**.

Lower triangular storage. The lower triangle of A is stored in the array **ab** as follows:

11	22	33	44	55	66	77
12	23	34	45	56	67	*
13	24	35	46	57	*	*

The asterisks represent elements in the ka -by- ka triangle at the lower right corner of **ab** that are not referenced. Thus, if a_{ij} is an element within the band of the lower triangle of A , it is stored in **ab**($1+i-j, j$). Therefore, the columns of the lower triangle of A are stored in the columns of **ab**, and the diagonals of the lower triangle of A are stored in the rows of **ab**.

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```

CHARACTER*1 jobz, uplo
INTEGER*4 info, ka, kb, ldab, ldbb, ldz, n
REAL*4 ab(ldab, n), bb(ldb, n), w(n), work(3*n), z(ldz, n)
CALL SSBGV (jobz, uplo, n, ka, kb, ab, ldab, bb, ldbb, w, z, ldz,
work, info)

```

```

CHARACTER*1 jobz, uplo
INTEGER*4 info, ka, kb, ldab, ldbb, ldz, n
REAL*8 ab(ldab, n), bb(ldb, n), w(n), work(3*n), z(ldz, n)
CALL DSBGV (jobz, uplo, n, ka, kb, ab, ldab, bb, ldbb, w, z, ldz,
work, info)

```

```

CHARACTER*1 jobz, uplo
INTEGER*4 info, ka, kb, ldab, ldbb, ldz, n
REAL*4 rwork(3*n), w(n)
COMPLEX*8 ab(ldab, n), bb(ldb, n), work(n), z(ldz, n)
CALL CHBGV (jobz, uplo, n, ka, kb, ab, ldab, bb, ldbb, w, z, ldz,
work, rwork, info)

```

```

CHARACTER*1 jobz, uplo
INTEGER*4 info, ka, kb, ldab, ldbb, ldz, n
REAL*8 rwork(3*n), w(n)
COMPLEX*16 ab(ldab, n), bb(ldb, n), work(n), z(ldz, n)
CALL ZHBGV (jobz, uplo, n, ka, kb, ab, ldab, bb, ldbb, w, z, ldz,
work, rwork, info)

```

LAPACK8, available on C Series and Exemplar architectures:

```

CHARACTER*1 jobz, uplo
INTEGER*8 info, ka, kb, ldab, ldbb, ldz, n
REAL*8 ab(ldab, n), bb(ldb, n), w(n), work(3*n), z(ldz, n)
CALL SSBGV (jobz, uplo, n, ka, kb, ab, ldab, bb, ldbb, w, z, ldz,
work, info)

```

```

CHARACTER*1 jobz, uplo
INTEGER*8 info, ka, kb, ldab, ldbb, ldz, n
REAL*8 rwork(3*n), w(n)
COMPLEX*16 ab(ldab, n), bb(ldb, n), work(n), z(ldz, n)
CALL CHBGV (jobz, uplo, n, ka, kb, ab, ldab, bb, ldbb, w, z, ldz,
work, rwork, info)

```

Continued

Input	jobz	Specifies whether eigenvectors are to be computed, as follows: jobz = 'N' or ' n ': Compute eigenvalues only. jobz = 'V' or ' v ': Compute eigenvectors as well.	
	uplo	Specifies whether the upper or lower triangular parts of the symmetric or Hermitian matrices A and B are stored in arrays ab and bb , respectively, as follows: uplo = 'U' or ' u ': The upper triangular parts are stored. uplo = 'L' or ' l ': The lower triangular parts are stored.	
	n	The order of the matrices A and B . $n \geq 0$.	
	ka	The number of super-diagonals of the matrix A if uplo = 'U' or ' u ', or the number of subdiagonals if uplo = 'L' or ' l '. $ka \geq 0$.	
	kb	The number of super-diagonals of the matrix B if uplo = 'U' or ' u ', or the number of subdiagonals if uplo = 'L' or ' l '. $0 \leq kb \leq ka$.	
	ab	The upper or lower triangle of the symmetric or Hermitian band matrix A , stored in the first $ka+1$ rows of array ab . The j -th column of A is stored in the j -th column of array ab as follows: If uplo = 'U' or ' u ', $ab(ka+1+i-j, j) = A(i, j)$ for $\max(1, j-ka) \leq i \leq j$; If uplo = 'L' or ' l ', $ab(1+i-j, j) = A(i, j)$ for $j \leq i \leq \min(n, j+ka)$.	
	ldab	The leading dimension of array ab in the calling program unit. $ldab \geq ka+1$.	
	bb	The upper or lower triangle of the symmetric or Hermitian band matrix B , stored in the first $kb+1$ rows of array bb . The j -th column of B is stored in the j -th column of array bb as follows: If uplo = 'U' or ' u ', $bb(kb+1+i-j, j) = B(i, j)$ for $\max(1, j-kb) \leq i \leq j$; If uplo = 'L' or ' l ', $bb(1+i-j, j) = B(i, j)$ for $j \leq i \leq \min(n, j+kb)$.	
	ldbb	The leading dimension of array bb in the calling program unit. $ldbb \geq kb+1$.	
	ldz	The leading dimension of array z in the calling program unit. $ldz \geq 1$, and if jobz = 'V' or ' v ', then $ldz \geq n$.	
	Working Storage	work, rwork	Arrays used for work space.
	Output	ab, bb	Destroyed.
		w	On successful exit, the generalized eigenvalues in ascending order.
		z	On successful exit, if jobz = 'V' or ' v ', the generalized eigenvectors. Not referenced if jobz = 'N' or ' n '.

info Status response:

info = 0: Successful exit.

info < 0: If **info** = $-k$, the k -th argument had an invalid value.

info > 0: If **info** = $k \leq n$, the algorithm failed to converge; k off-diagonal elements of an intermediate tridiagonal form did not converge to zero. If **info** = $k > n$, then the leading minor of order $k-n$ of B is not positive definite and no eigenvalues or eigenvectors could be computed.

Notes

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

jobz \neq 'N' or 'n' or 'V' or 'v',
uplo \neq 'L' or 'l' or 'U' or 'u',
n < 0,
ka < 0,
kb < 0,
kb > **ka**,
ldab < **ka**+1,
ldbb < **kb**+1, and
ldz too small.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding, for example, the **jobz** argument as 'NoVectors' for 'N' or 'Vectors' for 'V'.

Symmetric or Hermitian Packed Matrices**SSPGV/DSPGV/.../ZHPGV****Purpose**

These subprograms compute all eigenvalues and, optionally, all eigenvectors of generalized eigenproblems of the form $Az = \lambda Bz$, $ABz = \lambda z$, or $BAz = \lambda z$, where A and B are n -by- n real symmetric or complex Hermitian matrices stored in packed form and B is positive definite.

A real matrix is symmetric if $A = A^T$, its transpose; a complex matrix is Hermitian if $A = A^*$, its conjugate transpose.

A real symmetric matrix B is positive definite if the quadratic form $x^T B x$ is positive for all nonzero real vectors x ; a complex Hermitian matrix B is positive definite if the quadratic form $x^* B x$ is positive for all nonzero complex vectors x .

Given such matrices A and B , the generalized eigenvalues are scalars λ_i , $i = 1, 2, \dots, n$, for which there exist corresponding nonzero vectors, z_i , $i = 1, 2, \dots, n$, such that

$$Ax = \lambda Bx \quad \text{or} \quad ABx = \lambda x \quad \text{or} \quad BAx = \lambda x.$$

Optionally, the generalized eigenvectors z_i also may be computed.

Matrix Storage

Because either triangle of A or B may be obtained from its other triangle, you need only provide either the upper or the lower triangle of A , and the same triangle of B . Compared to storing the entire matrices, you save memory by supplying only one triangle of each matrix, stored column-by-column in packed form in two 1-dimensional arrays.

The following examples illustrate the packed storage of symmetric or Hermitian matrices.

Upper triangular storage. If the upper triangle of A is

$$\begin{array}{cccc} 11 & 12 & 13 & 14 \\ & 22 & 23 & 24 \\ & & 33 & 34 \\ & & & 44 \end{array}$$

then A is packed column-by-column into an array **ap** as follows:

k	1	2	3	4	5	6	7	8	9	10
ap(k)	11	12	22	13	23	33	14	24	34	44

Upper triangular matrix element a_{ij} is stored in array element **ap**($i+j \times (j-1)/2$).

Lower triangular storage. If the lower triangle of A is

$$\begin{array}{cccc} 11 & & & \\ 21 & 22 & & \\ 31 & 32 & 33 & \\ 41 & 42 & 43 & 44 \end{array}$$

then A is packed column-by-column into an array **ap** as follows:

k	1	2	3	4	5	6	7	8	9	10
ap(k)	11	21	31	41	22	32	42	33	43	44

Lower triangular matrix element a_{ij} is stored in array element **ap**($i+(j-1) \times (2n-j)/2$).

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```

CHARACTER*1 jobz, uplo
INTEGER*4    info, itype, ldz, n
REAL*4      ap((n*(n+1))/2), bp((n*(n+1))/2), w(n),
              work(3*n), z(ldz, n)
CALL SSPGV (itype, jobz, uplo, n, ap, bp, w, z, ldz, work, info)

```

```

CHARACTER*1 jobz, uplo
INTEGER*4    info, itype, ldz, n
REAL*8      ap((n*(n+1))/2), bp((n*(n+1))/2), w(n),
              work(3*n), z(ldz, n)
CALL DSPGV (itype, jobz, uplo, n, ap, bp, w, z, ldz, work, info)

```

```

CHARACTER*1 jobz, uplo
INTEGER*4    info, itype, ldz, n
REAL*4      rwork(max(1,3*n-2)), w(n)
COMPLEX*8   ap((n*(n+1))/2), bp((n*(n+1))/2),
              work(max(1,2*n-1)), z(ldz, n)
CALL CHPGV (itype, jobz, uplo, n, ap, bp, w, z, ldz, work, rwork,
              info)

```

```

CHARACTER*1 jobz, uplo
INTEGER*4    info, itype, ldz, n
REAL*8      rwork(max(1,3*n-2)), w(n)
COMPLEX*16  ap((n*(n+1))/2), bp((n*(n+1))/2),
              work(max(1,2*n-1)), z(ldz, n)
CALL ZHPGV (itype, jobz, uplo, n, ap, bp, w, z, ldz, work, rwork,
              info)

```

LAPACK8, available on C Series and Exemplar architectures:

```

CHARACTER*1 jobz, uplo
INTEGER*8    info, itype, ldz, n
REAL*8      ap((n*(n+1))/2), bp((n*(n+1))/2), w(n),
              work(3*n), z(ldz, n)
CALL SSPGV (itype, jobz, uplo, n, ap, bp, w, z, ldz, work, info)

```

```

CHARACTER*1 jobz, uplo
INTEGER*8    info, itype, ldz, n
REAL*8      rwork(max(1,3*n-2)), w(n)
COMPLEX*16  ap((n*(n+1))/2), bp((n*(n+1))/2),
              work(max(1,2*n-1)), z(ldz, n)
CALL CHPGV (itype, jobz, uplo, n, ap, bp, w, z, ldz, work, rwork,
              info)

```

Input **itype** Specifies the problem type to be solved, as follows:

```

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

```

jobz Specifies whether eigenvectors are to be computed, as follows:

```

jobz = 'N' or 'n':  Compute eigenvalues only.
jobz = 'V' or 'v':  Compute eigenvectors as well.

```

	uplo	Specifies whether the upper or lower triangular parts of the symmetric or Hermitian matrices A and B are stored in arrays ap and bp , respectively, as follows: uplo = 'U' or 'u': The upper triangular parts are stored. uplo = 'L' or 'l': The lower triangular parts are stored.
	n	The order of the matrices A and B . $n \geq 0$.
	ap	The upper or lower triangular part of the symmetric or Hermitian matrix A , packed columnwise in a linear array as follows: If uplo = 'U' or 'u', $ap(i + (j-1) \times j/2) = A(i, j)$ for $1 \leq i \leq j$; If uplo = 'L' or 'l', $ap(i + (j-1) \times (2n-j)/2) = A(i, j)$ for $j \leq i \leq n$.
	bp	The upper or lower triangular part of the symmetric or Hermitian matrix B , packed columnwise in a linear array as follows: If uplo = 'U' or 'u', $bp(i + (j-1) \times j/2) = B(i, j)$ for $1 \leq i \leq j$; If uplo = 'L' or 'l', $bp(i + (j-1) \times (2n-j)/2) = B(i, j)$ for $j \leq i \leq n$.
	ldz	The leading dimension of array z in the calling program unit. $ldz \geq 1$, and if jobz = 'V' or 'v', then $ldz \geq n$.
Working Storage	work, rwork	Arrays used for work space.
Output	ap	Destroyed.
	bp	On exit with info $\leq n$, the triangular factor U or L from the Cholesky factorization $B = U*U$ or $B = LL^*$, in the same storage format as B .
	w	On successful exit, the eigenvalues in ascending order. On exit with info $\leq n$, the eigenvalues are correct for indices 1, 2, ..., info -1, but they are unordered and may not be the smallest eigenvalues of the problem.
	z	On successful exit, if jobz = 'V' or 'v', the eigenvectors. If an error exit is made, z contains the eigenvectors associated with the stored eigenvalues. The eigenvectors are normalized as follows: if itype = 1 or 2, then $z_j^* B z_j = 1$; if itype = 3, then $z_j^* B^{-1} z_j = 1$. Not referenced if jobz = 'N' or 'n'.
	info	Status response: info = 0: Successful exit. info < 0: If info = $-k$, the k -th argument had an invalid value. info > 0: If info = $k \leq n$, the algorithm terminated before finding the k -th eigenvalue. If info = $k > n$, then the leading minor of order $k-n$ of B is not positive definite and no eigenvalues or eigenvectors could be computed.

Notes

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

itype \neq 1, 2, or 3,
jobz \neq 'N' or 'n' or 'V' or 'v',
uplo \neq 'L' or 'l' or 'U' or 'u',
n $<$ 0, and
ldz too small.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding, for example, the **jobz** argument as 'NoVectors' for 'N' or 'Vectors' for 'V'.

Symmetric or Hermitian Matrices**SSYGV/DSYGV/CHEGV/ZHEGV**

Purpose These subprograms compute all eigenvalues and, optionally, all eigenvectors of generalized eigenproblems of the form $Az = \lambda Bz$, $ABz = \lambda z$, or $BAz = \lambda z$, where A and B are n -by- n real symmetric or complex Hermitian matrices and B is positive definite.

A real matrix is symmetric if $A = A^T$, its transpose; a complex matrix is Hermitian if $A = A^*$, its conjugate transpose.

A real symmetric matrix B is positive definite if the quadratic form $x^T Bx$ is positive for all nonzero real vectors x ; a complex Hermitian matrix B is positive definite if the quadratic form $x^* Bx$ is positive for all nonzero complex vectors x .

Given such matrices A and B , the generalized eigenvalues are scalars λ_i , $i = 1, 2, \dots, n$, for which there exist corresponding nonzero vectors, z_i , $i = 1, 2, \dots, n$, such that

$$Ax = \lambda Bx \quad \text{or} \quad ABx = \lambda x \quad \text{or} \quad BAx = \lambda x$$

Optionally, the generalized eigenvectors z_i also may be computed.

Matrix Storage Because either triangle of A or B may be obtained from its other triangle, you need only provide one triangle of A and one triangle of B . You may supply either the upper or the lower triangle of A , and the same triangle of B , in 2 two-dimensional arrays large enough to hold the entire matrices. The other triangle of the arrays are not referenced.

