CONVEX LAPACK User's Guide

First Edition

CONVEX COMPUTER CORPORATION

CONVEX LAPACK
User's Guide

Document No. 720-005630-000

First Edition
February 1993

CONVEX Computer Corporation
Richardson, Texas USA

CONVEX LAPACK User's Guide
Order No. DSW-036
First Edition

Printed in the United States of America

Revision information for
CONVEX LAPACK User's Guide

Edition	Document No.	Description
First	720-005630-000	Initial release with CONVEX LAPACK software V8.0, February 1993. First release of manual. Describes LAPACK public domain version 1.0b.

Table of Contents

List of Tables

Permuted Index

Preface

Purpose and Audience

This guide describes the CONVEX LAPACK software library and shows you how to use it. CONVEX 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 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 CONVEX 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 LAPACK development team 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 *CONVEX LAPACK User's Guide* addresses experienced FORTRAN programmers who:

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

Familiarity with the ConvexOS operating system is helpful, but not required, to use this guide. If you are not familiar with ConvexOS, refer to the "Associated Documents" section at the end of this preface.

Organization

To learn fundamental information necessary for using the CONVEX 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 CONVEX 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 CONVEX LAPACK.
- Chapter 6 explains the simple driver subprograms for ordinary eigenanalysis.
- Chapter 7 describes the expert driver subprograms for ordinary eigenanalysis.
- Chapter 8 explains the generalized symmetric eigenvalue subprograms.
- Chapter 9 describes singular value decomposition subprograms.
- Chapter 10 describes user-visible auxiliary subprograms in CONVEX 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.

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 documents are provided by CONVEX Computer Corporation to help you:

- *CONVEX VECLIB User's Guide* (DSW-132). This guide provides information on the subprograms provided with the CONVEX VECLIB library.

- *CONVEX VECLIB Quick Reference* (DSW-134). This compact reference lists the name, purpose and usage for each VECLIB subprogram. Its organization is similar to the *CONVEX VECLIB User's Guide.*

- *CONVEX SCILIB User's Guide* (DSW-360). This guide provides information on the subprograms and functions provided with the CONVEX SCILIB library.

- *CONVEX SCILIB Quick Reference* (DSW-361). This compact reference lists the name, purpose and usage for each SCILIB subprogram. Its organization is similar to the *CONVEX SCILIB User's Guide.*

- *CONVEX FORTRAN Language Reference Manual* (DSW-037). This manual is a reference for the CONVEX FORTRAN programming language and is designed to provide a thorough working definition of the language.

- *CONVEX FORTRAN User's Guide* (DSW-038). This guide tells you how to use the CONVEX FORTRAN compiler, including how to compile, load, and execute programs.

- *CONVEX FORTRAN Optimization Guide* (DSW-034). This guide describes methods for optimizing FORTRAN programs.

- *CONVEX 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.

- *CONVEX Application Compiler User's Guide* (DSW-401). This guide describes the CONVEX Application Compiler and how to use it to optimize programs.

- *CONVEX 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.

- *ConvexOS Man Pages for Users* (DSW-331). This book contains copies of Sections 1 and 7 of the online man pages. These man pages are primarily concerned with operating system information for users.

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

- Anderson, E. *et al. LAPACK Users' Guide.* Philadelphia, PA: Society for Industrial and Applied Mathematics. 1992.

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

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

To order an edition other than the current edition, include the 12-digit document number, as listed on the "Revision information" page.

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.

1

Introduction to CONVEX LAPACK

Overview

CONVEX 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 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 CONVEX LAPACK was derived.

Although CONVEX LAPACK was designed for use with FORTRAN programs, C and Ada programs can call CONVEX LAPACK subprograms, as described in Appendices A and B, respectively, of the *CONVEX VECLIB User's Guide*, which is included in this documentation set.

This chapter provides information necessary for efficient use of CONVEX LAPACK, including discussions of CONVEX LAPACK conformance to public-domain LAPACK standards, the LAPACK and LAPACK8 library files, accessing CONVEX LAPACK subprograms, optimizations, including parallel processing and interactions with other CONVEX analysis and optimization products, supported floating point formats, roundoff effects, the naming convention, CONVEX 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 CONVEX LAPACK works in a parallel computing environment
- know how CONVEX LAPACK interacts with the CONVEX Performance Analyzer and other profilers, the Application Compiler, and CONVEX's two floating-point formats
- know LAPACK naming conventions
- know the overall structure and contents of CONVEX LAPACK
- be able to use FORTRAN type declarations and compiler options
- understand how CONVEX LAPACK handles errors
- know how to access the online *CONVEX LAPACK Programmer's Reference*
- know what to do if you are having trouble using CONVEX LAPACK subprograms

What You Need to Know to Use CONVEX LAPACK

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

Standardization

CONVEX 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 CONVEX computers.

Two CONVEX LAPACK Libraries

Often, it is desirable to run a single precision program in double precision. To support changing the precision without changing the code, the CONVEX FORTRAN Compiler provides several compilation options, namely, **-cfc**, **-p8**, and **-pd8**, that affect the size of FORTRAN data types. For compatibility with these compiler options, CONVEX 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.

For more information about these options, refer to the *CONVEX FORTRAN Language Reference Manual* 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 CONVEX 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 *fc* command line, as follows:

fc [*options*] *file* **-llapack**

or

fc [*options*] *file* **-llapack8**

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

CONVEX VECLIB is documented in the *CONVEX VECLIB User's Guide*. If your program uses subprograms from both CONVEX LAPACK and CONVEX 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.