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```
CHARACTER*1 jobz, uplo
INTEGER*4   info, itype, lda, ldb, lwork, n
REAL*4      a(lda, n), b(ldb, n), w(n), work(lwork)
CALL SSYGV (itype, jobz, uplo, n, a, lda, b, ldb, w, work, lwork,
           info)
```

```
CHARACTER*1 jobz, uplo
INTEGER*4   info, itype, lda, ldb, lwork, n
REAL*8      a(lda, n), b(ldb, n), w(n), work(lwork)
CALL DSYGV (itype, jobz, uplo, n, a, lda, b, ldb, w, work, lwork,
           info)
```

```
CHARACTER*1 jobz, uplo
INTEGER*4   info, itype, lda, ldb, lwork, n
REAL*4      rwork(max(1,3*n-2)), w(n)
COMPLEX*8   a(lda, n), b(ldb, n), work(lwork)
CALL CHEGV (itype, jobz, uplo, n, a, lda, b, ldb, w, work, lwork,
           rwork, info)
```

```
CHARACTER*1 jobz, uplo
INTEGER*4   info, itype, lda, ldb, lwork, n
REAL*8      rwork(max(1,3*n-2)), w(n)
COMPLEX*16  a(lda, n), b(ldb, n), work(lwork)
CALL ZHEGV (itype, jobz, uplo, n, a, lda, b, ldb, w, work, lwork,
           rwork, info)
```

LAPACK8, available on C Series and Exemplar architectures:

```

CHARACTER*1 jobz, uplo
INTEGER*8    info, itype, lda, ldb, lwork, n
REAL*8      a(lda, n), b(ldb, n), w(n), work(lwork)
CALL SSYGV (itype, jobz, uplo, n, a, lda, b, ldb, w, work, lwork,
             info)

```

```

CHARACTER*1 jobz, uplo
INTEGER*8    info, itype, lda, ldb, lwork, n
REAL*8      rwork(max(1,3*n-2)), w(n)
COMPLEX*16  a(lda, n), b(ldb, n), work(lwork)
CALL CHEGV (itype, jobz, uplo, n, a, lda, b, ldb, w, work, lwork,
             rwork, info)

```

Input **itype** Specifies the problem type to be solved, as follows:

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

jobz Specifies whether eigenvectors are to be computed, as follows:

jobz = 'N' or 'n': Compute eigenvalues only.
 jobz = 'V' or 'v': Compute eigenvectors as well.

uplo Specifies whether the upper or lower triangular parts of the symmetric or Hermitian matrices **A** and **B** are stored in arrays **a** and **b**, respectively, as follows:

uplo = 'U' or 'u': The upper triangular parts are stored.
 uplo = 'L' or 'l': The lower triangular parts are stored.

n The order of the matrices **A** and **B**. $n \geq 0$.

a The symmetric or Hermitian matrix **A**.

 If **uplo** = 'U' or 'u', only the upper triangular part of **a** is used to define the elements of the matrix and the strict lower triangular part of **a** is not used for input.

 If **uplo** = 'L' or 'l', only the lower triangular part of **a** is used to define the elements of the matrix and the strict upper triangular part of **a** is not used for input.

lda The leading dimension of array **a** in the calling program unit. $lda \geq \max(1, n)$.

b The symmetric positive definite matrix **B**.

 If **uplo** = 'U' or 'u', only the upper triangular part of **b** is used to define the elements of the matrix and the strict lower triangular part of **b** is not used for input.

 If **uplo** = 'L' or 'l', only the lower triangular part of **b** is used to define the elements of the matrix and the strict upper triangular part of **b** is not used for input.

ldb The leading dimension of array **b** in the calling program unit. $ldb \geq \max(1, n)$.

Continued

	lwork	The length of array work . $\text{lwork} \geq \max(1, 3n-1)$. For good performance, lwork must generally be larger. The optimum value of lwork for high performance is returned in work(1) .
Working Storage	work , rwork	Arrays used for work space. On successful exit, work(1) contains the optimal work space length lwork for high performance.
Output	a	On successful exit, if jobz = 'V' or 'v', the orthonormal eigenvectors of the matrix. If an error exit is made, a contains the eigenvectors associated with the stored eigenvalues. The eigenvectors are normalized as follows: if itype = 1 or 2, then $z_j^* B z_j = 1$; if itype = 3, then $z_j^* B^{-1} z_j = 1$. If jobz = 'N' or 'n', then the triangle of a specified by uplo , including the diagonal, has been destroyed.
	b	On exit with info \leq n , the triangular factor <i>U</i> or <i>L</i> from the Cholesky factorization $B = U^*U$ or $B = LL^*$, in the same storage format as <i>B</i> .
	w	On successful exit, the eigenvalues in ascending order. On exit with info \leq n , the eigenvalues are correct for indices 1, 2, ..., info -1, but they are unordered and may not be the smallest eigenvalues of the problem.
	info	Status response: info = 0: Successful exit. info < 0: If info = - <i>k</i> , the <i>k</i> -th argument had an invalid value. info > 0: If info = <i>k</i> \leq n , the algorithm terminated before finding the <i>k</i> -th eigenvalue. If info = <i>k</i> > n , then the leading minor of order <i>k</i> - n of <i>B</i> is not positive definite and no eigenvalues or eigenvectors could be computed.

Notes

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

itype \neq 1, 2, or 3,
jobz \neq 'N' or 'n' or 'V' or 'v',
uplo \neq 'L' or 'l' or 'U' or 'u',
n < 0,
lda < max(1,**n**),
ldb < max(1,**n**), and
lwork too small.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the CALL statement may be improved by coding, for example, the **jobz** argument as 'NoVectors' for 'N' or 'Vectors' for 'V'.

Drivers for the Singular Value Decomposition

Overview

This chapter explains how to use LAPACK subprograms to compute the singular value decomposition of a matrix or the generalized singular value decomposition of a pair of matrices.

Chapter Objectives

After reading this chapter you will know how to use the described subprograms.

What You Need to Know to Use These Subprograms

To use these subprograms you should know what the singular value decomposition of a matrix is and that such a decomposition exists for any matrix. It would be helpful to be familiar with some of the interpretations and applications of singular values, and especially to understand how small singular values may indicate ill-conditioning or numerical singularity, and how the SVD can be used to deal with matrices that, for computational purposes, are rank deficient. These concepts are beyond the scope of this chapter introduction; please refer to (Golub and Van Loan).

Supplemental Reading

Golub, G.H. and C.F. Van Loan. *Matrix Computations*, 2nd Edition. Baltimore, MD: The Johns Hopkins University Press. 1989.

Subprogram Descriptions

Singular Value Decomposition of a General Matrix SGESVD, DGESVD, CGESVD, ZGESVD	10-2
Generalized Singular Value Decomposition SGGSVD, DGGSD, CGGSVD, ZGGSD	10-5

Purpose These subprograms compute the singular value decomposition (SVD) of an m -by- n matrix A , optionally computing some or all of the left and/or right singular vectors. The SVD of A is written

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

where Σ is a diagonal matrix with the singular values on the diagonal, and U and V are orthogonal or unitary. The columns of U are the left singular vectors and the columns of V are the right singular vectors. If the right singular vectors are requested, V^T is returned.

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```

CHARACTER*1  jobu, jobvt
INTEGER*4    info, lda, ldu, ldvt, lwork, m, n
REAL*4      a(lda, n), s(min(m,n)), u(ldu, ucol), vt(ldvt, n),
              work(lwork)
CALL SGESVD (jobu, jobvt, m, n, a, lda, s, u, ldu, vt, ldvt,
              work, lwork, info)

```

```

CHARACTER*1  jobu, jobvt
INTEGER*4    info, lda, ldu, ldvt, lwork, m, n
REAL*8      a(lda, n), s(min(m,n)), u(ldu, ucol), vt(ldvt, n),
              work(lwork)
CALL DGESVD (jobu, jobvt, m, n, a, lda, s, u, ldu, vt, ldvt,
              work, lwork, info)

```

```

CHARACTER*1  jobu, jobvt
INTEGER*4    info, lda, ldu, ldvt, lwork, m, n
REAL*4      rwork(5*min(m,n)), s(min(m,n))
COMPLEX*8   a(lda, n), u(ldu, ucol), vt(ldvt, n), work(lwork)
CALL CGESVD (jobu, jobvt, m, n, a, lda, s, u, ldu, vt, ldvt,
              work, lwork, rwork, info)

```

```

CHARACTER*1  jobu, jobvt
INTEGER*4    info, lda, ldu, ldvt, lwork, m, n
REAL*8      rwork(5*min(m,n)), s(min(m,n))
COMPLEX*16  a(lda, n), u(ldu, ucol), vt(ldvt, n), work(lwork)
CALL ZGESVD (jobu, jobvt, m, n, a, lda, s, u, ldu, vt, ldvt,
              work, lwork, rwork, info)

```

LAPACK8, available on C Series and Exemplar architectures:

```

CHARACTER*1  jobu, jobvt
INTEGER*8    info, lda, ldu, ldvt, lwork, m, n
REAL*8      a(lda, n), s(min(m,n)), u(ldu, ucol), vt(ldvt, n),
              work(lwork)
CALL SGESVD (jobu, jobvt, m, n, a, lda, s, u, ldu, vt, ldvt,
              work, lwork, info)

```

```

CHARACTER*1  jobu, jobvt
INTEGER*8    info, lda, ldu, ldvt, lwork, m, n
REAL*8      rwork(5*min(m,n)), s(min(m,n))
COMPLEX*16  a(lda, n), u(ldu, ucol), vt(ldvt, n), work(lwork)
CALL CGESVD (jobu, jobvt, m, n, a, lda, s, u, ldu, vt, ldvt,
              work, lwork, rwork, info)

```

Continued

Input	jobu	Specifies which left singular vectors to compute, as follows: jobu = 'A' or 'a': All m m -dimensional left singular vectors are returned in u . jobu = 'S' or 's': Only min(m,n) m -dimensional left singular vectors are returned in u . jobu = 'O' or 'o': Only min(m,n) m -dimensional left singular vectors overwrite a . jobu = 'N' or 'n': No left singular vectors are computed.
	jobvt	Specifies which right singular vectors to compute, as follows: jobvt = 'A' or 'a': All n n -dimensional right singular vectors are returned in vt . jobvt = 'S' or 's': Only min(m,n) n -dimensional right singular vectors are returned in vt . jobvt = 'O' or 'o': Only min(m,n) n -dimensional right singular vectors overwrite a . jobvt = 'N' or 'n': No right singular vectors are computed. jobvt and jobu cannot simultaneously be 'O' or 'o'.
	m	The number of rows of the matrix A . m ≥ 0.
	n	The number of columns of the matrix A . n ≥ 0.
	a	The m -by- n matrix A .
	lda	The leading dimension of array a in the calling program unit. lda ≥ max(1,m) .
	ldu	The leading dimension of array u in the calling program unit. ldu ≥ 1. If jobu = 'S' or 's' or 'A' or 'a' , then ldu ≥ m .
	ldvt	The leading dimension of array vt in the calling program unit. ldvt ≥ 1, and if jobvt = 'S' or 's' , then ldvt ≥ min(m,n) , or if jobvt = 'A' or 'a' , then ldvt ≥ n .
	lwork	The length of array work . lwork ≥ max(1,3min(m,n)+max(m,n), 5min(m,n)-4) . For good performance, lwork must generally be larger. The optimum value of lwork for high performance is returned in work(1) .
Working Storage	work, rwork	Arrays used for work space. On successful exit, work(1) contains the optimal work space length lwork for high performance.
Output	a	On successful exit, if jobu = 'O' or 'o' , a contains min(m,n) left singular vectors. On successful exit, if jobvt = 'O' or 'o' , a contains min(m,n) right singular vectors.
	s	On successful exit, the singular values of A , sorted into nonincreasing order.

- u** On successful exit, none, some, or all of the left singular vectors of A , that is, columns of the matrix U . The left singular vectors are of dimension m . The number of left singular vectors returned, and hence the second dimension, $ucol$, of the array u , depends on $jobu$ as follows:
- If $jobu = 'A'$ or $'a'$, then m left singular vectors are returned and $ucol \geq m$.
- If $jobu = 'S'$ or $'s'$, then $\min(m,n)$ left singular vectors are returned and $ucol \geq \min(m,n)$.
- If $jobu = 'N'$ or $'n'$ or $'O'$ or $'o'$ then no left singular vectors are returned and u is not referenced.
- vt** On successful exit, the rows of vt hold none, some, or all of the right singular vectors of A , that is, columns of the matrix V . The right singular vectors are of dimension n .
- If $jobvt = 'A'$ or $'a'$, then n right singular vectors are returned.
- If $jobvt = 'S'$ or $'s'$, then $\min(m,n)$ right singular vectors are returned
- If $jobvt = 'N'$ or $'n'$ or $'O'$ or $'o'$ then no right singular vectors are returned and vt is not referenced
- info** Status response:
- info** = 0: Successful exit.
info < 0: If **info** = $-k$, the k -th argument had an invalid value.
info > 0: The algorithm did not converge.

Notes

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

$jobu \neq 'A'$ or $'a'$ or $'S'$ or $'s'$ or $'O'$ or $'o'$ or $'N'$ or $'n'$,
 $jobvt \neq 'A'$ or $'a'$ or $'S'$ or $'s'$ or $'O'$ or $'o'$ or $'N'$ or $'n'$,
 $m < 0$,
 $n < 0$,
 $lda < \max(1,m)$,
 ldu too small,
 $ldvt$ too small, and
 $lwork$ too small.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding, for example, the $jobu$ argument as 'AllVectors' for 'A', 'SomeVectors' for 'S', 'OverwriteA' for 'O', or 'NoVectors' for 'N'.

Generalized SVD

SGGSVD/DGGSVD/CGGSVD/ZGGSVD

Purpose These subprograms compute the generalized singular value decomposition (GSVD) of the m -by- n matrix A and the p -by- n matrix B .

Let l be the effective numerical rank of the matrix B and let $k+l$ be the effective numerical rank of the matrix $[A^* B^*]^*$, where $*$ indicates conjugate transpose (ordinary transpose if the matrices are real). Then the GSVD of A and B is written

$$U^*AQ = D_1[0 R], \quad V^*BQ = D_2[0 R] \quad (1)$$

where U is an m -by- m orthogonal or unitary matrix, V is a p -by- p orthogonal or unitary matrix, Q is an n -by- n orthogonal or unitary matrix, D_1 is a real m -by- $(k+l)$ diagonal matrix, D_2 is a real p -by- $(k+l)$ "diagonal" matrix, 0 is a $(k+l)$ -by- $(n-k-l)$ zero matrix, and R is a $(k+l)$ -by- $(k+l)$ nonsingular upper triangular matrix.

Alternatively, the GSVD of A and B is sometimes presented in the form

$$U^*AX = [0 D_1], \quad V^*BX = [0 D_2]. \quad (2)$$

where U , V , D_1 , and D_2 are as above and X is an n -by- n nonsingular matrix. Forms (1) and (2) are equivalent if

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

D_1 , D_2 , and R have the following structures, where subscripts on zero and identity matrices indicate their dimensions:

If $k+l \leq m$:

$$D_1 = \begin{bmatrix} I_{kk} & 0_{kl} \\ 0_{lk} & C \\ 0_{m-k-l,k} & 0_{m-k-l,l} \end{bmatrix}$$

$$D_2 = \begin{bmatrix} 0_{l,k} & S \\ 0_{p-l,k} & 0_{p-l,l} \end{bmatrix}$$

$$R = \begin{bmatrix} R_{11} & R_{12} \\ 0_{lk} & R_{22} \end{bmatrix}$$

where

$$C = \text{diag}(\alpha_{k+1}, \alpha_{k+2}, \dots, \alpha_{k+l}),$$

with

$$S = \text{diag}(\beta_{k+1}, \beta_{k+2}, \dots, \beta_{k+l}),$$

$$0 \leq \alpha_i \leq 1, \quad i = k+1, k+2, \dots, k+l,$$

$$0 \leq \beta_i \leq 1, \quad i = k+1, k+2, \dots, k+l,$$

and

$$\alpha_i^2 + \beta_i^2 = 1, \quad i = k+1, k+2, \dots, k+l.$$

If $k+l > m$:

$$D_1 = \begin{bmatrix} I_{kk} & 0_{k,m-k} & 0_{k,k+l-m} \\ 0_{m-k,k} & C & 0_{m-k,k+l-m} \end{bmatrix}$$

$$D_2 = \begin{bmatrix} 0_{m-k,k} & S & 0_{m-k,k+l-m} \\ 0_{k+l-m,k} & 0_{k+l-m,m-k} & I_{k+l-m,k+l-m} \\ 0_{p-l,k} & 0_{p-l,m-k} & 0_{p-l,k+l-m} \end{bmatrix}$$

$$R = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ 0_{m-k,k} & R_{22} & R_{23} \\ 0_{k+l-m,k} & 0_{k+l-m,m-k} & R_{33} \end{bmatrix}$$

where

$$C = \text{diag}(\alpha_{k+1}, \alpha_{k+2}, \dots, \alpha_m),$$

$$S = \text{diag}(\beta_{k+1}, \beta_{k+2}, \dots, \beta_m),$$

with

$$0 \leq \alpha_i \leq 1, \quad i = k+1, k+2, \dots, m,$$

$$0 \leq \beta_i \leq 1, \quad i = k+1, k+2, \dots, m,$$

and

$$\alpha_i^2 + \beta_i^2 = 1, \quad i = k+1, k+2, \dots, m.$$

The ratios α_i/β_i , $i = k+1, k+2, \dots, \min(k+l, m)$, are called the generalized singular values of A and B . Avoid computing these ratios directly, or compute them with caution, to prevent overflow or division by zero.

The subprograms compute C , S , R , and optionally the orthogonal or unitary transformation matrices U , V , and Q .