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

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

Add the linker option **–lveclib8** if CONVEX 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 CONVEX VECLIB, CONVEX SCILIB, and CONVEX 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 VECLIB and LAPACK

A subprogram name conflict exists between VECLIB and LAPACK subprograms SGTSV, DGTSV, CGTSV, and ZGTSV. Both sets of subprograms solve tridiagonal systems of linear equations, but their argument lists and functionality differ. If you use these subprograms, be sure to load the ones you want by specifying their library file name first.

Differences between VECLIB8 and LAPACK8

A subprogram name conflict exists between VECLIB8 and LAPACK8 subprograms SGTSV and CGTSV. Both sets of subprograms solve tridiagonal systems of linear equations, but their argument lists and functionality differ. If you use these subprograms, be sure to load the ones you want by specifying their library file name first.

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

CONVEX 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 CONVEX C2 and C3 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.

To support parallel processing, the CONVEX C2 and C3 Series hardware employs automatic self-allocating processors (ASAP) that effectively manage CPU assignment. When another processor can be used to assist in executing a process, the active thread posts a request for any idle CPU(s) to help. New threads are created and executed by available CPUs. If no CPUs are available, the original thread executes all the work for the process.

CONVEX LAPACK works in both single processor (CONVEX C1 Series) and parallel processor (CONVEX C2 and C3 Series) environments. At run time, a CONVEX LAPACK subprogram determines whether it is being used on a CONVEX C1, C2, or C3 Series processor. If it is running on a CONVEX C2 or C3 Series processor with multiple heads, it detects if the program is already using multiple threads. It uses this information to automatically choose between a single or parallel processor algorithm. Consequently, you can move programs that use CONVEX LAPACK between CONVEX C1, C2, and C3 Series systems freely, without losing compatibility or the advantages of either architecture.

If you are computing on a CONVEX C2 or C3 Series system with multiple processors, you can realize the performance benefits of parallel processing in two ways. First, you can simply call any parallelized CONVEX LAPACK subprogram and let it use parallelism internally. Alternatively, you can call CONVEX LAPACK subprograms in a parallelized loop or region. To obtain parallelism via the second mechanism, you need to be familiar with the concept of reentrancy and with the FORCE_PARALLEL, BEGIN_TASKS, NEXT_TASK, and END_TASKS compiler directives.

CONVEX 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 CONVEX 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:

```
C$DIR FORCE_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 CONVEX 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. For example, the following FORTRAN code makes parallel calls to subprogram SGETRF:

```
C$DIR BEGIN_TASKS
      CALL SGETRF(M1, N1, A1, LDA1, IPIV1, INFO1)
C$DIR NEXT_TASK
      CALL SGETRF(M2, N2, A2, LDA2, IPIV2, INFO2)
C$DIR END_TASKS
```

You usually will not call an LAPACK subprogram in a parallel loop or region unless the problem size is small. Most CONVEX LAPACK subprograms are parallelized internally. If a subprogram is called from a parallelized loop or region, the internal parallelism is disabled because the ASAP mechanism does not support nested parallelism.

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

Profiling CONVEX LAPACK Applications

The CONVEX Performance Analyzer, CXpa, is an interactive tool 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.

CONVEX 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 CONVEX 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 *CONVEX Performance Analyzer (CXpa) User's Guide*, or contact your CONVEX sales representative.

Optimizing with the Application Compiler

The CONVEX Application Compiler is an interprocedural analyzer that tracks the flow of data and control between procedures. The information generated by this analysis removes scope restrictions on optimization, which allows the Application Compiler to generate more efficient code by taking the entire program, with all its dependencies, into account. The database of program information that the interprocedural analyzer builds up also allows the Application Compiler to do better error checking, leading to more robust and reliable programs. CONVEX LAPACK has been annotated to permit the Application Compiler to effectively use CONVEX LAPACK subprograms.

The CONVEX Application Compiler is an optional product. For more information about the Application Compiler, refer to the *CONVEX Application Compiler User's Guide,* or contact your CONVEX sales representative.

Floating Point Formats

C-Series CONVEX 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 *CONVEX Architecture Reference (C Series).*

CONVEX 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 CONVEX LAPACK subprograms into programs whether you compile them to use native or IEEE format. For further information on CONVEX floating point formats, refer to the *CONVEX FORTRAN User's Guide.*

Roundoff Effects

CONVEX 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 CONVEX 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 CONVEX 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 TXXYYY, 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 an 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
GB	General band
GE	General full
GT	General tridiagonal
HB	Complex Hermitian Band
HE	Complex Hermitian
HP	Complex Hermitian, packed storage
HS	Upper Hessenberg
OP	Real orthogonal, packed storage
OR	Real orthogonal
PB	Symmetric or Hermitian positive definite band
PO	Symmetric or Hermitian positive definite
PP	Symmetric or Hermitian positive definite, packed storage
PT	Symmetric or Hermitian positive definite tridiagonal
SB	Real symmetric band
SP	Symmetric, packed storage
ST	Real symmetric tridiagonal
SY	Symmetric
TB	Triangular band
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
EIN	Compute selected eigenvectors by inverse transformations
EQR	Compute eigenvalues and/or Schur form using the *QR* algorithms
EQU	Equilibrate a matrix to reduce its condition number
ERF	Compute eigenvectors using the Pal-Walker-Kahan variant of the *QL* or *QR* 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
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 _GEBRD
GHR	Generate the orthogonal/unitary matrix from _GEHRD
GLQ	Generate the orthogonal/unitary matrix from _GELQF
GQL	Generate the orthogonal/unitary matrix from _GEQLF
GQR	Generate the orthogonal/unitary matrix from _GEQRF
GRQ	Generate the orthogonal/unitary matrix from _GERQF
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 _XXTRD
GV	Driver to compute generalized eigenvalues and/or generalized eigenvectors
HRD	Reduce to upper Hessenberg form by orthogonal transformations
LQF	Compute an *LQ* factorization without pivoting
LS	Driver to solve over- or underdetermined linear system using orthogonal factorizations
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 _GEBRD
MHR	Multiply by the orthogonal/unitary matrix from _GEHRD
MLQ	Multiply by the orthogonal/unitary matrix from _GELQF
MQL	Multiply by the orthogonal/unitary matrix from _GEQLF
MQR	Multiply by the orthogonal/unitary matrix from _GEQRF
MRQ	Multiply by the orthogonal/unitary matrix from _GERQF
MTR	Multiply by the orthogonal/unitary matrix from _XXTRD
QLF	Compute a *QL* factorization without pivoting

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

YYY	Computation
QPF	Compute a *QR* factorization with column pivoting
QRF	Compute a *QR* factorization without pivoting
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
SNA	Estimate reciprocal condition numbers of eigenvalue/vector pairs
SQR	Compute singular values and/or singular vectors using the *QR* algorithm
SV	Simple driver to solve a system of linear equations
SVD	Driver to compute the singular value decomposition and/or singular vectors
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 CONVEX 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 CONVEX 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 XXYYY combination.

Table 1-4: Driver Subprograms

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

Table 1-5: Computational Subprograms for Linear Equations

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

Table 1-8: Computational Subprograms for Singular Value Decomposition

Computation (YYY)	Matrix Form (XX)			
	BD	GE	OR	UN
BRD	-	√	-	-
GBR	-	-	S	C
MBR	-	-	S	C
SQR	√	-	-	-

Most CONVEX 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: CONVEX LAPACK User-Visible Auxiliary Subprograms

Subprogram Name	Operation or Function Performed
ILAENV	Choose Problem-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 CONVEX 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 CONVEX 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, the CONVEX FORTRAN Compiler provides 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 CONVEX 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 *CONVEX 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 the *CONVEX FORTRAN Language Reference Manual.*

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 the 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** $\neq 0$. 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.

CONVEX LAPACK Programmer's Reference

The *CONVEX LAPACK Programmer's Reference* is online documentation that includes information from the *CONVEX LAPACK User's Guide.* This reference contains an introduction to CONVEX LAPACK and to each set of subprograms in CONVEX LAPACK, and reference entries for each subprogram.

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

To access reference entries, use the ConvexOS command

man 3lap *entry_name*

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

man 3lap lapack

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 CONVEX LAPACK has been tested extensively and is very reliable. Therefore, before contacting the TAC about a CONVEX 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 ConvexOS command

man contact

2

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 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:

```
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:

```
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** ≥ 0.

kl The number of subdiagonals within the band of A. **kl** ≥ 0.

ku The number of superdiagonals within the band of A. **ku** ≥ 0.

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

ab The matrix A in band storage, in rows **kl**+1 to 2**kl**+**ku**+1; rows 1 to **kl** of the array need not be set. The 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. **ldab** ≥ **2kl+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. **ldb** ≥ max(1,**n**).

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 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, 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 10), may be replaced with a user-supplied version to change the error procedure. Error conditions are

n < 0,
kl < 0,
ku < 0,
nrhs < 0,
ldab < **2kl+ku+1**, and
ldb < max(1,**n**).

SGESV/DGESV/CGESV/ZGESV — Solve General Linear System

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:

```
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:

```
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** ≥ 0.

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

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

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

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 10), 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**).

SGTSV/DGTSV/.../ZGTSV Solve General Tridiagonal Linear System

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:

```
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:

```
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. **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. **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 **dl**(1), ..., **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 **info** = -k, the k-th argument had an invalid value.

info > 0: If **info** = k, $U(k,k)$ is zero, and the solution has not been computed. The factorization has not been completed unless **info** = **n**.

Notes

These subprograms have different functionality and usage than those with the same names in CONVEX 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 10) 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**).

SPBSV/DPBSV/.../ZPBSV Solve Positive Definite Band Linear System

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 $\mathbf{ab}(kd+1+i-j,j)$. 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 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:

```
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:

```
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** ≥ 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** ≥ 0.

nrhs The number of right hand sides, that is, the number of columns of the matrix B. **nrhs** ≥ 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 the array **ab** as follows:

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

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

ldab The leading dimension of array **ab** in the calling program unit. **ldab** ≥ **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. **ldb** ≥ 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 10), 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 < **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:

```
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:

```
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)
```

SPOSV/DPOSV/CPOSV/ZPOSV Continued

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** ≥ 0.

nrhs The number of right hand sides, that is, the number of columns of the matrix B. **nrhs** ≥ 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 10), 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'.

SPPSV/DPPSV/.../ZPPSV Solve Positive Definite Packed Linear System

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^TAx 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:

```
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 CPPSV (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:

```
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 CPPSV (uplo, n, nrhs, ap, 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** ≥ 0.

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

ap 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:

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**$(i + (j-1) \times (2\mathbf{n}-j)/2) = A(i,j)$ for $j \leq i \leq \mathbf{n}$.

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 factor U or L, from the Cholesky factorization $A = U^*U$ or $A = LL^*$, overwrites the input in the same storage format as A.

SPPSV/DPPSV/CPPSV/ZPPSV Continued

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 10), 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:

```
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)
```

SPTSV/DPTSV/CPTSV/ZPTSV Continued

LAPACK8:

```
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 10), 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

$$\begin{matrix} 11 & 12 & 13 & 14 \\ & 22 & 23 & 24 \\ & & 33 & 34 \\ & & & 44 \end{matrix}$$

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

$$\begin{matrix} 11 & & & \\ 21 & 22 & & \\ 31 & 32 & 33 & \\ 41 & 42 & 43 & 44 \end{matrix}$$

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:

```
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:

```
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** ≥ 0.

nrhs The number of right hand sides, that is, the number of columns of the matrix B. **nrhs** ≥ 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**$(i + (j-1) \times (2\mathbf{n}-j)/2) = A(i,j)$ for $j \leq i \leq \mathbf{n}$.

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 $A = U^T DU$ 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 **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 $D(k\text{-}1{:}k,k\text{-}1{:}k)$ 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.

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 10), 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'.

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:

```
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)
```

SSYSV/DSYSV/CHESV/CSYSV/ZHESV/ZSYSV Continued

```
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:

```
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** ≥ 0.

nrhs The number of right hand sides, that is, the number of columns of the matrix B. **nrhs** ≥ 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.

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** ≥ max(1,**n**).

lwork The length of array **work**. **lwork** ≥ 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.

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**(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 $D(k\text{-}1{:}k,k\text{-}1{:}k)$ 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.

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 10), 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'.

3

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,

$$\|A - L'U'\|_\infty \leq \epsilon\, p_2(n)\, r(n)\, \|A\|_\infty$$

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

$$\|b - Ax'\|_\infty \leq \epsilon\, p_3(n)\, r(n)\, \|A\|_\infty\, \|x'\|_\infty$$

- 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

$$\|\delta A\|_\infty \leq \epsilon\, p_3(n)\, r(n)\, \|A\|_\infty$$

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

$$\|x' - x\|_\infty \leq \epsilon\, p_3(n)\, r(n)\, \kappa(A)\, \|x'\|_\infty$$

In the above, $\|\ \|_\infty$ represents the ∞ vector norm and its subordinate matrix norm, ϵ is the machine epsilon (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.* Philadelphia, PA: Society for Industrial and Applied Mathematics. 1992.

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

SGBSVX/DGBSVX/.../ZGBSVX Solve General Band Linear System

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 **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:

```
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)
```

SGBSVX/DGBSVX/CGBSVX/ZGBSVX Continued

```
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:

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

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

kl The number of subdiagonals within the band of A. **kl** ≥ 0.

ku The number of superdiagonals within the band of A. **ku** ≥ 0.

nrhs The number of right hand sides, that is, the number of columns of the matrix B. **nrhs** ≥ 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) = $A(i,j)$ for max(1,j−**ku**) ≤ i ≤ min(**n**,j+**kl**)

ldab The leading dimension of array **ab** in the calling program unit. **ldab** ≥ **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 2**kl**+**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. **ldafb** ≥ 2**kl**+**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'). $\mathbf{r}(i) > 0$, $i = 1, 2, ..., \mathbf{n}$.

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'). $\mathbf{c}(i) > 0$, $i = 1, 2, ..., \mathbf{n}$.

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** ≥ max(1,**n**).

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

Working Storage

work, iwork, rwork Arrays used for work space.

Output

ab If **fact** = 'E' or 'e' and **equed** = 'R', then A was overwritten by RA.

If **fact** = 'E' or 'e' and **equed** = 'C', then A was overwritten by AC.

If **fact** = 'E' or 'e' and **equed** = 'B', then A was overwritten by RAC.

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', then **afb** returns details of the LU factorization of A, after equilibration, if performed. 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 2**kl**+**ku**+1.

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

ipiv If **fact** = 'N' or 'n', then **ipiv** contains the pivot indices from the factorization $A = PLU$ of the original matrix A.

If **fact** = 'E' or 'e', then **ipiv** contains the pivot indices from the factorization $A = PLU$ of the equilibrated matrix A.

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 = 'R': Row equilibration; A was premultiplied by R.
equed = 'C': Column equilibration; A was postmultiplied by C.
equed = 'B': Both row and column equilibration; A was replaced with RAC.

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

r If **fact** = 'E' or 'e' and **equed** = 'R' or 'B' on exit, the diagonal elements of the diagonal row scaling matrix R.

Destroyed if **fact** = 'E' or 'e' and **equed** = 'N' or 'C' on exit.

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

c If **fact** = 'E' or 'e' and **equed** = 'C' or 'B' on exit, the diagonal elements of the diagonal column scaling matrix C.

Destroyed if **fact** = 'E' or 'e' and **equed** = 'N' or 'R' 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 **trans** = 'N' or 'n' and **equed** = 'R' or 'r' or 'B' or 'b', B is overwritten by RB.

If **trans** = 'T' or 't' or 'C' or 'c' and **equed** = 'C' or 'c' or 'B' or 'b', B was overwritten by CB.