If B is square and nonsingular, the GSVD of A and B implicitly gives the singular value decomposition of the matrix AB^{-1} :

$$AB^{-1} = U(D_1 D_2^{-1})V^*.$$

Usage

LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```

CHARACTER*1 jobq, jobu, jobv
INTEGER*4 info, k, l, lda, ldb, ldq, ldu, ldv, m, n, p
INTEGER*4 iwork(n)
REAL*4 a(lda, n), alpha(n), b(ldb, n), beta(n), q(ldq, n),
u(ldu, m), v(ldv, n), work(max(3*n, m, p)+n)
CALL 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*1 jobq, jobu, jobv
INTEGER*4 info, k, l, lda, ldb, ldq, ldu, ldv, m, n, p
INTEGER*4 iwork(n)
REAL*8 a(lda, n), alpha(n), b(ldb, n), beta(n), q(ldq, n),
u(ldu, m), v(ldv, n), work(max(3*n, m, p)+n)
CALL DGGGSVD (jobu, jobv, jobq, m, n, p, k, l, a, lda, b, ldb,
alpha, beta, u, ldu, v, ldv, q, ldq, work, iwork,
info)

```

CHARACTER*1 jobq, jobu, jobv
INTEGER*4 info, k, l, lda, ldb, ldq, ldu, ldv, m, n, p
INTEGER*4 iwork(n)
REAL*4 alpha(n), beta(n), rwork(2*n)
COMPLEX*8 a(lda, n), b(ldb, n), q(ldq, n), u(ldu, m), v(ldv, n),
work(max(3*n,m,p)+n)
CALL 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*1 jobq, jobu, jobv
INTEGER*4 info, k, l, lda, ldb, ldq, ldu, ldv, m, n, p
INTEGER*4 iwork(n)
REAL*8 alpha(n), beta(n), rwork(2*n)
COMPLEX*16 a(lda, n), b(ldb, n), q(ldq, n), u(ldu, m), v(ldv, n),
work(max(3*n,m,p)+n)
CALL ZGGSVD (jobu, jobv, jobq, m, n, p, k, l, a, lda, b, ldb,
alpha, beta, u, ldu, v, ldv, q, ldq, work, rwork,
iwork, info)

LAPACK8, available on C Series and Exemplar architectures:

CHARACTER*1 jobq, jobu, jobv
INTEGER*8 info, k, l, lda, ldb, ldq, ldu, ldv, m, n, p
INTEGER*8 iwork(n)
REAL*8 a(lda, n), alpha(n), b(ldb, n), beta(n), q(ldq, n),
u(ldu, m), v(ldv, n), work(max(3*n,m,p)+n)
CALL 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*1 jobq, jobu, jobv
INTEGER*8 info, k, l, lda, ldb, ldq, ldu, ldv, m, n, p
INTEGER*8 iwork(n)
REAL*8 alpha(n), beta(n), rwork(2*n)
COMPLEX*16 a(lda, n), b(ldb, n), q(ldq, n), u(ldu, m), v(ldv, n),
work(max(3*n,m,p)+n)
CALL 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)

Input

jobu Specifies whether to compute the U matrix or not, as follows:

jobu = 'N' or 'n': Do not compute the U matrix.
jobu = 'U' or 'u': Compute the U matrix.

jobv Specifies whether to compute the V matrix or not, as follows:

jobv = 'N' or 'n': Do not compute the V matrix.
jobv = 'V' or 'v': Compute the V matrix.

jobq Specifies whether to compute the Q matrix or not, as follows:

jobq = 'N' or 'n': Do not compute the Q matrix.
jobq = 'Q' or 'q': Compute the Q matrix.

	m	The number of rows of the matrix A . $m \geq 0$.
	n	The number of columns of the matrices A and B . $n \geq 0$.
	p	The number of rows of the matrix B . $p \geq 0$.
	a	The m -by- n matrix A .
	lda	The leading dimension of array a in the calling program unit. $lda \geq \max(1, m)$.
	b	The p -by- n matrix B .
	ldb	The leading dimension of array b in the calling program unit. $ldb \geq \max(1, p)$.
	ldu	The leading dimension of array u in the calling program unit. $ldu \geq 1$, and if jobu = 'U' or 'u', then $ldu \geq m$.
	ldv	The leading dimension of array v in the calling program unit. $ldv \geq 1$, and if jobv = 'V' or 'v', then $ldv \geq p$.
	ldq	The leading dimension of array q in the calling program unit. $ldq \geq 1$, and if jobq = 'Q' or 'q', then $ldq \geq n$.
Working Storage	work, rwork, iwork	Arrays used for work space.
Output	k, l	On successful exit, k and l specify the dimensions of the subblocks of D_1 , D_2 , and R as described in "Purpose."
	a	On successful exit with $k+1 \leq m$, the $(k+1)$ -by- $(k+1)$ nonsingular triangular matrix R is stored in rows 1 to $k+1$ of columns $n-k-1+1$ to n of a . On successful exit with $k+1 > m$, rows 1 to m of the $(k+1)$ -by- $(k+1)$ nonsingular triangular matrix R are stored in rows 1 to m of columns $n-k-1+1$ to n of a .
	b	On successful exit with $k+1 > m$, the R_{33} block of R is stored in rows $m-k+1$ to l of columns $m+n-k-1+1$ to n of b . Destroyed if $k+1 \leq m$.
	alpha, beta	On successful exit, alpha and beta contain the generalized singular value pairs of A and B , as described in "Purpose." The α_i and β_i are not sorted into any particular order.

If $k+1 \leq m$:

$$\mathbf{alpha}(i) = \begin{cases} 1 & \text{if } i = 1, 2, \dots, k, \\ \alpha_i & \text{if } i = k+1, \dots, k+l, \\ 0 & \text{if } i = k+l+1, \dots, n. \end{cases}$$

$$\mathbf{beta}(i) = \begin{cases} 0 & \text{if } i = 1, 2, \dots, k, \\ \beta_i & \text{if } i = k+1, \dots, k+l, \\ 0 & \text{if } i = k+l+1, \dots, n. \end{cases}$$

If $k+1 > m$:

$$\alpha(i) = \begin{cases} 1 & \text{if } i = 1, 2, \dots, k, \\ \alpha_i & \text{if } i = k+1, \dots, m, \\ 0 & \text{if } i = m+1, \dots, n. \end{cases}$$

$$\beta(i) = \begin{cases} 0 & \text{if } i = 1, 2, \dots, k, \\ \beta_i & \text{if } i = k+1, \dots, m, \\ 1 & \text{if } i = m+1, \dots, k+1, \\ 0 & \text{if } i = k+1+1, \dots, n. \end{cases}$$

- u** On successful exit, if **jobu** = 'U' or 'u', the matrix U . Not referenced if **jobu** = 'N' or 'n'.
- v** On successful exit, if **jobv** = 'V' or 'v', the matrix V . Not referenced if **jobv** = 'N' or 'n'.
- q** On successful exit, if **jobq** = 'Q' or 'q', the matrix Q . Not referenced if **jobq** = 'N' or 'n'.
- info** Status response:
- info** = 0: Successful exit.
- info** < 0: If **info** = $-k$, the k -th argument had an invalid value.
- info** > 0: The algorithm did not converge.

Notes

If an error in the arguments is detected, the subprograms call error handler XERBLA, which writes an error message onto the standard error file and terminates execution. The standard version of XERBLA (see Chapter 11), may be replaced with a user-supplied version to change the error procedure. Error conditions are

jobu ≠ 'U' or 'u',
jobv ≠ 'V' or 'v',
jobq ≠ 'Q' or 'q',
m < 0,
n < 0,
p < 0,
lda < max(1,m),
ldb < max(1,p),
ldu too small,
ldv too small, and
ldq too small.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the CALL statement may be improved by coding, for example, the **jobu** argument as 'U wanted' for 'U' or 'NoU' for 'N'.

LAPACK Auxiliary Subprograms

Overview

This chapter describes selected LAPACK auxiliary subprograms. Although the auxiliary subprograms are a part of the public domain release of LAPACK, the CONVEX implementation of LAPACK does not support all of them. They are viewed as internal to the package and subject to change. The auxiliary subprograms described in this chapter, however, are supported as part of LAPACK and will exist in subsequent releases of the product. The operations covered are:

- choosing machine- or problem-dependent parameters
- computing a norm of a matrix, for matrices stored in several different formats
- reporting errors in a consistent manner

Chapter Objectives

After reading this chapter you will:

- know what a vector norm and a matrix norm is and which matrix norms can be computed by LAPACK auxiliary subprograms.
- know how blocking parameters are set and used
- understand how to use the described subprograms

What You Need to Know to Use These Subprograms

Norms of Vectors and Matrices

To use the norm-computing subprograms, you need to understand the basics of vector and matrix norms. For completeness, the following is a brief discussion of vector and matrix norms. Most standard texts on linear algebra, including the one listed in the “Supplemental Reading” section in this chapter, cover the prerequisite material in greater detail.

Definition: A *vector norm* on \mathbf{R}^n , the vector space of n -dimensional real vectors, is a function $f: \mathbf{R}^n \rightarrow \mathbf{R}$ that has the following properties:

$$\begin{array}{ll} f(x) \geq 0 & x \in \mathbf{R}^n \\ f(x) = 0 & \text{if and only if } x = 0 \\ f(x+y) \leq f(x) + f(y) & x, y \in \mathbf{R}^n \\ f(\alpha x) = |\alpha| f(x) & \alpha \in \mathbf{R}, x \in \mathbf{R}^n \end{array}$$

Such a function is denoted with a double-bar notation: $f(x) = \|x\|$. Subscripts on the double bar are used to distinguish between various norms. The most important vector norms are the 1-, 2-, and ∞ -norms, defined in Table 11-1.

The vector space of m -by- n matrices is isomorphic to the vector space of mn -dimensional vectors. Therefore, the definition of a matrix norm follows from the definition of a vector norm.

Definition: A *matrix norm* on $\mathbf{R}^{m \times n}$, the vector space of m -by- n real matrices, is a function $f: \mathbf{R}^{m \times n} \rightarrow \mathbf{R}$ that has the following properties:

$$\begin{aligned} f(A) &\geq 0 & A \in \mathbf{R}^{m \times n} \\ f(A) &= 0 & \text{if and only if } A = 0 \\ f(A+B) &\leq f(A) + f(B) & A, B \in \mathbf{R}^{m \times n} \\ f(\alpha A) &= |\alpha| f(A) & \alpha \in \mathbf{R}, A \in \mathbf{R}^{m \times n} \end{aligned}$$

As with vector norms, f is denoted with the double-bar notation: $f(A) = \|A\|$, again using subscripts to designate different matrix norms.

The formal definition of a matrix norm, given above, ignores the uses of matrices as operators in matrix-vector and matrix-matrix multiplication. Therefore, a matrix norm usually is required to satisfy several additional conditions related to such products.

Let $\|\cdot\|_\alpha$ be a vector norm on \mathbf{R}^m , $\|\cdot\|_\beta$ be a vector norm on \mathbf{R}^n , and $\|\cdot\|_{\alpha,\beta}$ be a matrix norm on $\mathbf{R}^{m \times n}$. The matrix norm is said to be *consistent* with the vector norm if

$$\|Ax\|_\beta \leq \|A\|_{\alpha,\beta} \|x\|_\alpha$$

Let $\|\cdot\|_\alpha$ be a vector norm on \mathbf{R}^m , $\|\cdot\|_\beta$ be a vector norm on \mathbf{R}^n , and $\|\cdot\|_{\alpha,\beta}$ be a matrix norm on $\mathbf{R}^{m \times n}$. The matrix norm is said to be *induced* by or *subordinate* to the vector norm if

$$\|A\|_{\alpha,\beta} = \max_{x \neq 0} \frac{\|Ax\|_\beta}{\|x\|_\alpha}$$

Finally, let $\|\cdot\|_\alpha$, $\|\cdot\|_\beta$, and $\|\cdot\|_\gamma$ be matrix norms on $\mathbf{R}^{m \times q}$, $\mathbf{R}^{m \times n}$, and $\mathbf{R}^{n \times q}$, respectively. Then the norms are *consistent* if the submultiplicative property

$$\|AB\|_\alpha \leq \|A\|_\beta \|B\|_\gamma$$

is satisfied for all $A \in \mathbf{R}^{m \times n}$ and $B \in \mathbf{R}^{n \times q}$.

The norm-computing auxiliary subprograms in LAPACK will evaluate the 1-, ∞ -, Frobenius-, or Δ -norms of a matrix stored in a variety of forms.

Table 11-1: Norms of Vectors and Matrices

Name	Vector Norm	Matrix Norm
1-norm	$\ x\ _1 = \sum_i x_i $	$\ A\ _1 = \max_j \sum_i a_{ij} $
2-norm	$\ x\ _2 = (\sum_i x_i ^2)^{1/2}$	$\ A\ _2 = \max_{x \neq 0} \ Ax\ / \ x\ $
∞ -norm	$\ x\ _\infty = \max_i x_i $	$\ A\ _\infty = \max_i \sum_j a_{ij} $
Frobenius norm	$\ x\ _F = \ x\ _2$	$\ A\ _F = (\sum_{ij} a_{ij} ^2)^{1/2}$
Δ -norm	—	$\ A\ _\Delta = \max_{ij} a_{ij} $

The Frobenius matrix norm is not subordinate to the Frobenius vector norm. The Δ matrix norm is not subordinate to any vector norm, nor is it consistent with itself as a matrix norm.

Supplemental Reading

Golub, G.H. and C.F. Van Loan. *Matrix Computations*, 2nd Edition. Baltimore, MD: The Johns Hopkins University Press. 1989.

Subprogram Descriptions

Choose Problem-Dependent Parameters for LAPACK Subprograms	
ILAENV	11-4
Return Machine-Dependent Parameters for LAPACK Subprograms	
SLAMCH, DLAMCH	11-6
Compute a Norm of a General Band Matrix	
SLANGB, DLANGB, CLANGB, ZLANGB	11-7
Compute a Norm of a General Full Matrix	
SLANGE, DLANGE, CLANGE, ZLANGE	11-10
Compute a Norm of a General Tridiagonal Matrix	
SLANGT, DLANGT, CLANGT, ZLANGT	11-12
Compute a Norm of a Symmetric or Hermitian Band Matrix	
SLANSB, DLANSB, CLANHB, CLANSB, ZLANHB, ZLANSB	11-14
Compute a Norm of a Symmetric or Hermitian Matrix Stored in Packed Form	
SLANSP, DLANSP, CLANHP, CLANSP, ZLANHP, ZLANSB	11-17
Compute a Norm of a Symmetric or Hermitian Tridiagonal Matrix	
SLANST, DLANST, CLANHT, ZLANHT	11-20
Compute a Norm of a Symmetric or Hermitian Full Matrix	
SLANSY, DLANSY, CLANHE, CLANSY, ZLANHE, ZLANSY	11-22
LAPACK Error Handler	
XERBLA	11-25

Purpose	ILAENV is called by various LAPACK subprograms to set problem-dependent parameters.
Usage	<p>LAPACK, available on C Series, Exemplar, and PA-RISC architectures:</p> <pre> CHARACTER*(*) name, opts INTEGER*4 ispec, n1, n2, n3, n4 INTEGER*4 ivalue, ILAENV ivalue = ILAENV (ispec, name, opts, n1, n2, n3, n4) </pre> <p>LAPACK8, available on C Series and Exemplar architectures:</p> <pre> CHARACTER*(*) name, opts INTEGER*8 ispec, n1, n2, n3, n4 INTEGER*8 ivalue, ILAENV ivalue = ILAENV (ispec, name, opts, n1, n2, n3, n4) </pre>
Input	<p>ispec Specifies which parameter is to be returned as the value of ILAENV, as follows:</p> <ul style="list-style-type: none"> ispec = 1: The optimal block size; if this value is 1, an unblocked algorithm will give the best performance. ispec = 2: The minimum block size for which the blocked algorithm should be used; if the usable block size is less than this value, an unblocked algorithm should be used. ispec = 3: The crossover point: in a blocked algorithm, for n less than this value, an unblocked algorithm should be used. ispec = 4: The number of shifts, used in the nonsymmetric eigenvalue subroutines. ispec = 5: The minimum column size for blocking to be used; rectangular blocks must have size at least k-by-m, where k is given by ILAENV(2,...) and m by ILAENV(5,...). ispec = 6: The crossover point for the SVD: when reducing an m-by-n matrix to bidiagonal form, if $\max(\mathbf{m}, \mathbf{n}) / \min(\mathbf{m}, \mathbf{n})$ exceeds this value, a QR factorization is used first to reduce the matrix to a triangular form. ispec = 7: The number of processors (unused in the current LAPACK implementation). ispec = 8: The crossover point for the multishift QR and QZ methods for nonsymmetric eigenvalue problems. <p>name The name of the calling subprogram in either all uppercase or all lowercase characters.</p> <p>opts The character options passed to subroutine name, concatenated into a single character string in the same order in which they appear in the argument list for subroutine name. For example, uplo = 'U', trans = 'T', and diag = 'N' for a triangular subroutine would be specified as opts = 'UTN'.</p> <p>n1, n2, n3, n4 The problem size arguments passed to subroutine name, in the same order in which they appear in the argument list for subroutine name; these may not all be required.</p>
Output	<p>ivalue The function value is the value of the requested problem-dependent parameter, or an error code, as follows:</p> <ul style="list-style-type: none"> ivalue ≥ 0: The value of the problem parameter specified by ispec. ivalue < 0: If ivalue = -<i>k</i>, the <i>k</i>-th argument had an invalid value.

Notes The following conventions have been used when calling ILAENV from LAPACK subprograms:

opts is a concatenation of all of the character options to subroutine **name**, in the same order in which they appear in the argument list for **name**, even if they are not used in determining the value of the problem-dependent parameter specified by **ispec**.

The problem size arguments **n1**, **n2**, **n3**, and **n4** are specified in the order in which they appear in the argument list for **name**. **n1** is used first, **n2** second, and so on, and unused problem size arguments are passed a value of -1.

The parameter value returned by ILAENV is checked for validity in the calling subroutine. For example, ILAENV is used to retrieve the optimal block size for STRTRI as follows:

```
NB = ILAENV (1, 'STRTRI', UPLO // DIAG, N, -1, -1, -1)
IF ( NB .LE. 1 ) NB = MAX(1, N)
```

Purpose	These subprograms are called by various LAPACK subprograms to return machine-dependent parameters, thus promoting machine independence in LAPACK.	
Usage	LAPACK, available on C Series, Exemplar, and PA-RISC architectures:	
	<pre> CHARACTER*1 name REAL*4 SLAMCH, value value = SLAMCH (name) CHARACTER*1 name REAL*8 DLAMCH, value value = DLAMCH (name) </pre>	
	LAPACK8, available on C Series and Exemplar architectures:	
	<pre> CHARACTER*1 name REAL*8 SLAMCH, value value = SLAMCH (name) </pre>	
Input	name	Specifies which machine-dependent parameter is to be returned, as follows:
	name = 'E' or 'e':	<i>eps</i> the relative machine precision: the smallest number ϵ such that $1 + \epsilon$ may be represented as a floating-point number with $1 + \epsilon > 1$.
	name = 'S' or 's':	<i>sfmin</i> the safe minimum: the smallest number such that $1/sfmin$ does not overflow.
	name = 'B' or 'b':	<i>base</i> the base of the machine, $base = 2.0$ on all CONVEX machine types.
	name = 'P' or 'p':	<i>prec</i> $eps \times base$.
	name = 'N' or 'n':	<i>t</i> the number of base-2 digits in the mantissa.
	name = 'R' or 'r':	<i>rnd</i> 1.0 on machines where rounding occurs on addition; zero otherwise. $rnd = 1.0$ on all CONVEX machine types.
	name = 'M' or 'm':	<i>emin</i> the smallest exponent before underflow.
	name = 'U' or 'u':	<i>rmin</i> the underflow threshold: $base^{emin-1}$.
	name = 'L' or 'l':	<i>emax</i> the largest exponent before overflow.
	name = 'O' or 'o':	<i>rmin</i> the overflow threshold: $base^{emax} \times (1 - \epsilon)$.
Output	value	The function value is the value of the requested machine-dependent parameter.
Notes	Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the function reference may be improved by coding, for example, the name argument as 'Eps' for 'E' or 'Safemin' for 'S'.	

Compute Norm of General Band Matrix**SLANGB/DLANGB/.../ZLANGB**

Purpose These subprograms compute a norm of an m -by- n general band matrix A . A band matrix is a matrix whose nonzero elements all lie near the principal diagonal. Specifically, $a_{ij} = 0$ if $i-j > kl$ or $j-i > ku$ for some integers kl and ku . The smallest such kl and ku for a given matrix are called the lower and upper bandwidths, respectively, and $k = kl+ku+1$ is the total bandwidth.

Matrix Storage Because it is not necessary to store or operate on the zeros outside the band of A , you need only provide the elements within the band of A . Compared to storing the entire matrix, this can save memory if $kl+ku+1 < n$.

The following example illustrates the storage of general band matrices. Consider the following matrix A of order $n = 9$ and lower and upper bandwidths $kl = 2$ and $ku = 3$, respectively:

11	12	13	14	0	0	0	0	0
21	22	23	24	25	0	0	0	0
31	32	33	34	35	36	0	0	0
0	42	43	44	45	46	47	0	0
0	0	53	54	55	56	57	58	0
0	0	0	64	65	66	67	68	69
0	0	0	0	75	76	77	78	79
0	0	0	0	0	86	87	88	89
0	0	0	0	0	0	97	98	99

A is given in an array **ab** with at least $kl+ku+1 = 6$ rows and $n = 9$ columns as follows:

*	*	*	14	25	36	47	58	69
*	*	13	24	35	46	57	68	79
*	12	23	34	45	56	67	78	89
11	22	33	44	55	66	77	88	99
21	32	43	54	65	76	87	98	*
31	42	53	64	75	86	97	*	*

The asterisks in the ku -by- ku triangle at the upper left corner and in the kl -by- kl triangle at the lower right corner represent elements of **ab** that are not referenced. Thus, if a_{ij} is an element within the band of A , then it is stored in $ab(ku+1+i-j, j)$. Therefore, the columns of A are stored in the columns of **ab**, and the diagonals of A are stored in the rows of **ab**, such that the principal diagonal is stored in row $ku+1$ of **ab**.

Note that this storage format omits the first kl rows reserved for fill-in in the general band storage for **_GBSV** and **_GBTRF**.

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```

CHARACTER*1 norm
INTEGER*4    kl, ku, ldab, n
REAL*4      ab(ldab, n), work(n)
REAL*4      anorm, SLANGB
anorm = SLANGB (norm, n, kl, ku, ab, ldab, work)

```

CHARACTER*1 norm
INTEGER*4 kl, ku, ldab, n
REAL*8 ab(ldab, n), work(n)
REAL*8 anorm, DLANGB
 anorm = DLANGB (norm, n, kl, ku, ab, ldab, work)

CHARACTER*1 norm
INTEGER*4 kl, ku, ldab, n
REAL*4 rwork(n)
COMPLEX*8 ab(ldab, n)
REAL*4 anorm, CLANGB
 anorm = CLANGB (norm, n, kl, ku, ab, ldab, rwork)

CHARACTER*1 norm
INTEGER*4 kl, ku, ldab, n
REAL*8 rwork(n)
COMPLEX*16 ab(ldab, n)
REAL*8 anorm, ZLANGB
 anorm = ZLANGB (norm, n, kl, ku, ab, ldab, rwork)

LAPACK8, available on C Series and Exemplar architectures:

CHARACTER*1 norm
INTEGER*8 kl, ku, ldab, n
REAL*8 ab(ldab, n), work(n)
REAL*8 anorm, SLANGB
 anorm = SLANGB (norm, n, kl, ku, ab, ldab, work)

CHARACTER*1 norm
INTEGER*8 kl, ku, ldab, n
REAL*8 rwork(n)
COMPLEX*16 ab(ldab, n)
REAL*8 anorm, CLANGB
 anorm = CLANGB (norm, n, kl, ku, ab, ldab, rwork)

Input **norm** Specifies which norm is to be computed, as follows:

norm = 'F', 'f', 'E', or 'e': Compute $\|A\|_F$ = the Frobenius norm.
norm = 'I' or 'i': Compute $\|A\|_\infty$ = maximum row sum.
norm = '1', 'O', or 'o': Compute $\|A\|_1$ = maximum column sum.
norm = 'M' or 'm': Compute $\max(|A_{ij}|)$.

n The order of the matrix A . $n \geq 0$.

kl The number of subdiagonals within the band of A . $kl \geq 0$.

ku The number of superdiagonals within the band of A . $ku \geq 0$.

ab The matrix A in band storage, in the first $kl+ku+1$ rows. The j -th column of A is stored in the j -th column of array **ab** as follows: $\mathbf{ab}(ku+1+i-j, j) = A(i, j)$ for $\max(1, j-ku) \leq i \leq \min(n, j+kl)$

ldab The leading dimension of array **ab** in the calling program unit. $ldab \geq kl+ku+1$.

Working Storage	work, rwork	Arrays used for work space. Not referenced unless norm = 'I' or 'i'.
Output	anorm	The function value is the value of the requested norm of A .
Notes		Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the CALL statement may be improved by coding the norm argument as 'Frobenius' for 'F', 'Infinity-Norm' for 'I', 'One-norm' or '1-norm' for 'O', or 'Max-Element' for 'M'.

Purpose These subprograms compute a norm of a general m -by- n matrix A .

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

CHARACTER*1 norm
INTEGER*4 lda, m, n
REAL*4 a(lda, n), work(n)
REAL*4 anorm, SLANGE
anorm = SLANGE (norm, m, n, a, lda, work)

CHARACTER*1 norm
INTEGER*4 lda, m, n
REAL*8 a(lda, n), work(n)
REAL*8 anorm, DLANGE
anorm = DLANGE (norm, m, n, a, lda, work)

CHARACTER*1 norm
INTEGER*4 lda, m, n
REAL*4 rwork(n)
COMPLEX*8 a(lda, n)
REAL*4 anorm, CLANGE
anorm = CLANGE (norm, m, n, a, lda, rwork)

CHARACTER*1 norm
INTEGER*4 lda, m, n
REAL*8 rwork(n)
COMPLEX*16 a(lda, n)
REAL*8 anorm, ZLANGE
anorm = ZLANGE (norm, m, n, a, lda, rwork)

LAPACK8, available on C Series and Exemplar architectures:

CHARACTER*1 norm
INTEGER*8 lda, m, n
REAL*8 a(lda, n), work(n)
REAL*8 anorm, SLANGE
anorm = SLANGE (norm, m, n, a, lda, work)

CHARACTER*1 norm
INTEGER*8 lda, m, n
REAL*8 rwork(n)
COMPLEX*16 a(lda, n)
REAL*8 anorm, CLANGE
anorm = CLANGE (norm, m, n, a, lda, rwork)

Input norm Specifies which norm is to be computed, as follows:

norm = 'F', 'f', 'E', or 'e': Compute $\|A\|_F$ = the Frobenius norm.
norm = 'I' or 'i': Compute $\|A\|_\infty$ = maximum row sum.
norm = '1', 'O', or 'o': Compute $\|A\|_1$ = maximum column sum.
norm = 'M' or 'm': Compute $\max(|A_{ij}|)$.

m The number of rows of the matrix A . $n \geq 0$.

	n	The number of columns of the matrix A . $n \geq 0$.
	a	The m -by- n matrix A .
	lda	The leading dimension of array a in the calling program unit. $lda \geq \max(1,m)$.
Working Storage	work, rwork	Arrays used for work space. Not referenced unless norm = 'I' or 'i'.
Output	anorm	The function value is the value of the requested norm of A .
Notes		Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the CALL statement may be improved by coding the norm argument as 'Frobenius' for 'F', 'Infinity-Norm' for 'I', 'One-norm' or '1-norm' for 'O', or 'Max-Element' for 'M'.

Purpose These subprograms compute a norm of a general tridiagonal matrix A . A matrix $A = (a_{ij})$ is tridiagonal if its nonzero elements lie only on the principal diagonal ($i = j$), the subdiagonal ($i = j+1$), and the superdiagonal ($i = j-1$) of the matrix.

Matrix Storage The following example illustrates the storage of general tridiagonal matrices. Consider the following tridiagonal matrix of order $n = 7$:

11	12	0	0	0	0	0
21	22	23	0	0	0	0
0	32	33	34	0	0	0
0	0	43	44	45	0	0
0	0	0	54	55	56	0
0	0	0	0	65	66	67
0	0	0	0	0	76	77

The subdiagonal is stored in array **dl**, the principal diagonal is stored in array **d**, and the superdiagonal is stored in array **du**, as follows:

i	dl (i)	d (i)	du (i)
1	21	11	12
2	32	22	23
3	43	33	34
4	54	44	45
5	65	55	56
6	76	66	67
7		77	

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```

CHARACTER*1 norm
INTEGER*4    n
REAL*4      d(n), dl(n-1), du(n-1)
REAL*4      anorm, SLANGT
anorm = SLANGT (norm, n, dl, d, du)

```

```

CHARACTER*1 norm
INTEGER*4    n
REAL*8      d(n), dl(n-1), du(n-1)
REAL*8      anorm, DLANGT
anorm = DLANGT (norm, n, dl, d, du)

```

```

CHARACTER*1 norm
INTEGER*4    n
COMPLEX*8   d(n), dl(n-1), du(n-1)
REAL*4      anorm, CLANGT
anorm = CLANGT (norm, n, dl, d, du)

```

```

CHARACTER*1 norm
INTEGER*4    n
COMPLEX*16  d(n), dl(n-1), du(n-1)
REAL*8      anorm, ZLANGT
anorm = ZLANGT (norm, n, dl, d, du)

```

LAPACK8, available on C Series and Exemplar architectures:

```

CHARACTER*1 norm
INTEGER*8    n
REAL*8       d(n), dl(n-1), du(n-1)
REAL*8       anorm, SLANGT
anorm = SLANGT (norm, n, dl, d, du)

```

```

CHARACTER*1 norm
INTEGER*8    n
COMPLEX*16  d(n), dl(n-1), du(n-1)
REAL*8       anorm, CLANGT
anorm = CLANGT (norm, n, dl, d, du)

```

Input **norm** Specifies which norm is to be computed, as follows:

norm = 'F', 'f', 'E', or 'e': Compute $\|A\|_F$ = the Frobenius norm.

norm = 'I' or 'i': Compute $\|A\|_\infty$ = maximum row sum.

norm = '1', 'O', or 'o': Compute $\|A\|_1$ = maximum column sum.

norm = 'M' or 'm': Compute $\max(|A_{ij}|)$.

n The order of the matrix A . $n \geq 0$.

dl The $n-1$ subdiagonal elements of A .

d The diagonal elements of A .

du The $n-1$ superdiagonal elements of A .

Output **anorm** The function value is the value of the requested norm of A .

Notes Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding the **norm** argument as 'Frobenius' for 'F', 'Infinity-Norm' for 'I', 'One-norm' or '1-norm' for 'O', or 'Max-Element' for 'M'.

Purpose These subprograms compute a norm of a real or complex symmetric or complex Hermitian band matrix A . A real matrix is symmetric if $A = A^T$, its transpose; a complex matrix is Hermitian if $A = A^*$, its conjugate transpose.

Tridiagonal matrices are the special case $kd = 1$. They can be handled more efficiently by the LAPACK subprograms SLANST, DLANST, CLANHT, and ZLANHT.

The structure of A is indicated by the name of the subprogram used:

SLANSB	or DLANSB	A is a real symmetric matrix.
CLANSB	or ZLANSB	A is a complex symmetric matrix.
CLANHB	or ZLANHB	A is a complex Hermitian matrix.

Matrix Storage Because it is not necessary to store or operate on the zeros outside the band of A , and since either triangle of A may be obtained from the other, you need only provide the band within one triangle of A . Compared to storing the entire matrix, this can save memory in two ways: only the elements within the band are stored, and of them, only the upper or the lower triangle.

The following examples illustrate the storage of symmetric or Hermitian band matrices. Consider the following matrix A of order $n = 7$ and half bandwidth $kd = 2$:

11	12	13	0	0	0	0
12	22	23	24	0	0	0
13	23	33	34	35	0	0
0	24	34	44	45	46	0
0	0	35	45	55	56	57
0	0	0	46	56	66	67
0	0	0	0	57	67	77

Upper triangular storage. The upper triangle of A is stored in an array \mathbf{ab} with at least $kd+1 = 3$ rows and 7 columns as follows:

*	*	13	24	35	46	57
*	12	23	34	45	56	67
11	22	33	44	55	66	77

The asterisks represent elements in the kd -by- kd triangle at the upper left corner of \mathbf{ab} that are not referenced. Thus, if a_{ij} is an element within the band of the upper triangle of A , it is stored in $\mathbf{ab}(kd+1+i-j, j)$. Therefore, the columns of the upper triangle of A are stored in the columns of \mathbf{ab} , and the diagonals of the upper triangle of A are stored in the rows of \mathbf{ab} .

Lower triangular storage. The lower triangle of A is stored in the array \mathbf{ab} as follows:

11	22	33	44	55	66	77
12	23	34	45	56	67	*
13	24	35	46	57	*	*

Continued

SLANSB/DLANSB/CLANHB/CLANSB/ZLANHB/ZLANSB

The asterisks represent elements in the kd -by- kd triangle at the lower right corner of \mathbf{ab} that are not referenced. Thus, if a_{ij} is an element within the band of the lower triangle of A , it is stored in $\mathbf{ab}(kd+1+i-j, j)$. Therefore, the columns of the lower triangle of A are stored in the columns of \mathbf{ab} , and the diagonals of the lower triangle of A are stored in the rows of \mathbf{ab} .

Usage

LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

CHARACTER*1 norm, uplo
INTEGER*4 kd, ldab, n
REAL*4 ab(ldab, n), work(n)
REAL*4 anorm, SLANSB
 anorm = SLANSB (norm, uplo, n, kd, ab, ldab, work)

CHARACTER*1 norm, uplo
INTEGER*4 kd, ldab, n
REAL*8 ab(ldab, n), work(n)
REAL*8 anorm, DLANSB
 anorm = DLANSB (norm, uplo, n, kd, ab, ldab, work)

CHARACTER*1 norm, uplo
INTEGER*4 kd, ldab, n
REAL*4 rwork(n)
COMPLEX*8 ab(ldab, n)
REAL*4 anorm, CLANHB
 anorm = CLANHB (norm, uplo, n, kd, ab, ldab, rwork)

CHARACTER*1 norm, uplo
INTEGER*4 kd, ldab, n
REAL*4 rwork(n)
COMPLEX*8 ab(ldab, n)
REAL*4 anorm, CLANSB
 anorm = CLANSB (norm, uplo, n, kd, ab, ldab, rwork)

CHARACTER*1 norm, uplo
INTEGER*4 kd, ldab, n
REAL*8 rwork(n)
COMPLEX*16 ab(ldab, n)
REAL*8 anorm, ZLANHB
 anorm = ZLANHB (norm, uplo, n, kd, ab, ldab, rwork)

CHARACTER*1 norm, uplo
INTEGER*4 kd, ldab, n
REAL*8 rwork(n)
COMPLEX*16 ab(ldab, n)
REAL*8 anorm, ZLANSB
 anorm = ZLANSB (norm, uplo, n, kd, ab, ldab, rwork)

LAPACK8, available on C Series and Exemplar architectures:

CHARACTER*1 norm, uplo
INTEGER*8 kd, ldab, n
REAL*8 ab(ldab, n), work(n)
REAL*8 anorm, SLANSB
 anorm = SLANSB (norm, uplo, n, kd, ab, ldab, work)

CHARACTER*1 norm, uplo
INTEGER*8 kd, ldab, n
REAL*8 rwork(n)
COMPLEX*16 ab(ldab, n)
REAL*8 anorm, CLANHB
 anorm = CLANHB (norm, uplo, n, kd, ab, ldab, rwork)

CHARACTER*1 norm, uplo
INTEGER*8 kd, ldab, n
REAL*8 rwork(n)
COMPLEX*16 ab(ldab, n)
REAL*8 anorm, CLANSB
 anorm = CLANSB (norm, uplo, n, kd, ab, ldab, rwork)

- Input**
- norm** Specifies which norm is to be computed, as follows:
- norm** = 'F', 'f', 'E', or 'e': Compute $\|A\|_F$ = the Frobenius norm.
norm = 'I' or 'i': Compute $\|A\|_\infty$ = maximum row sum.
norm = '1', 'O', or 'o': Compute $\|A\|_1$ = maximum column sum.
norm = 'M' or 'm': Compute $\max(|A_{ij}|)$.
- uplo** Specifies whether the upper or lower triangular part of the symmetric or Hermitian matrix A is stored, as follows:
- uplo** = 'U' or 'u': The upper triangular part of A is stored.
uplo = 'L' or 'l': The lower triangular part of A is stored.
- n** The order of the matrix A . $n \geq 0$.
- kd** The number of super-diagonals of the matrix A if **uplo** = 'U' or 'u', or the number of sub-diagonals if **uplo** = 'L' or 'l'. $kd \geq 0$.
- ab** The upper or lower triangle of the symmetric or Hermitian band matrix A , stored in the first $kd+1$ rows of the array. The j -th column of A is stored in the j -th column of array **ab** as follows:
- If **uplo** = 'U' or 'u', $ab(kd+1+i-j, j) = A(i, j)$ for $\max(1, j-kd) \leq i \leq j$;
- If **uplo** = 'L' or 'l', $ab(1+i-j, j) = A(i, j)$ for $j \leq i \leq \min(n, j+kd)$.
- ldab** The leading dimension of array **ab** in the calling program unit. $ldab \geq kd+1$.
- Working Storage**
- work, rwork** Arrays used for work space. Not referenced unless **norm** = '1' or 'I' or 'i' or 'O' or 'o'.
- Output**
- anorm** The function value is the value of the requested norm of A .
- Notes**
- Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding the **norm** argument as 'Frobenius' for 'F', 'Infinity-Norm' for 'I', 'One-norm' or '1-norm' for 'O', or 'Max-Element' for 'M'.

Compute Norm of Symmetric or Hermitian Packed Matrix**SLANSF/...**

Purpose These subprograms compute a norm of a real or complex symmetric or complex Hermitian matrix A that is stored in an array in packed form.

A matrix is symmetric if $A = A^T$, its transpose; a matrix is Hermitian if $A = A^*$, its conjugate transpose.

The structure of A is indicated by the name of the subprogram used:

SLANSF or DLANSF A is a real symmetric packed matrix.
 CLANSF or ZLANSF A is a complex symmetric packed matrix.
 CLANHF or ZLANHF A is a complex Hermitian packed matrix.

Matrix Storage Because either triangle of A may be obtained from the other, you need only provide one triangle of A , either the upper or the lower triangle. Compared to storing the entire matrix, you save memory by supplying that triangle stored column-by-column in packed form in a 1-dimensional array.

The following examples illustrate the packed storage of symmetric or Hermitian matrices.

Upper triangular storage. If the upper triangle of A is

11	12	13	14
	22	23	24
		33	34
			44

then A is packed column-by-column into an array **ap** as follows:

k	1	2	3	4	5	6	7	8	9	10
ap(k)	11	12	22	13	23	33	14	24	34	44

Upper triangular matrix element a_{ij} is stored in array element **ap**($i+j \times (j-1)/2$).

Lower triangular storage. If the lower triangle of A is

11				
21	22			
31	32	33		
41	42	43	44	

then A is packed column-by-column into an array **ap** as follows:

k	1	2	3	4	5	6	7	8	9	10
ap(k)	11	21	31	41	22	32	42	33	43	44

Lower triangular matrix element a_{ij} is stored in array element **ap**($i+(j-1) \times (2n-j)/2$).

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

CHARACTER*1 norm, uplo
INTEGER*4 n
REAL*4 ap((n*(n+1))/2), work(n)
REAL*4 anorm, SLANSF
anorm = SLANSF (norm, uplo, n, ap, work)

CHARACTER*1 norm, uplo
INTEGER*4 n
REAL*8 ap((n*(n+1))/2), work(n)
REAL*8 anorm, DLANSF
 anorm = DLANSF (norm, uplo, n, ap, work)

CHARACTER*1 norm, uplo
INTEGER*4 n
REAL*4 rwork(n)
COMPLEX*8 ap((n*(n+1))/2)
REAL*4 anorm, CLANSF
 anorm = CLANSF (norm, uplo, n, ap, rwork)

CHARACTER*1 norm, uplo
INTEGER*4 n
REAL*4 rwork(n)
COMPLEX*8 ap((n*(n+1))/2)
REAL*4 anorm, CLANSF
 anorm = CLANSF (norm, uplo, n, ap, rwork)

CHARACTER*1 norm, uplo
INTEGER*4 n
REAL*8 rwork(n)
COMPLEX*16 ap((n*(n+1))/2)
REAL*8 anorm, ZLANHP
 anorm = ZLANHP (norm, uplo, n, ap, rwork)

CHARACTER*1 norm, uplo
INTEGER*4 n
REAL*8 rwork(n)
COMPLEX*16 ap((n*(n+1))/2)
REAL*8 anorm, CLANSF
 anorm = ZLANSP (norm, uplo, n, ap, rwork)

LAPACK8, available on C Series and Exemplar architectures:

CHARACTER*1 norm, uplo
INTEGER*8 n
REAL*8 ap((n*(n+1))/2), work(n)
REAL*8 anorm, SLANSF
 anorm = SLANSF (norm, uplo, n, ap, work)

CHARACTER*1 norm, uplo
INTEGER*8 n
REAL*8 rwork(n)
COMPLEX*16 ap((n*(n+1))/2)
REAL*8 anorm, CLANSF
 anorm = CLANSF (norm, uplo, n, ap, rwork)

CHARACTER*1 norm, uplo
INTEGER*8 n
REAL*8 rwork(n)
COMPLEX*16 ap((n*(n+1))/2)
REAL*8 anorm, CLANSF
 anorm = CLANSF (norm, uplo, n, ap, rwork)

Input	norm	Specifies which norm is to be computed, as follows: norm = 'F', 'f', 'E', or 'e': Compute $\ A\ _F$ = the Frobenius norm. norm = 'I' or 'i': Compute $\ A\ _\infty$ = maximum row sum. norm = '1', 'O', or 'o': Compute $\ A\ _1$ = maximum column sum. norm = 'M' or 'm': Compute $\max(A_{ij})$.
	uplo	Specifies whether the upper or lower triangular part of the symmetric or Hermitian matrix A is stored, as follows: uplo = 'U' or 'u': The upper triangular part of A is stored. uplo = 'L' or 'l': The lower triangular part of A is stored.
	n	The order of the matrix A . $n \geq 0$.
	ap	The upper or lower triangular part of the symmetric or Hermitian matrix A , packed columnwise in a linear array as follows: If uplo = 'U' or 'u', $ap(i + (j-1) \times j/2) = A(i, j)$ for $1 \leq i \leq j$; If uplo = 'L' or 'l', $ap(i + (j-1) \times (2n-j)/2) = A(i, j)$ for $j \leq i \leq n$.
Working Storage	work, rwork	Arrays used for work space. Not referenced unless norm = '1' or 'I' or 'i' or 'O' or 'o'.
Output	anorm	The function value is the value of the requested norm of A .
Notes		Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the CALL statement may be improved by coding the norm argument as 'Frobenius' for 'F', 'Infinity-Norm' for 'I', 'One-norm' or '1-norm' for 'O', or 'Max-Element' for 'M'.

Purpose These subprograms compute a norm of a real symmetric or complex Hermitian tridiagonal matrix A . A real matrix is symmetric if $A = A^T$, its transpose; a complex matrix is Hermitian if $A = A^*$, its conjugate transpose.

A matrix $A = (a_{ij})$ is tridiagonal if its nonzero elements lie only on the principal diagonal ($i = j$), the subdiagonal ($i = j+1$), and the superdiagonal ($i = j-1$) of the matrix.

Matrix Storage The following example illustrates the storage of symmetric or Hermitian tridiagonal matrices. Consider the following symmetric tridiagonal matrix of order $n = 7$:

11	21	0	0	0	0	0
21	22	32	0	0	0	0
0	32	33	43	0	0	0
0	0	43	44	54	0	0
0	0	0	54	55	65	0
0	0	0	0	65	66	76
0	0	0	0	0	76	77

The subdiagonal is stored in array **e** and the principal diagonal is stored in array **d**, as follows:

<i>i</i>	e (<i>i</i>)	d (<i>i</i>)
1	21	11
2	32	22
3	43	33
4	54	44
5	65	55
6	76	66
7		77

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```

CHARACTER*1 norm
INTEGER*4    n
REAL*4      d(n), e(n-1)
REAL*4      anorm, SLANST
anorm = SLANST (norm, n, d, e)

```

```

CHARACTER*1 norm
INTEGER*4    n
REAL*8      d(n), e(n-1)
REAL*8      anorm, DLANST
anorm = DLANST (norm, n, d, e)

```

```

CHARACTER*1 norm
INTEGER*4    n
REAL*4      d(n)
COMPLEX*8   e(n-1)
REAL*4      anorm, CLANHT
anorm = CLANHT (norm, n, d, e)

```

CHARACTER*1 norm
INTEGER*4 n
REAL*8 d(n)
COMPLEX*16 e(n-1)
REAL*8 anorm, ZLANHT
 anorm = ZLANHT (norm, n, d, e)

LAPACK8, available on C Series and Exemplar architectures:

CHARACTER*1 norm
INTEGER*8 n
REAL*8 d(n), e(n-1)
REAL*8 anorm, SLANST
 anorm = SLANST (norm, n, d, e)

CHARACTER*1 norm
INTEGER*8 n
REAL*8 d(n)
COMPLEX*16 e(n-1)
REAL*8 anorm, CLANHT
 anorm = CLANHT (norm, n, d, e)

Input **norm** Specifies which norm is to be computed, as follows:

norm = 'F', 'f', 'E', or 'e': Compute $\|A\|_F$ = the Frobenius norm.
norm = 'I' or 'i': Compute $\|A\|_\infty$ = maximum row sum.
norm = '1', 'O', or 'o': Compute $\|A\|_1$ = maximum column sum.
norm = 'M' or 'm': Compute $\max(|A_{ij}|)$.

n The order of the matrix A . $n \geq 0$.

d The n diagonal elements of the tridiagonal matrix A .

e The $n-1$ subdiagonal elements of the tridiagonal matrix A .

Output **anorm** The function value is the value of the requested norm of A .

Notes Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding the **norm** argument as 'Frobenius' for 'F', 'Infinity-Norm' for 'I', 'One-norm' or '1-norm' for 'O', or 'Max-Element' for 'M'.

Purpose These subprograms compute a norm of a real or complex symmetric or complex Hermitian matrix A .

A matrix is symmetric if $A = A^T$, its transpose; a matrix is Hermitian if $A = A^*$, its conjugate transpose.

The structure of A is indicated by the name of the subprogram used:

SLANSY or DLANSY A is a real symmetric matrix.
 CLANSY or ZLANSY A is a complex symmetric matrix.
 CLANHE or ZLANHE A is a complex Hermitian matrix.

Matrix Storage Because either triangle of A may be obtained from the other, you need only provide one triangle of A . You may supply either the upper or the lower triangle of A , in a two-dimensional array large enough to hold the entire matrix. The other triangle of the array is not referenced.

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```
CHARACTER*1 norm, uplo
INTEGER*4   lda, n
REAL*4      a(lda, n), work(n)
REAL*4      anorm, SLANSY
anorm = SLANSY (norm, uplo, n, a, lda, work)
```

```
CHARACTER*1 norm, uplo
INTEGER*4   lda, n
REAL*8      a(lda, n), work(n)
REAL*8      anorm, DLANSY
anorm = DLANSY (norm, uplo, n, a, lda, work)
```

```
CHARACTER*1 norm, uplo
INTEGER*4   lda, n
REAL*4      rwork(n)
COMPLEX*8   a(lda, n)
REAL*4      anorm, CLANHE
anorm = CLANHE (norm, uplo, n, a, lda, rwork)
```

```
CHARACTER*1 norm, uplo
INTEGER*4   lda, n
REAL*4      rwork(n)
COMPLEX*8   a(lda, n)
REAL*4      anorm, CLANSY
anorm = CLANSY (norm, uplo, n, a, lda, rwork)
```

```
CHARACTER*1 norm, uplo
INTEGER*4   lda, n
REAL*8      rwork(n)
COMPLEX*16  a(lda, n)
REAL*8      anorm, ZLANHE
anorm = ZLANHE (norm, uplo, n, a, lda, rwork)
```

Continued

SLANSY/DLANSY/CLANHE/CLANSY/ZLANHE/ZLANSY

CHARACTER*1 norm, uplo
INTEGER*4 lda, n
REAL*8 rwork(n)
COMPLEX*16 a(lda, n)
REAL*8 anorm, ZLANSY
 anorm = ZLANSY (norm, uplo, n, a, lda, rwork)

LAPACK8, available on C Series and Exemplar architectures:

CHARACTER*1 norm, uplo
INTEGER*8 lda, n
REAL*8 a(lda, n), work(n)
REAL*8 anorm, SLANSY
 anorm = SLANSY (norm, uplo, n, a, lda, work)

CHARACTER*1 norm, uplo
INTEGER*8 lda, n
REAL*8 rwork(n)
COMPLEX*16 a(lda, n)
REAL*8 anorm, CLANHE
 anorm = CLANHE (norm, uplo, n, a, lda, rwork)

CHARACTER*1 norm, uplo
INTEGER*8 lda, n
REAL*8 rwork(n)
COMPLEX*16 a(lda, n)
REAL*8 anorm, CLANSY
 anorm = CLANSY (norm, uplo, n, a, lda, rwork)

Input	norm	Specifies which norm is to be computed, as follows: norm = 'F', 'f', 'E', or 'e': Compute $\ A\ _F$ = the Frobenius norm. norm = 'I' or 'i': Compute $\ A\ _\infty$ = maximum row sum. norm = '1', 'O', or 'o': Compute $\ A\ _1$ = maximum column sum. norm = 'M' or 'm': Compute $\max(A_{ij})$.
	uplo	Specifies whether the upper or lower triangular part of the symmetric or Hermitian matrix A is stored, as follows: uplo = 'U' or 'u': The upper triangular part of A is stored. uplo = 'L' or 'l': The lower triangular part of A is stored.
	n	The order of the matrix A . $n \geq 0$.
	a	The symmetric or Hermitian matrix A , as follows: If uplo = 'U' or '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' or '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	The leading dimension of array a in the calling program unit. $lda \geq \max(1,n)$.
Working Storage	work, rwork	Arrays used for work space. Not referenced unless norm = '1' or 'I' or 'i' or 'O' or 'o'.
Output	anorm	The function value is the value of the requested norm of A .
Notes		Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the CALL statement may be improved by coding the norm argument as 'Frobenius' for 'F', 'Infinity-Norm' for 'I', 'One-norm' or '1-norm' for 'O', or 'Max-Element' for 'M'.

Error Handler**XERBLA**

Purpose This subprogram is the error handler for many LAPACK subprograms, as indicated in the "Notes" section in the applicable subprogram descriptions. As supplied in LAPACK, XERBLA writes the following error message onto the standard error file:

```
*****
* XERBLA: subprogram name called with invalid value of argument number iarg *
*****
```

where *name* is the name of the subprogram in which the error was detected, and *iarg* is the argument number of the offending argument. For example, in SGBSV, *n* is argument number 1 and *kl* is argument number 2. If the main program is in Fortran, and executed on a Convex C Series machine, a call traceback is also written onto the standard error file (XERBLA does not write a call traceback when used on an Exemplar or other PA-RISC machine). XERBLA then terminates execution with a nonzero exit status.

You may supply a version of XERBLA that alters this action. All LAPACK subprograms that call XERBLA have a **RETURN** statement after the **CALL XERBLA** statement, so if your version of XERBLA exits with a **RETURN** statement, you could detect the error by examining the **info** flag after each LAPACK subprogram call. Other subprograms, such as the Level 2 and 3 BLAS, also call XERBLA. All BLAS, VECLIB, and SCILIB subprograms that call XERBLA also follow the **CALL XERBLA** statement with a **RETURN** statement. However, many of those subprograms do not have a status response variable such as **info** in their argument list to alert the caller. If you write an XERBLA that does not end with a **STOP** statement, you need some other mechanism to detect errors occurring in non-LAPACK subprograms. One such mechanism is a flag in a common block that is set by your XERBLA and tested by the calling program after calls where errors could be detected.

Usage LAPACK, available on C Series, Exemplar, and PA-RISC architectures:

```
CHARACTER*6 name
INTEGER*4 iarg
CALL XERBLA (name, iarg)
```

LAPACK8, available on C Series and Exemplar architectures:

```
CHARACTER*6 name
INTEGER*8 iarg
CALL XERBLA (name, iarg)
```

Input **name** The name of the subprogram in which the error was detected.

iarg The number of the argument that was found to be in error.

Notes This subprogram conforms to specifications of the Level 2 and 3 BLAS and LAPACK.

Index

A

accessing LAPACK 1-2
accuracy, improve 3-6, 3-14, 3-21, 3-26, 3-33,
3-39, 3-45, 3-49, 3-55
Ada, calling LAPACK from 1-1
arithmetic format 1-5
ASAP, automatic self allocating processors 1-3
automatic self allocating processors (ASAP) 1-3
auxiliary subprograms 1-9, 11-1

B

backward error bound 3-2
backward stability 3-3
band matrix eigenvalues, Hermitian 7-10, 7-13,
8-15
band matrix eigenvalues, symmetric 7-10, 7-13,
8-15
band matrix eigenvectors, Hermitian 7-10, 7-13,
8-15
band matrix eigenvectors, symmetric 7-10, 7-13,
8-15
band matrix, factor general 4-11
band matrix, factor positive definite 4-34
band matrix, general 4-8
band matrix generalized eigenvalues, Hermitian
9-11
band matrix generalized eigenvalues, symmetric
9-11
band matrix generalized eigenvectors, Hermitian
9-11
band matrix generalized eigenvectors, symmetric
9-11
band matrix, norm of general 11-7
band matrix, norm of Hermitian 11-14
band matrix, norm of symmetric 11-14
band matrix, positive definite 4-31
band matrix, solve general 2-3, 3-6, 4-14
band matrix, solve positive definite 2-10, 3-26,
4-37
band matrix, solve triangular 4-91
band matrix, triangular 4-87
Basic Linear Algebra Subprograms (BLAS) 1-3
BEGIN_TASKS compiler directive 1-4
bibliography xxi
BLAS 1-3, 11-25
BLAS, Extended 1-3
BLAS, Level 1 1-3
BLAS, Level 2 1-3
BLAS, Level 3 1-3
block size 11-4, 11-6
bounds, error 3-6, 3-14, 3-21, 3-26, 3-33, 3-39,
3-45, 3-49, 3-55
Bunch-Kaufman factorization 4-66, 4-78

C

C, calling LAPACK from 1-1
Calling LAPACK from Ada 1-1
Calling LAPACK from C 1-1
-cfc compiler option 1-2, 1-14

CGBCON 4-8
CGBEQU 4-110
CGBRFS 4-110
CGBSV 2-3
CGBSVX 3-6
CGBTRF 4-11
CGBTRS 4-14
CGECON 4-16
CGEEQU 4-110
CGEES 7-3
CGEESX 8-3
CGEEV 7-7
CGEEVX 8-9
CGEGS 9-3
CGEGV 9-7
CGELQF 6-4
CGELS 5-4
CGELSS 5-7
CGELSX 5-10
CGEMMS 1-3
CGEQLF 6-6
CGEQPF 6-8
CGEQRF 6-11
CGERFS 4-110
CGERQF 6-13
CGESV 2-6
CGESVD 10-2
CGESVX 3-14
CGETRF 4-18
CGETRI 4-20
CGETRS 4-22
CGGGLM 5-13
CGGLSE 5-15
CGGQRF 6-15
CGGRQF 6-19
CGGSVD 10-5
CGTCON 4-24
CGTRFS 4-110
CGTSV 2-8
CGTSVX 3-21
CGTTRF 4-27
CGTTRS 4-29
CHBEV 7-10
CHBEVD 7-13
CHBEVX 8-15
CHBGV 9-11
CHECON 4-75
CHEEV 7-29
CHEEVD 7-32
CHEEVX 8-28
CHEGV 9-19
CHERFS 4-110
CHESV 2-25
CHESVX 3-55
CHETRF 4-78
CHETRI 4-81
CHETRS 4-84
Cholesky factorization 4-34, 4-41, 4-49, 4-59
CHPCON 4-63
CHPEV 7-17

- CHPEVD 7-20
 CHPEVX 8-20
 CHPGV 9-15
 CHPRFS 4-110
 CHPSV 2-21
 CHPSVX 3-49
 CHPTRF 4-66
 CHPTRI 4-70
 CHPTRS 4-73
 CLANGB 11-7
 CLANGE 11-10
 CLANGT 11-12
 CLANHB 11-14
 CLANHE 11-22
 CLANHP 11-17
 CLANHT 11-20
 CLANSB 11-14
 CLANSP 11-17
 CLANSY 11-22
 compiler directives 1-4
 complete orthogonal factorization 5-10
 complex symmetric matrix 2-21, 2-25, 3-49, 3-55,
 4-63, 4-66, 4-70, 4-73, 4-75, 4-78, 4-81,
 4-84
 component stability, backward 3-3
 componentwise condition number 3-3
 computational subprograms 1-9, 4-1, 6-1
 condition number 3-2, 3-6, 3-14, 3-21, 3-26, 3-33,
 3-39, 3-45, 3-49, 3-55, 4-2, 4-8, 4-16,
 4-24, 4-31, 4-39, 4-47, 4-57, 4-63, 4-75,
 4-87, 4-94, 4-103
 condition number, componentwise 3-3
 constraints, linear equality 5-15
 ConvexMLIB Man Pages: LAPACK 1-15
 CPBCON 4-31
 CPBEQU 4-110
 CPBRFS 4-110
 CPBSV 2-10
 CPBSVX 3-26
 CPBTRF 4-34
 CPBTRS 4-37
 CPOCON 4-39
 CPOEQU 4-110
 CPORFS 4-111
 CPOSV 2-13
 CPOSVX 3-33
 CPOTRF 4-41
 CPOTRI 4-43
 CPOTRS 4-45
 CPPCON 4-47
 CPPEQU 4-111
 CPPRFS 4-111
 CPPSV 2-16
 CPPSVX 3-39
 CPPTRF 4-49
 CPPTRI 4-52
 CPPTRS 4-55
 CPTCON 4-57
 CPTRFS 4-111
 CPTSV 2-19
 CPTSVX 3-45
 CPTTRF 4-59
 CPTTRS 4-61
 CSPCON 4-63
 CSPRFS 4-111
 CSPSV 2-21
 CSPSVX 3-49
 CSPTRF 4-66
 CSPTRI 4-70
 CSPTRS 4-73
 CSYCON 4-75
 CSYRFS 4-111
 CSYSV 2-25
 CSYSVX 3-55
 CSYTRF 4-78
 CSYTRI 4-81
 CSYTRS 4-84
 CTBCON 4-87
 CTBRFS 4-111
 CTBTRS 4-91
 CTPCON 4-94
 CTPRFS 4-111
 CTPTRI 4-97
 CTPTRS 4-100
 CTRCON 4-103
 CTRRFS 4-111
 CTRTRI 4-106
 CTRTRS 4-108
 CTZRQF 6-43
 CUNGLQ 6-23
 CUNGQL 6-25
 CUNGQR 6-27
 CUNGRQ 6-29
 CUNMLQ 6-31
 CUNMQL 6-34
 CUNMQR 6-37
 CUNMRQ 6-40
 CXpa 1-4

D

- decomposition, generalized singular value 10-5
 decomposition, singular value 5-7, 10-2
 DGBCON 4-8
 DGBEQU 4-110
 DGBRFS 4-110
 DGBSV 2-3
 DGBSVX 3-6
 DGBTRF 4-11
 DGBTRS 4-14
 DGECON 4-16
 DGEEQU 4-110
 DGEES 7-3
 DGEESX 8-3
 DGEEV 7-7
 DGEEVX 8-9
 DGEES 9-3
 DGEGV 9-7
 DGELQF 6-4

DGELS 5-4
 DGELSS 5-7
 DGELSX 5-10
 DGEQLF 6-6
 DGEQPF 6-8
 DGEQRF 6-11
 DGERFS 4-110
 DGERQF 6-13
 DGESV 2-6
 DGESVD 10-2
 DGESVX 3-14
 DGETRF 4-18
 DGETRI 4-20
 DGETRS 4-22
 DGGGLM 5-13
 DGGLE 5-15
 DGGQRF 6-15
 DGGRQF 6-19
 DGGSVD 10-5
 DGTCON 4-24
 DGTRFS 4-110
 DGTSV 2-8
 DGTSVX 3-21
 DGTTRF 4-27
 DGTTRS 4-29
 DLAMCH 11-6
 DLANGB 11-7
 DLANGE 11-10
 DLANGT 11-12
 DLANSB 11-14
 DLANSP 11-17
 DLANST 11-20
 DLANSY 11-22
 documentation, online 1-15
 documentation, ordering xxii
 DORGLQ 6-23
 DORGQL 6-25
 DORGQR 6-27
 DORGRQ 6-29
 DORMLQ 6-31
 DORMQL 6-34
 DORMQR 6-37
 DORMRQ 6-40
 DPBCON 4-31
 DPBEQU 4-110
 DPBRFS 4-110
 DPBSV 2-10
 DPBSVX 3-26
 DPBTRF 4-34
 DPBTRS 4-37
 DPOCON 4-39
 DPOEQU 4-110
 DPORFS 4-111
 DPOSV 2-13
 DPOSVX 3-33
 DPOTRF 4-41
 DPOTRI 4-43
 DPOTRS 4-45
 DPPCON 4-47
 DPPEQU 4-111
 DPPRFS 4-111
 DPPSV 2-16
 DPPSVX 3-39
 DPPTRF 4-49
 DPPTRI 4-52
 DPPTRS 4-55
 DPTCON 4-57
 DPTRFS 4-111
 DPTSV 2-19
 DPTSVX 3-45
 DPTTRF 4-59
 DPTTRS 4-61
 driver subprograms 1-9, 5-1, 9-1, 10-1
 driver subprograms, expert 3-1, 8-1
 driver subprograms, simple 2-1, 7-1
 DSBEV 7-10
 DSBEVD 7-13
 DSBEVX 8-15
 DSBGV 9-11
 DSPCON 4-63
 DSPEV 7-17
 DSPEVD 7-20
 DSPEVX 8-20
 DSPGV 9-15
 DSPRFS 4-111
 DSPSV 2-21
 DSPSVX 3-49
 DSPTRF 4-66
 DSPTRI 4-70
 DSPTRS 4-73
 DSTEV 7-24
 DSTEVD 7-26
 DSTEVX 8-25
 DSYCON 4-75
 DSYEV 7-29
 DSYEVD 7-32
 DSYEVX 8-28
 DSYGV 9-19
 DSYRFS 4-111
 DSYSV 2-25
 DSYSVX 3-55
 DSYTRF 4-78
 DSYTRI 4-81
 DSYTRS 4-84
 DTBCON 4-87
 DTBRFS 4-111
 DTBTRS 4-91
 DTPCON 4-94
 DTPRFS 4-111
 DTPTRI 4-97
 DTPTRS 4-100
 DTRCON 4-103
 DTRRFS 4-111
 DTRTRI 4-106
 DTRTRS 4-108
 DTZRQF 6-43

E

- eigenproblem, generalized 9-1, 9-7, 9-11, 9-15, 9-19
- eigenproblem, ordinary 7-1, 7-7, 7-10, 7-13, 7-17, 7-20, 7-24, 7-26, 7-29, 7-32, 8-1, 8-9, 8-15, 8-20, 8-25, 8-28
- eigenvalues 7-1, 7-10, 7-13, 7-17, 7-20, 7-24, 7-26, 7-29, 7-32, 8-1, 8-15, 8-20, 8-25, 8-28, 9-1, 9-11, 9-15, 9-19
- eigenvalues, general full matrix 7-3, 7-7, 8-3, 8-9
- eigenvalues, generalized full matrix 9-3, 9-7
- eigenvalues, Hermitian band matrix 7-10, 7-13, 8-15
- eigenvalues, Hermitian band matrix generalized 9-11
- eigenvalues, Hermitian full matrix 7-29, 7-32, 8-28
- eigenvalues, Hermitian full matrix generalized 9-19
- eigenvalues, Hermitian packed matrix 7-17, 7-20, 8-20
- eigenvalues, Hermitian packed matrix generalized 9-15
- eigenvalues, symmetric band matrix 7-10, 7-13, 8-15
- eigenvalues, symmetric band matrix generalized 9-11
- eigenvalues, symmetric full matrix 7-29, 7-32, 8-28
- eigenvalues, symmetric full matrix generalized 9-19
- eigenvalues, symmetric packed matrix 7-17, 7-20, 8-20
- eigenvalues, symmetric packed matrix generalized 9-15
- eigenvalues, symmetric tridiagonal matrix 7-24, 7-26, 8-25
- eigenvectors 7-1, 7-10, 7-13, 7-17, 7-20, 7-24, 7-26, 7-29, 7-32, 8-1, 8-15, 8-20, 8-25, 8-28, 9-1, 9-11, 9-15, 9-19
- eigenvectors, general full matrix 7-7, 8-9
- eigenvectors, generalized full matrix 9-7
- eigenvectors, Hermitian band matrix 7-10, 7-13, 8-15
- eigenvectors, Hermitian band matrix generalized 9-11
- eigenvectors, Hermitian full matrix 7-29, 7-32, 8-28
- eigenvectors, Hermitian full matrix generalized 9-19
- eigenvectors, Hermitian packed matrix 7-17, 7-20, 8-20
- eigenvectors, Hermitian packed matrix generalized 9-15
- eigenvectors, left 7-7, 8-9, 9-7
- eigenvectors, right 7-7, 8-9, 9-7
- eigenvectors, symmetric band matrix 7-10, 7-13, 8-15
- eigenvectors, symmetric band matrix generalized 9-11
- eigenvectors, symmetric full matrix 7-29, 7-32, 8-28
- eigenvectors, symmetric full matrix generalized 9-19
- eigenvectors, symmetric packed matrix 7-17, 7-20, 8-20
- eigenvectors, symmetric packed matrix generalized 9-15
- eigenvectors, symmetric tridiagonal matrix 7-24, 7-26, 8-25
- EISPACK Guide* 1-16
- EISPACK Guide Extension* 1-16
- END_TASKS compiler directive 1-4
- environment 11-4, 11-6
- equality-constrained least squares, solve 5-15
- equations, linear 4-1
- equilibration 3-3, 3-6, 3-14, 3-26, 3-33, 3-39
- error analysis 3-2
- error bound, backward 3-2
- error bound, forward 3-2
- error bounds 3-6, 3-14, 3-21, 3-26, 3-33, 3-39, 3-45, 3-49, 3-55
- error handler 11-25
- error reporting 11-1
- expert driver subprograms 3-1, 8-1
- Extended BLAS 1-3

F

- factor general band matrix 4-11
- factor general full matrix 4-18
- factor general tridiagonal matrix 4-27
- factor Hermitian indefinite full matrix 4-78
- factor Hermitian indefinite packed matrix 4-66
- factor positive definite band matrix 4-34
- factor positive definite full matrix 4-41
- factor positive definite packed matrix 4-49
- factor positive definite tridiagonal matrix 4-59
- factor symmetric indefinite full matrix 4-78
- factor symmetric indefinite packed matrix 4-66
- factorization, Bunch-Kaufman 4-66, 4-78
- factorization, Cholesky 4-34, 4-41, 4-49, 4-59
- factorization, complete orthogonal 5-10
- factorization, generalized *QR* 6-15
- factorization, generalized *RQ* 6-19
- factorization, *LQ* 5-4, 6-23, 6-31
- factorization, *LU* 4-11, 4-18, 4-27
- factorization of general full matrix, *LQ* 6-4
- factorization of general full matrix, *QL* 6-6
- factorization of general full matrix, *QR* 6-8, 6-11
- factorization of general full matrix, *RQ* 6-13
- factorization of upper trapezoidal matrix, *RQ* 6-43
- factorization, orthogonal 6-1
- factorization, *QL* 6-25, 6-34