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 + 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 ≤ **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 10), 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,
kl < 0,
ku < 0,
nrhs < 0,
ldab < **kl**+**ku**+1,
ldafb < 2**kl**+**ku**+1,
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 Linear System — SGESVX/DGESVX/CGESVX/ZGESVX

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.

SGESVX/DGESVX/CGESVX/ZGESVX Continued

Usage

LAPACK:

```
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:

```
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 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*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 $A^TX = B$.
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. **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, ..., **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, ..., **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** ≥ max(1,**n**).

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

Working Storage

work, iwork, rwork Arrays used for work space.

Output

a If **fact** = 'E' or 'e' and **equed** = 'R', then A was overwritten by RA.

If **fact** = 'E' or 'e' and **equed** = 'C', then A was overwritten by AC.

If **fact** = 'E' or 'e' and **equed** = 'B', then A was overwritten by RAC.

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 factors L and U from the factorization $A = PLU$ of the matrix A, after equilibration, if performed.

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

ipiv If **fact** = 'N' or 'n', then **ipiv** contains the pivot indices from the factorization $A = PLU$ of the original matrix A.

If **fact** = 'E' or 'e', then **ipiv** contains the pivot indices from the factorization $A = PLU$ of the equilibrated matrix A.

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 = 'R': Row equilibration; A was premultiplied by R.
equed = 'C': Column equilibration; A was postmultiplied by C.
equed = 'B': Both row and column equilibration; A was replaced with RAC.

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

r If **fact** = 'E' or 'e' and **equed** = 'R' or 'B' on exit, the diagonal elements of the diagonal row scaling matrix R.

Destroyed if **fact** = 'E' or 'e' and **equed** = 'N' or 'C' on exit.

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

c If **fact** = 'E' or 'e' and **equed** = 'C' or 'B' on exit, the diagonal elements of the diagonal column scaling matrix C.

Destroyed if **fact** = 'E' or 'e' and **equed** = 'N' or 'R' 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 **trans** = 'N' or 'n' and **equed** = 'R' or 'r' or 'B' or 'b', B is overwritten by RB.

If **trans** = 'T' or 't' or 'C' or 'c' and **equed** = 'C' or 'c' or 'B' or 'b', B was overwritten by CB.

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 + 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** = $k \le$ **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 10), 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-$nrhs$ 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.

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	

SGTSVX/DGTSVX/CGTSVX/ZGTSVX Continued

Usage LAPACK:

```
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:

```
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** ≥ 0.

nrhs The number of right hand sides, that is, the number of columns of the matrix B. **nrhs** ≥ 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'.

SGTSVX/DGTSVX/CGTSVX/ZGTSVX Continued

	df	If **fact** = 'F' or 'f', the **n** diagonal elements of the upper triangular matrix U from the LU factorization of A. 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 U. 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 U. Not used as input if **fact** = 'N' or 'n' or 'E' or 'e'.
	ipiv	If **fact** = 'F' or 'f', the pivot indices from the LU factorization of A 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** ≥ max(1,**n**).
	ldx	The leading dimension of array **x** in the calling program unit. **ldx** ≥ 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 L from the LU factorization of A. Not used as output if **fact** = 'F' or 'f'.
	df	If **fact** = 'N' or 'n', the **n** diagonal elements of the upper triangular matrix U from the LU factorization of A. Not used as output if **fact** = 'F' or 'f'.
	duf	If **fact** = 'N' or 'n', the **n**–1 elements of the first superdiagonal of U. Not used as output if **fact** = 'F' or 'f'.
	du2	If **fact** = 'N' or 'n', the **n**–2 elements of the second superdiagonal of U. Not used as output if **fact** = 'F' or 'f'.
	ipiv	If **fact** = 'N' or 'n', the pivot indices from the LU factorization of A; row i of the matrix was interchanged with row **ipiv**(i). **ipiv**(i) will always be either i or $i+1$; **ipiv**(i) = i indicates a row interchange was not required.
	x	On successful exit, the **n**-by-**nrhs** matrix of solution vectors X 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** $= k \le$ **n**, $U(k,k)$ is zero, or 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 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 10), 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'.

SPBSVX/.../ZPBSVX Solve Positive Definite Band Linear System

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

   $A = U^*U$, if **uplo** = 'U' or 'u', or

   $A = LL^*$, if **uplo** = '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 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 $\mathbf{ab}(kd+1+i-j,j)$. 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 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 $\mathbf{ab}(1+i-j,j)$. 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**.

SPBSVX/DPBSVX/CPBSVX/ZPBSVX Continued

Usage

LAPACK:

```
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:

```
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** ≥ 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** ≥ 0.

nrhs The number of right hand sides, that is, the number of columns of the matrix B. **nrhs** ≥ 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', $\mathbf{ab}(\mathbf{kd}+1+i-j,j) = A(i,j)$ for $\max(1,j-\mathbf{kd}) \le i \le j$.

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

ldab The leading dimension of array **ab** in the calling program unit. **ldab** ≥ **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** ≥ **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'). $\mathbf{s}(i) > 0$, $i = 1, 2, ..., \mathbf{n}$. 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** ≥ max(**1**,**n**).
	ldx	The leading dimension of array **x** in the calling program unit. **ldx** ≥ 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 + 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, 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 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 10), 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'.

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

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

   $A = U^*U$, if **uplo** = 'U' or 'u', or

   $A = LL^*$, if **uplo** = '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:

```
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:

```
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 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'). $\mathbf{s}(i) > 0$, $i = 1, 2, ..., \mathbf{n}$.

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** ≥ max(**1**,**n**).

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

Working Storage

work, iwork, rwork Arrays used for work space.

Output

a 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.

af If **fact** = 'N' or 'n' or 'E' or 'e', then **af** returns 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 + 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, 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 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 10), 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,
nrhs < 0,
lda < max(1,**n**),
ldaf < max(1,**n**),
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 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-$nrhs$ 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. 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 **ap** to array **afp** (after equilibration if **fact** = 'E' or 'e') where 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, 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

$$\begin{matrix} 11 & 12 & 13 & 14 \\ & 22 & 23 & 24 \\ & & 33 & 34 \\ & & & 44 \end{matrix}$$

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

$$\begin{matrix} 11 & & & \\ 21 & 22 & & \\ 31 & 32 & 33 & \\ 41 & 42 & 43 & 44 \end{matrix}$$

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:

```
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:

```
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 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': **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**.
fact = 'N' or 'n': The matrix A is copied to **afp** and factored.
fact = 'E' or 'e': The matrix A is equilibrated, if necessary, then copied to **afp** 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** ≥ 0.
	nrhs	The number of right hand sides, that is, the number of columns of the matrix B. **nrhs** ≥ 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**$(i) > 0$, $i = 1, 2, ...,$ **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 + 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, 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 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 10), 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,
nrhs < 0,
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 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-*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^TAx 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

SPTSVX/DPTSVX/CPTSVX/ZPTSVX Continued

Usage

LAPACK:

```
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:

```
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** ≥ 0.
	nrhs	The number of right hand sides, that is, the number of columns of the matrix B. **nrhs** ≥ 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** ≥ max(1,**n**).
	ldx	The leading dimension of array **x** in the calling program unit. **ldx** ≥ 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, 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 10), may be replaced with a user-supplied version to change the error procedure. Error conditions are

fact ≠ '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

   $A = UDU^T$, if **uplo** = 'U' or 'u', or

   $A = LDL^T$, if **uplo** = '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

$$\begin{matrix} 11 & 12 & 13 & 14 \\ & 22 & 23 & 24 \\ & & 33 & 34 \\ & & & 44 \end{matrix}$$

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:

```
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 ZSPSVX (fact, uplo, n, nrhs, ap, afp, ipiv, b, ldb, x,
             ldx, rcond, ferr, berr, work, rwork, info)
```

LAPACK8:

```
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  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)
```

SSPSVX/DSPSVX/CHPSVX/CSPSVX/ZHPSVX/ZSPSVX Continued

Input	**fact**	Specifies whether or not the factored form of A has been supplied on entry, as follows: **fact** = 'F' or 'f': **afp** and **ipiv** contain the factored form of A. **fact** = 'N' or 'n': The matrix A is copied to **afp** 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** ≥ 0.
	nrhs	The number of right hand sides, that is, the number of columns of the matrix B. **nrhs** ≥ 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**$(i + (j-1) \times (2\mathbf{n}-j)/2) = A(i,j)$ for $j \leq i \leq$ **n**.
	afp	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**).
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 -**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.

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, 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.

SSPSVX/DSPSVX/CHPSVX/CSPSVX/ZHPSVX/ZSPSVX Continued

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 10), may be replaced with a user-supplied version to change the error procedure. Error conditions are

fact ≠ 'F' or 'f' or 'N' or 'n',
uplo ≠ '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

   $A = UDU^*$, if **uplo** = 'U' or 'u', or

   $A = LDL^*$, if **uplo** = '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 * 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:

```
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)
```

SSYSVX/DSYSVX/CHESVX/CSYSVX/ZHESVX/ZSYSVX Continued

```
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 DSYSVX (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 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 ZSYSVX (fact, uplo, n, nrhs, a, lda, af, ldaf, ipiv, b,
             ldb, x, ldx, rcond, ferr, berr, work, lwork,
             rwork, info)
```

LAPACK8:

```
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 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** ≥ 0.

nrhs The number of right hand sides, that is, the number of columns of the matrix B. **nrhs** ≥ 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** ≥ max(1,**n**).

af If **fact** = 'F' or 'f', the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization $A = UDU^*$ 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** ≥ max(1,**n**).

ipiv If **fact** = 'F' or 'f', 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 input if **fact** = 'N' or 'n'.

SSYSVX/DSYSVX/CHESVX/CSYSVX/ZHESVX/ZSYSVX Continued

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** ≥ max(1,**n**).

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

lwork The length of array **work**. **lwork** ≥ 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, iwork, rwork Arrays used for work space.

Output

af If **fact** = 'N' or 'n', the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization $A = UDU^*$ 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 $D(k\text{-}1{:}k,k\text{-}1{:}k)$ 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** ≤ **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 10), may be replaced with a user-supplied version to change the error procedure. Error conditions are

fact ≠ 'F' or 'f' or 'N' or 'n',
uplo ≠ '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 < 2***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'.

4

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 $\| \ \|$ 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)
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.* Philadelphia, PA: Society for Industrial and Applied Mathematics. 1992.

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

Condition Number of General Band Matrix SGBCON/.../ZGBCON

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 **rcond** $= (\|A\| \, \|A^{-1}\|)^{-1}$.

Usage

LAPACK:

```
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:

```
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** ≥ 0.

kl The number of subdiagonals within the band of A. **kl** ≥ 0.

ku The number of superdiagonals within the band of A. **ku** ≥ 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 2**kl**+**ku**+1.

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

ipiv The pivot indices; for 1 ≤ i ≤ **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 .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 10) 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'.

SGBTRF/DGBTRF/.../ZGBTRF — Factor General Band Matrix

Purpose

These subprograms compute an LU factorization of a 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 **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 $\mathbf{ab}(kl + ku + 1 + i - j, j)$. 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**.

Continued — SGBTRF/DGBTRF/CGBTRF/ZGBTRF

Usage

LAPACK:

```
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:

```
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. $\mathbf{m} \geq 0$.

n The number of columns of the matrix A. $\mathbf{n} \geq 0$.

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

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

ab The matrix A in band storage, in rows **kl**+1 to 2**kl**+**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: $\mathbf{ab}(\mathbf{kl}+\mathbf{ku}+1+i-j,j) = A(i,j)$ for $\max(1,j-\mathbf{ku}) \leq i \leq \min(\mathbf{m},j+\mathbf{kl})$

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

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 2**kl**+**ku**+1.

ipiv On successful exit, the pivot indices; for $1 \leq i \leq \min(\mathbf{m},\mathbf{n})$, 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 10) 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**.

Solve General Band System — SGBTRS/DGBTRS/CGBTRS/ZGBTRS

Purpose

These subprograms solve a system of linear equations $AX = B$, $A^TX = B$, 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:

```
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:

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

n The order of the matrix A. **n** ≥ 0.

kl The number of subdiagonals within the band of A. **kl** ≥ 0.

ku The number of superdiagonals within the band of A. **ku** ≥ 0.

SGBTRS/DGBTRS/CGBTRS/ZGBTRS Continued

nrhs The number of right hand sides, that is, the number of columns of the matrix B. **nrhs** ≥ 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 2**kl**+**ku**+1.

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

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 10), 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 $<$ 2**kl**+**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'.

Condition Number of General Matrix — SGECON/.../ZGECON

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 **rcond** $= (\|A\| \, \|A^{-1}\|)^{-1}$.

Usage

LAPACK:

```
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:

```
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 $\| \ \|_1$.
norm = 'I' or 'i': Use $\| \ \|_\infty$.

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. **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, 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 10), 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,
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'.

Factor General Matrix — SGETRF/DGETRF/CGETRF/ZGETRF

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:

```
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:

```
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** ≥ 0.

n The number of columns of the matrix A. **n** ≥ 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).

SGETRF/DGETRF/CGETRF/ZGETRF Continued

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 10), 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**).

Invert General Matrix — SGETRI/DGETRI/CGETRI/ZGETRI

Purpose

These subprograms compute the inverse of a general matrix using the *LU* factorization computed by _GETRF.

Usage

LAPACK:

```
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)

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)

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)

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)
```

LAPACK8:

```
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)

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)
```

Input

n The order of the matrix A. **n** ≥ 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. **lda** ≥ max(1,**n**).

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

lwork The length of array **work**. **lwork** ≥ 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 exit, **work**(1) contains the optimal work space length **lwork** for high performance.

SGETRI/DGETRI/CGETRI/ZGETRI Continued

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 10), 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**).

Solve General Linear System — SGETRS/DGETRS/CGETRS/ZGETRS

Purpose

These subprograms solve a system of linear equations $AX = B$, $A^TX = B$, 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:

```
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:

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

n The order of the matrix A. **n** ≥ 0.

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

SGETRS/DGETRS/CGETRS/ZGETRS Continued

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. **lda** ≥ max(1,**n**).

ipiv The pivot indices from _GETRF; 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** ≥ 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 10), may be replaced with a user-supplied version to change the error procedure. Error conditions are

trans ≠ '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'.

Condition Number of General Tridiagonal Matrix SGTCON/.../ZGTCON

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 **rcond** $= (\|A\|\ \|A^{-1}\|)^{-1}$.

Usage

LAPACK:

```
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:

```
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)
```

SGTCON/DGTCON/CGTCON/ZGTCON Continued

```
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 $\| \quad \|_1$.
norm = 'I' or 'i': Use $\| \quad \|_\infty$.

n The order of the matrix A. **n** ≥ 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 $1 \le i \le$ **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 **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 10) 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, 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'.

SGTTRF/DGTTRF/.../ZGTTRF Factor General Tridiagonal Matrix

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:

```
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:

```
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. $\mathbf{n} \geq 0$.

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 **ipiv**(i). **ipiv**(i) will always be either i or $i+1$; **ipiv**(i) = i indicates a row interchange was not required.

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 10) may be replaced with a user-supplied version to change the error procedure. Error conditions are

$$\mathbf{n} < 0.$$

SGTTRS/DGTTRS/.../ZGTTRS Solve General Tridiagonal System

Purpose These subprograms solve a system of linear equations $AX = B$, $A^TX = B$, 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:

```
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:

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

n The order of the matrix A. **n** ≥ 0.

nrhs The number of right hand sides, that is, the number of columns of the matrix B. **nrhs** ≥ 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** ≥ 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 10), may be replaced with a user-supplied version to change the error procedure. Error conditions are

trans ≠ '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'.