- factorization, *QR* 5-4, 5-10, 6-27, 6-37
- factorization, *RQ* 6-29, 6-40
- floating-point format 1-5
- FORCE_PARALLEL compiler directive 1-4

forward error bound 3-2
 full matrix eigenvalues, general 7-3, 7-7, 8-3, 8-9
 full matrix eigenvalues, generalized 9-3, 9-7
 full matrix eigenvalues, Hermitian 7-29, 7-32, 8-28
 full matrix eigenvalues, symmetric 7-29, 7-32, 8-28
 full matrix eigenvectors, general 7-7, 8-9
 full matrix eigenvectors, generalized 9-7
 full matrix eigenvectors, Hermitian 7-29, 7-32, 8-28
 full matrix eigenvectors, symmetric 7-29, 7-32, 8-28
 full matrix, factor general 4-18
 full matrix, factor Hermitian indefinite 4-78
 full matrix, factor positive definite 4-41
 full matrix, factor symmetric indefinite 4-78
 full matrix, general 4-16
 full matrix generalized eigenvalues, Hermitian 9-19
 full matrix generalized eigenvalues, symmetric 9-19
 full matrix generalized eigenvectors, Hermitian 9-19
 full matrix generalized eigenvectors, symmetric 9-19
 full matrix, Hermitian 4-75
 full matrix, invert general 4-20
 full matrix, invert Hermitian indefinite 4-81
 full matrix, invert positive definite 4-43
 full matrix, invert symmetric indefinite 4-81
 full matrix, invert triangular 4-106
 full matrix, LQ factorization of general 6-4
 full matrix, norm of general 11-10
 full matrix, norm of Hermitian 11-22
 full matrix, norm of symmetric 11-22
 full matrix, positive definite 4-39
 full matrix, QL factorization of general 6-6
 full matrix, QR factorization of general 6-8, 6-11
 full matrix, RQ factorization of general 6-13
 full matrix, solve general 2-6, 3-14, 4-22
 full matrix, solve Hermitian indefinite 4-84
 full matrix, solve positive definite 2-13, 3-33, 4-45
 full matrix, solve symmetric indefinite 4-84
 full matrix, solve triangular 4-108
 full matrix, symmetric 4-75
 full matrix, triangular 4-103
 further reference xxi

G

general band matrix 4-8
 general band matrix, factor 4-11
 general band matrix, norm of 11-7
 general band matrix, solve 2-3, 3-6, 4-14
 general full matrix 4-16
 general full matrix eigenvalues 7-3, 7-7, 8-3, 8-9
 general full matrix eigenvectors 7-7, 8-9
 general full matrix, factor 4-18

general full matrix, invert 4-20
 general full matrix, LQ factorization of 6-4
 general full matrix, norm of 11-10
 general full matrix, QL factorization of 6-6
 general full matrix, QR factorization of 6-8, 6-11
 general full matrix, RQ factorization of 6-13
 general full matrix, solve 2-6, 3-14, 4-22
 general least squares, solve 5-4, 5-7, 5-10
 general matrices, generalized QR factorization of 6-15
 general matrices, generalized RQ factorization of 6-19
 general diagonal matrix 4-24
 general tridiagonal matrix, factor 4-27
 general tridiagonal matrix, norm of 11-12
 general tridiagonal matrix, solve 2-8, 3-21, 4-29
 generalized eigenproblem 9-1, 9-7, 9-11, 9-15, 9-19
 generalized eigenvalues, Hermitian band matrix 9-11
 generalized eigenvalues, Hermitian full matrix 9-19
 generalized eigenvalues, Hermitian packed matrix 9-15
 generalized eigenvalues, symmetric band matrix 9-11
 generalized eigenvalues, symmetric full matrix 9-19
 generalized eigenvalues, symmetric packed matrix 9-15
 generalized eigenvectors, Hermitian band matrix 9-11
 generalized eigenvectors, Hermitian full matrix 9-19
 generalized eigenvectors, Hermitian packed matrix 9-15
 generalized eigenvectors, symmetric band matrix 9-11
 generalized eigenvectors, symmetric full matrix 9-19
 generalized eigenvectors, symmetric packed matrix 9-15
 generalized full matrix eigenvalues 9-3, 9-7
 generalized full matrix eigenvectors 9-7
 generalized linear regression model, solve 5-13
 generalized QR factorization 6-15
 generalized RQ factorization 6-19
 generalized Schur Form 9-3
 generalized Schur vectors 9-3
 generalized singular value decomposition 10-5
 generate orthogonal matrix 6-23, 6-25, 6-27, 6-29
 GLM problem, solve 5-13
 GQR 6-15
 GRQ 6-19
 GSVD 10-5

H

Hermitian band matrix eigenvalues 7-10, 7-13, 8-15

Hermitian band matrix eigenvectors 7-10, 7-13, 8-15
 Hermitian band matrix generalized eigenvalues 9-11
 Hermitian band matrix generalized eigenvectors 9-11
 Hermitian band matrix, norm of 11-14
 Hermitian full matrix 4-75
 Hermitian full matrix eigenvalues 7-29, 7-32, 8-28
 Hermitian full matrix eigenvectors 7-29, 7-32, 8-28
 Hermitian full matrix generalized eigenvalues 9-19
 Hermitian full matrix generalized eigenvectors 9-19
 Hermitian full matrix, norm of 11-22
 Hermitian indefinite full matrix, factor 4-78
 Hermitian indefinite full matrix, invert 4-81
 Hermitian indefinite full matrix, solve 4-84
 Hermitian indefinite matrix 2-25, 3-55
 Hermitian indefinite matrix, solve 2-25, 3-55
 Hermitian indefinite packed matrix, factor 4-66
 Hermitian indefinite packed matrix, invert 4-70
 Hermitian indefinite packed matrix, solve 2-21, 3-49, 4-73
 Hermitian matrix 4-34, 4-41, 4-49, 4-59
 Hermitian packed matrix 4-63
 Hermitian packed matrix eigenvalues 7-17, 7-20, 8-20
 Hermitian packed matrix eigenvectors 7-17, 7-20, 8-20
 Hermitian packed matrix generalized eigenvalues 9-15
 Hermitian packed matrix generalized eigenvectors 9-15
 Hermitian packed matrix, norm of 11-17
 Hermitian tridiagonal matrix, norm of 11-20

I

ICAMAX 1-3
 IEEE arithmetic format 1-5
 ILAENV 11-4
 improve accuracy 3-6, 3-14, 3-21, 3-26, 3-33, 3-39, 3-45, 3-49, 3-55
 improvement, iterative 3-6, 3-14, 3-21, 3-26, 3-33, 3-39, 3-45, 3-49, 3-55
 indefinite full matrix, factor Hermitian 4-78
 indefinite full matrix, factor symmetric 4-78
 indefinite full matrix, invert Hermitian 4-81
 indefinite full matrix, invert symmetric 4-81
 indefinite full matrix, solve Hermitian 4-84
 indefinite full matrix, solve symmetric 4-84
 indefinite matrix, solve Hermitian 2-25, 3-55
 indefinite matrix, solve symmetric 2-25, 3-55
 indefinite packed matrix, factor Hermitian 4-66
 indefinite packed matrix, factor symmetric 4-66
 indefinite packed matrix, invert Hermitian 4-70
 indefinite packed matrix, invert symmetric 4-70

indefinite packed matrix, solve Hermitian 2-21, 3-49, 4-73
 indefinite packed matrix, solve symmetric 2-21, 3-49, 4-73
 instability, numerical 3-2
 inverse of a matrix 3-4
 inversion, matrix 3-4, 4-2, 4-20, 4-43, 4-52, 4-70, 4-81, 4-97, 4-106
 invert, don't 3-4, 4-2
 invert general full matrix 4-20
 invert Hermitian indefinite full matrix 4-81
 invert Hermitian indefinite packed matrix 4-70
 invert positive definite full matrix 4-43
 invert positive definite packed matrix 4-52
 invert symmetric indefinite full matrix 4-81
 invert symmetric indefinite packed matrix 4-70
 invert triangular full matrix 4-106
 invert triangular packed matrix 4-97
 ISAMAX 1-3
 ISAMIN 1-3
 ISMAX 1-3
 ISMIN 1-3
 iterative refinement 3-3, 3-6, 3-14, 3-21, 3-26, 3-33, 3-39, 3-45, 3-49, 3-55

L

LAPACK 1-2, 1-3
 LAPACK man pages 1-15
 LAPACK8 1-2
 least squares 5-1
 least squares, solve equality-constrained 5-15
 least squares, solve general 5-4, 5-7, 5-10
 left eigenvectors 7-7, 8-9, 9-7
 Level 1 BLAS 1-3
 Level 2 BLAS 1-3, 11-25
 Level 3 BLAS 1-3, 11-25
 linear equality constraints 5-15
 linear equations 2-1, 3-1, 4-1
 linear least squares 5-1
 linear regression model, solve generalized 5-13
LINPACK Users' Guide 1-16
`-llapack` compiler option 1-2
`-llapack8` compiler option 1-2
LQ factorization 5-4, 6-23, 6-31
LQ factorization of general full matrix 6-4
`-lscilib` compiler option 1-3
LU factorization 4-11, 4-18, 4-27
`-lveclib` compiler option 1-2
`-lveclib8` compiler option 1-2

M

machine epsilon 3-2
man pages 1-15
 matrix, general band 4-8
 matrix, general full 4-16
 matrix, general tridiagonal 4-24
 matrix, Hermitian full 4-75
 matrix, Hermitian packed 4-63

- matrix inversion 3-4, 4-2, 4-20, 4-43, 4-52, 4-70, 4-81, 4-97, 4-106
 - matrix inversion, don't 3-4, 4-2
 - matrix multiplication, orthogonal 6-31, 6-34, 6-37, 6-40
 - matrix norm 11-1
 - matrix, positive definite band 4-31
 - matrix, positive definite full 4-39
 - matrix, positive definite packed 4-47
 - matrix, positive definite tridiagonal 4-57
 - matrix, symmetric full 4-75
 - matrix, symmetric packed 4-63
 - matrix, triangular band 4-87
 - matrix, triangular full 4-103
 - matrix, triangular packed 4-94
 - minimum-norm solution 5-1
 - model, solve generalized linear regression 5-13
- N**
- native arithmetic format 1-5
 - NEXT_TASK compiler directive 1-4
 - norm, general band matrix 11-7
 - norm, general full matrix 11-10
 - norm, general tridiagonal matrix 11-12
 - norm, Hermitian band matrix 11-14
 - norm, Hermitian full matrix 11-22
 - norm, Hermitian packed matrix 11-17
 - norm, Hermitian tridiagonal matrix 11-20
 - norm of a matrix 11-1
 - norm, symmetric band matrix 11-14
 - norm, symmetric full matrix 11-22
 - norm, symmetric packed matrix 11-17
 - norm, symmetric tridiagonal matrix 11-20
 - normal equations 5-2
 - numerical instability 3-2
 - numerical singularity 3-2
- O**
- online documentation 1-15
 - optimal block size 11-4, 11-6
 - ordering documentation xxii
 - ordinary eigenproblem 7-1, 7-7, 7-10, 7-13, 7-17, 7-20, 7-24, 7-26, 7-29, 7-32, 8-1, 8-9, 8-15, 8-20, 8-25, 8-28
 - orthogonal factorization 5-4, 6-1
 - orthogonal factorization, complete 5-10
 - orthogonal matrix, generate 6-23, 6-25, 6-27, 6-29
 - orthogonal matrix multiplication 6-31, 6-34, 6-37, 6-40
 - overdetermined 5-1
- P**
- p8 compiler option 1-2, 1-14
 - packed matrix eigenvalues, Hermitian 7-17, 7-20, 8-20
 - packed matrix eigenvalues, symmetric 7-17, 7-20, 8-20
 - packed matrix eigenvectors, Hermitian 7-17, 7-20, 8-20
 - packed matrix eigenvectors, symmetric 7-17, 7-20, 8-20
 - packed matrix, factor Hermitian indefinite 4-66
 - packed matrix, factor positive definite 4-49
 - packed matrix, factor symmetric indefinite 4-66
 - packed matrix generalized eigenvalues, Hermitian 9-15
 - packed matrix generalized eigenvalues, symmetric 9-15
 - packed matrix generalized eigenvectors, Hermitian 9-15
 - packed matrix generalized eigenvectors, symmetric 9-15
 - packed matrix, Hermitian 4-63
 - packed matrix, invert Hermitian indefinite 4-70
 - packed matrix, invert positive definite 4-52
 - packed matrix, invert symmetric indefinite 4-70
 - packed matrix, invert triangular 4-97
 - packed matrix, norm of Hermitian 11-17
 - packed matrix, norm of symmetric 11-17
 - packed matrix, positive definite 4-47
 - packed matrix, solve Hermitian indefinite 2-21, 3-49, 4-73
 - packed matrix, solve positive definite 2-16, 3-39, 4-55
 - packed matrix, solve symmetric indefinite 2-21, 3-49, 4-73
 - packed matrix, solve triangular 4-100
 - packed matrix, symmetric 4-63
 - packed matrix, triangular 4-94
 - parallel processing 1-3
 - parameters 11-4, 11-6
 - pd8 compiler option 1-2, 1-14
 - performance analysis 1-4
 - perturbed problem 3-2
 - positive definite band matrix 4-31
 - positive definite band matrix, factor 4-34
 - positive definite band matrix, solve 2-10, 3-26, 4-37
 - positive definite full matrix 4-39
 - positive definite full matrix, factor 4-41
 - positive definite full matrix, invert 4-43
 - positive definite full matrix, solve 2-13, 3-33, 4-45
 - positive definite packed matrix 4-47
 - positive definite packed matrix, factor 4-49
 - positive definite packed matrix, invert 4-52
 - positive definite packed matrix, solve 2-16, 3-39, 4-55
 - positive definite tridiagonal matrix 4-57
 - positive definite tridiagonal matrix, factor 4-59
 - positive definite tridiagonal matrix, solve 2-19, 3-45, 4-61
 - profiling 1-4
 - programmer's reference 1-15

Q

QL factorization 6-25, 6-34
QL factorization of general full matrix 6-6
QR factorization 5-4, 5-10, 6-27, 6-37
QR factorization, generalized 6-15
QR factorization of general full matrix 6-8, 6-11

R

reentrance 1-4
refinement, iterative 3-6, 3-14, 3-21, 3-26, 3-33,
3-39, 3-45, 3-49, 3-55
regression model, solve generalized linear 5-13
reporting errors 11-1
residual 3-2
right eigenvectors 7-7, 8-9, 9-7
RQ factorization 6-29, 6-40
RQ factorization, generalized 6-19
RQ factorization of general full matrix 6-13
RQ factorization of upper trapezoidal matrix 6-43

S

scaling 3-3
Schur Form 7-3, 8-3
Schur Form, generalized 9-3
Schur vectors 7-3, 8-3
Schur vectors, generalized 9-3
SCILIB 1-3, 11-25
SGBCON 4-8
SGBEQU 4-110
SGBRFS 4-110
SGBSV 2-3
SGBSVX 3-6
SGBTRF 4-11
SGBTRS 4-14
SGECON 4-16
SGEEQU 4-110
SGEES 7-3
SGEESX 8-3
SGEEV 7-7
SGEEVX 8-9
SGEGS 9-3
SGEGV 9-7
SGELQF 6-4
SGELS 5-4
SGELSS 5-7
SGELSX 5-10
SGEMMS 1-3
SGEQLF 6-6
SGEQPF 6-8
SGEQRF 6-11
SGERFS 4-110
SGERQF 6-13
SGESV 2-6
SGESVD 10-2
SGESVX 3-14
SGETRF 4-18
SGETRI 4-20
SGETRS 4-22
SGGGLM 5-13

SGGLSE 5-15
SGGQRF 6-15
SGGRQF 6-19
SGGSVD 10-5
SGTCON 4-24
SGTRFS 4-110
SGTSV 2-8
SGTSVX 3-21
SGTTRF 4-27
SGTTRS 4-29
simple driver subprograms 2-1, 7-1
singular value decomposition 5-7, 10-1, 10-2
singular value decomposition, generalized 10-5
singularity, numerical 3-2
SLAMCH 11-6
SLANB 11-7
SLANGE 11-10
SLANGT 11-12
SLANSB 11-14
SLANSP 11-17
LANST 11-20
SLANSY 11-22
solve equality-constrained least squares 5-15
solve general band matrix 2-3, 3-6, 4-14
solve general full matrix 2-6, 3-14, 4-22
solve general least squares 5-4, 5-7, 5-10
solve general tridiagonal matrix 2-8, 3-21, 4-29
solve generalized linear regression model 5-13
solve Hermitian indefinite full matrix 4-84
solve Hermitian indefinite matrix 2-25, 3-55
solve Hermitian indefinite packed matrix 2-21,
3-49, 4-73
solve positive definite band matrix 2-10, 3-26,
4-37
solve positive definite full matrix 2-13, 3-33, 4-45
solve positive definite packed matrix 2-16, 3-39,
4-55
solve positive definite tridiagonal matrix 2-19,
3-45, 4-61
solve symmetric indefinite full matrix 4-84
solve symmetric indefinite matrix 2-25, 3-55
solve symmetric indefinite packed matrix 2-21,
3-49, 4-73
solve triangular band matrix 4-91
solve triangular full matrix 4-108
solve triangular packed matrix 4-100
SORGLQ 6-23
SORGQL 6-25
SORGQR 6-27
SORGRQ 6-29
SORMLQ 6-31
SORMQL 6-34
SORMQR 6-37
SORMRQ 6-40
SPBCON 4-31
SPBEQU 4-110
SPBRFS 4-110
SPBSV 2-10
SPBSVX 3-26

- SPBTRF 4-34
- SPBTRS 4-37
- SPOCON 4-39
- SPOEQU 4-110
- SPORFS 4-111
- SPOSV 2-13
- SPOSVX 3-33
- SPOTRF 4-41
- SPOTRI 4-43
- SPOTRS 4-45
- SPPCON 4-47
- SPPEQU 4-111
- SPPRFS 4-111
- SPPSV 2-16
- SPPSVX 3-39
- SPPTRF 4-49
- SPPTRI 4-52
- SPPTRS 4-55
- SPTCON 4-57
- SPTRFS 4-111
- SPTSX 2-19
- SPTSXV 3-45
- SPTRF 4-59
- SPTRS 4-61
- SSBEV 7-10
- SSBEVD 7-13
- SSBEVX 8-15
- SSBGV 9-11
- SSPCON 4-63
- SSPEV 7-17
- SSPEVD 7-20
- SSPEVX 8-20
- SSPGV 9-15
- SSPRFS 4-111
- SSPSV 2-21
- SSPSVX 3-49
- SSPTRF 4-66
- SSPTRI 4-70
- SSPTRS 4-73
- SSTEVD 7-24
- SSTEVD 7-26
- SSTEVD 8-25
- SSYCON 4-75
- SSYEV 7-29
- SSYEVX 7-32
- SSYEVX 8-28
- SSYGV 9-19
- SSYRFS 4-111
- SSYSV 2-25
- SSYSVX 3-55
- SSYTRF 4-78
- SSYTRI 4-81
- SSYTRS 4-84
- stability, backward 3-3
- standardization 1-2
- STBCON 4-87
- STBRFS 4-111
- STBTRS 4-91
- STPCON 4-94
- STPRFS 4-111
- STPTRI 4-97
- STPTRS 4-100
- STRCON 4-103
- STRRFS 4-111
- STRTRI 4-106
- STRTRS 4-108
- STZRQF 6-43
- subprograms, computational 4-1, 6-1
- subprograms, driver 5-1, 9-1, 10-1
- subprograms, expert driver 3-1, 8-1
- subprograms, simple driver 2-1, 7-1
- supplemental reading xxi, 1-16, 2-1, 3-4, 4-5, 5-2, 6-3, 7-2, 8-2, 9-2, 10-1, 11-3
- SVD 5-7, 10-1, 10-2, 10-5
- symmetric band matrix eigenvalues 7-10, 7-13, 8-15
- symmetric band matrix eigenvectors 7-10, 7-13, 8-15
- symmetric band matrix generalized eigenvalues 9-11
- symmetric band matrix generalized eigenvectors 9-11
- symmetric band matrix, norm of 11-14
- symmetric full matrix 4-75
- symmetric full matrix eigenvalues 7-29, 7-32, 8-28
- symmetric full matrix eigenvectors 7-29, 7-32, 8-28
- symmetric full matrix generalized eigenvalues 9-19
- symmetric full matrix generalized eigenvectors 9-19
- symmetric full matrix, norm of 11-22
- symmetric indefinite full matrix, factor 4-78
- symmetric indefinite full matrix, invert 4-81
- symmetric indefinite full matrix, solve 4-84
- symmetric indefinite matrix, solve 2-25, 3-55
- symmetric indefinite packed matrix, factor 4-66
- symmetric indefinite packed matrix, invert 4-70
- symmetric indefinite packed matrix, solve 2-21, 3-49, 4-73
- symmetric matrix 4-34, 4-41, 4-49, 4-59
- symmetric matrix, complex 2-21, 2-25, 3-49, 3-55, 4-63, 4-66, 4-70, 4-73, 4-75, 4-78, 4-81, 4-84
- symmetric packed matrix 4-63
- symmetric packed matrix eigenvalues 7-17, 7-20, 8-20
- symmetric packed matrix eigenvectors 7-17, 7-20, 8-20
- symmetric packed matrix generalized eigenvalues 9-15
- symmetric packed matrix generalized eigenvectors 9-15
- symmetric packed matrix, norm of 11-17
- symmetric tridiagonal matrix eigenvalues 7-24, 7-26, 8-25

symmetric tridiagonal matrix eigenvectors 7-24,
7-26, 8-25
symmetric tridiagonal matrix, norm of 11-20

T

TAC, technical assistance center xxii
technical assistance center, TAC xxii
thread, definition 1-3
trapezoidal matrix, *RQ* factorization of upper 6-43
triangular band matrix 4-87
triangular band matrix, solve 4-91
triangular full matrix 4-103
triangular full matrix, invert 4-106
triangular full matrix, solve 4-108
triangular packed matrix 4-94
triangular packed matrix, invert 4-97
triangular packed matrix, solve 4-100
tridiagonal matrix eigenvalues, symmetric 7-24,
7-26, 8-25
tridiagonal matrix eigenvectors, symmetric 7-24,
7-26, 8-25
tridiagonal matrix, factor general 4-27
tridiagonal matrix, factor positive definite 4-59
tridiagonal matrix, general 4-24
tridiagonal matrix, norm of general 11-12
tridiagonal matrix, norm of Hermitian 11-20
tridiagonal matrix, norm of symmetric 11-20
tridiagonal matrix, positive definite 4-57
tridiagonal matrix, solve general 2-8, 3-21, 4-29
tridiagonal matrix, solve positive definite 2-19,
3-45, 4-61

U

undetermined 5-1
upper trapezoidal matrix, *RQ* factorization of 6-43

V

VECLIB 1-2, 1-3, 11-25

X

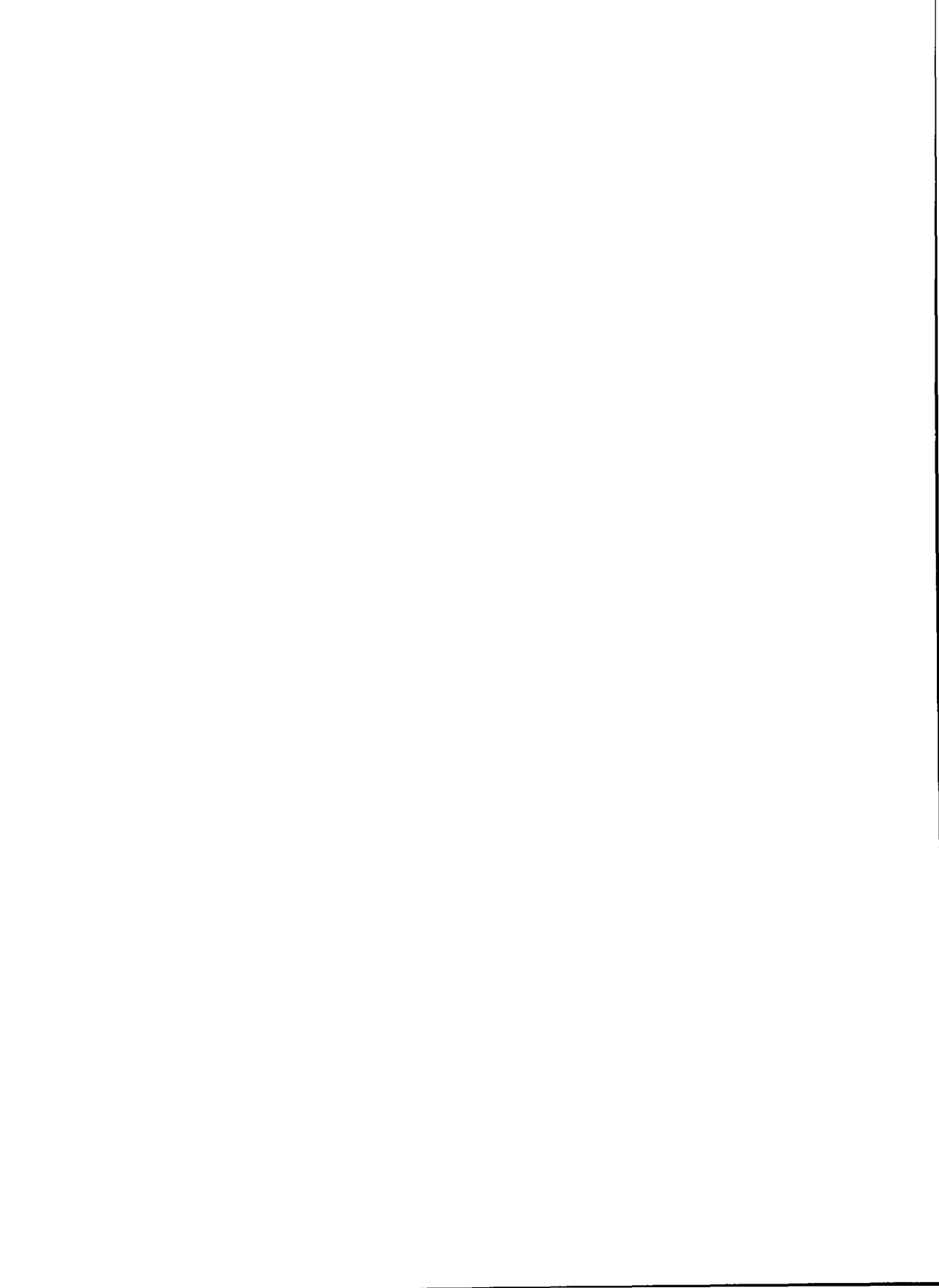
XERBLA 11-25

Z

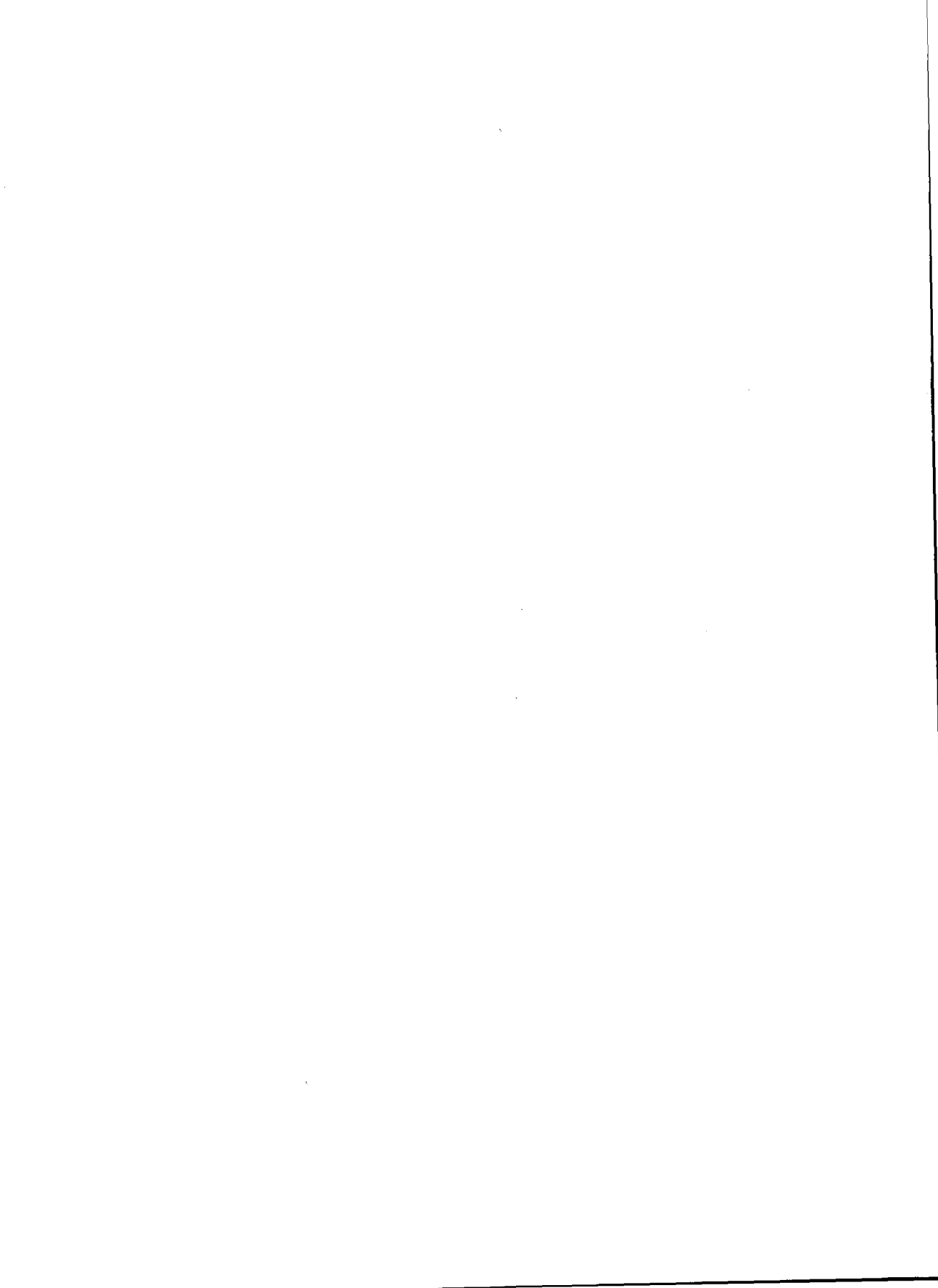
ZGBCON 4-8
ZGBEQU 4-110
ZGBRFS 4-110
ZGBSV 2-3
ZGBSVX 3-6
ZGBTRF 4-11
ZGBTRS 4-14
ZGECON 4-16
ZGEEQU 4-110
ZGEES 7-3
ZGEESX 8-3
ZGEEV 7-7
ZGEEVX 8-9
ZGEGS 9-3
ZGEGV 9-7

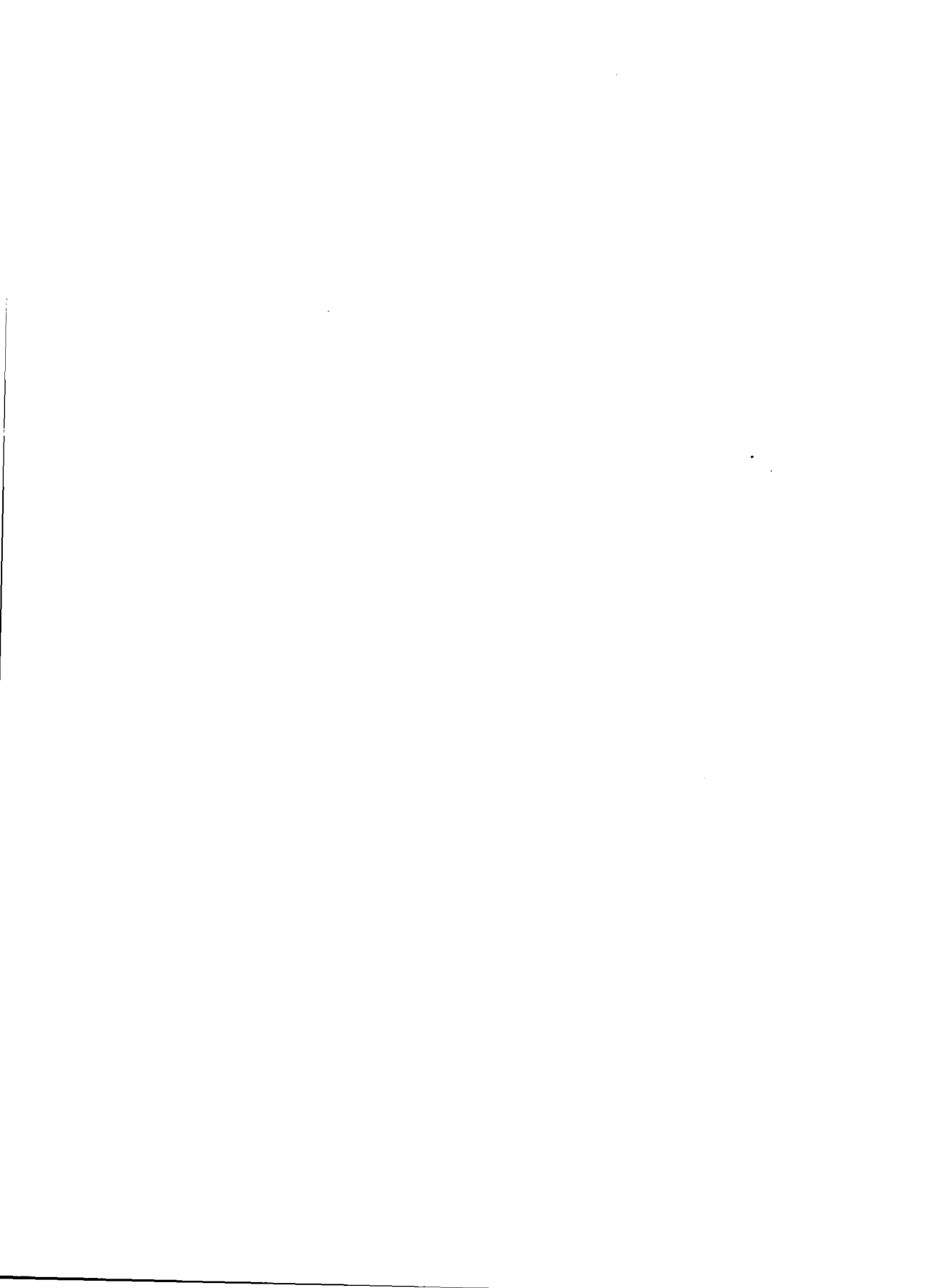
ZGELQF 6-4
ZGELS 5-4
ZGELSS 5-7
ZGELSX 5-10
ZGEQLF 6-6
ZGEQPF 6-8
ZGEQRF 6-11
ZGERFS 4-110
ZGERQF 6-13
ZGESV 2-6
ZGESVD 10-2
ZGESVX 3-14
ZGETRF 4-18
ZGETRI 4-20
ZGETRS 4-22
ZGGGLM 5-13
ZGGLSE 5-15
ZGGQRF 6-15
ZGGRQF 6-19
ZGGSVD 10-5
ZGTCON 4-24
ZGTRFS 4-110
ZGTSV 2-8
ZGTSVX 3-21
ZGTTRF 4-27
ZGTTRS 4-29
ZHBEV 7-10
ZHBEVD 7-13
ZHBEVX 8-15
ZHBGV 9-11
ZHECON 4-75
ZHEEV 7-29
ZHEEVD 7-32
ZHEEVX 8-28
ZHEGV 9-19
ZHERFS 4-110
ZHEV 2-25
ZHESVX 3-55
ZHETRF 4-78
ZHETRI 4-81
ZHETRS 4-84
ZHPCON 4-63
ZHPEV 7-17
ZHPEVD 7-20
ZHPEVX 8-20
ZHPGV 9-15
ZHPRFS 4-110
ZHPSV 2-21
ZHPSVX 3-49
ZHPTRF 4-66
ZHPTRI 4-70
ZHPTRS 4-73
ZLANGB 11-7
ZLANGE 11-10
ZLANGT 11-12
ZLANHB 11-14
ZLANHE 11-22
ZLANHP 11-17
ZLANHT 11-20

ZLANSB	11-14	ZUNGLQ	6-23
ZLANSP	11-17	ZUNGRQ	6-27
ZLANSY	11-22	ZUNGRQ	6-29
ZPBCON	4-31	ZUNMLQ	6-31
ZPBEQU	4-110	ZUNMQL	6-34
ZPBRFS	4-110	ZUNMQR	6-37
ZPBSV	2-10	ZUNMRQ	6-40
ZPBSVX	3-26		
ZPBTRF	4-34		
ZPBTRS	4-37		
ZPOCON	4-39		
ZPOEQU	4-110		
ZPORFS	4-111		
ZPOSV	2-13		
ZPOSVX	3-33		
ZPOTRF	4-41		
ZPOTRI	4-43		
ZPOTRS	4-45		
ZPPCON	4-47		
ZPPEQU	4-111		
ZPPRFS	4-111		
ZPPSV	2-16		
ZPPSVX	3-39		
ZPPTRF	4-49		
ZPPTRI	4-52		
ZPPTRS	4-55		
ZPTCON	4-57		
ZPTRFS	4-111		
ZPTSV	2-19		
ZPTS VX	3-45		
ZPTTRF	4-59		
ZPTTRS	4-61		
ZSPCON	4-63		
ZSPRFS	4-111		
ZSPSV	2-21		
ZSPSVX	3-49		
ZSPTRF	4-66		
ZSPTRI	4-70		
ZSPTRS	4-73		
ZSYCON	4-75		
ZSYRFS	4-111		
ZSYSV	2-25		
ZSYSVX	3-55		
ZSYTRF	4-78		
ZSYTRI	4-81		
ZSYTRS	4-84		
ZTBCON	4-87		
ZTBRFS	4-111		
ZTBTRS	4-91		
ZTPCON	4-94		
ZTPRFS	4-111		
ZTPTRI	4-97		
ZTPTRS	4-100		
ZTRCON	4-103		
ZTRRFS	4-111		
ZTRTRI	4-106		
ZTRTRS	4-108		
ZTZRQF	6-43		
ZUNGLQ	6-23		











ORDER NUMBER
DSW-866

DOCUMENT NUMBER
720-005630-005