SPBCON/.../ZPBCON Condition Number of Positive Definite Band Matrix

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 **rcond** $= (\|A\|_1 \|A^{-1}\|_1)^{-1}$.

Usage

LAPACK:

```
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:

```
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** ≥ 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** ≥ 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** ≥ 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 10), 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,
ldab < **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'.

Factor Positive Definite Band Matrix SPBTRF/DPBTRF/.../ZPBTRF

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 $x^T Ax$ 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 SPTTRF, 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:

```
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:

```
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** ≥ 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 10), 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, 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'.

SPBTRS/.../ZPBTRS — Solve Positive Definite Band System

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:

```
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:

```
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** ≥ 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** ≥ 0.

nrhs The number of right hand sides, that is, the number of columns of the matrix B. **nrhs** ≥ 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', $\mathbf{ab}(\mathbf{kd}+1+i-j,j) = U(i,j)$ for $\max(1,j-\mathbf{kd}) \le i \le j$;

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

ldab The leading dimension of array **ab** in the calling program unit. **ldab** ≥ **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** ≥ 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 10), 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 < **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'.

SPOCON/.../ZPOCON Condition Number of Positive Definite Matrix

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:**

```
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:

```
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 _POTRF.

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 10), 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'.

SPOTRF/DPOTRF/CPOTRF/ZPOTRF Factor Positive Definite Matrix

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 $x^T Ax$ 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.

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, 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:

```
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:

```
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** ≥ 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** ≥ 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 10), 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'.

SPOTRI/DPOTRI/CPOTRI/ZPOTRI Invert Positive Definite Matrix

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:

```
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:

```
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** ≥ 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** ≥ 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 10), 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'.

SPOTRS/DPOTRS/.../ZPOTRS Solve Positive Definite Linear System

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:

```
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:

```
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** ≥ 0.

nrhs The number of right hand sides, that is, the number of columns of the matrix B. **nrhs** ≥ 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** ≥ 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** ≥ 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 10), 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'.

SPPCON/... Condition Number of Positive Definite Packed Matrix

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 **rcond** $= (\|A\|_1 \|A^{-1}\|_1)^{-1}$.

Usage

LAPACK:

```
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:

```
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 _PPTRF.

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 10), 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'.

SPPTRF/DPPTRF/.../ZPPTRF Factor Positive Definite Packed Matrix

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^TAx 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.

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, 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

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:

```
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:

```
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. $\mathbf{n} \geq 0$.

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', $\mathbf{ap}(i + (j-1) \times j/2) = A(i,j)$ for $1 \leq i \leq j$;

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

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 10), 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'.

Invert Positive Definite Packed Matrix SPPTRI/DPPTRI/.../ZPPTRI

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

$$\begin{matrix} 11 & 12 & 13 & 14 \\ & 22 & 23 & 24 \\ & & 33 & 34 \\ & & & 44 \end{matrix}$$

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
ap(k)	11	12	22	13	23	33	14	24	34	44

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

Lower triangular storage. If the lower triangle of A^{-1} is

$$\begin{matrix} 11 & & & \\ 21 & 22 & & \\ 31 & 32 & 33 & \\ 41 & 42 & 43 & 44 \end{matrix}$$

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
ap(k)	11	21	31	41	22	32	42	33	43	44

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

SPPTRI/DPPTRI/CPPTRI/ZPPTRI — Continued

Usage

LAPACK:

```
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:

```
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** ≥ 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**$(i + (j-1) \times (2\mathbf{n}-j)/2) = A^{-1}(i,j)$ for $j \leq i \leq \mathbf{n}$.

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 10), 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'.

SPPTRS/.../ZPPTRS Solve Positive Definite Packed Linear System

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:

```
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:

```
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** ≥ 0.

nrhs The number of right hand sides, that is, the number of columns of the matrix B. **nrhs** ≥ 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** ≥ 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 10), 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'.

SPTCON/... Condition Number of Positive Definite Tridiagonal Matrix

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 **rcond** $= (\|A\|_1 \|A^{-1}\|_1)^{-1}$.

Usage

LAPACK:

```
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:

```
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** ≥ 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. **anorm** ≥ 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 **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 10), may be replaced with a user-supplied version to change the error procedure. Error conditions are

n < 0, and
anorm < 0.0.

SPTTRF/.../ZPTTRF Factor Positive Definite Tridiagonal Matrix

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^TAx 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:

```
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:

```
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** ≥ 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 10), may be replaced with a user-supplied version to change the error procedure. Error conditions are

$$\mathbf{n} < 0.$$

SPTTRS/.../ZPTTRS Solve Positive Definite Tridiagonal System

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:

```
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:

```
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': $A = U^T DU$ and **e** is the superdiagonal of U.
uplo = 'L' or 'l': $A = LDL^T$ and **e** is the superdiagonal of L.

n The order of the matrix A. **n** ≥ 0.

nrhs The number of right hand sides, that is, the number of columns of the matrix B. **nrhs** ≥ 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.

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** ≥ 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 10), 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'.

SSPCON/... Condition Number of Symmetric or Hermitian Packed Matrix

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 **rcond** $= (\|A\| \ \|A^{-1}\|)^{-1}$.

Usage

LAPACK:

```
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)
```

Continued SSPCON/DSPCON/CHPCON/CSPCON/ZHPCON/ZSPCON

```
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:

```
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** ≥ 0.

ap The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by _SPTRF or _HPTRF.

ipiv Details of the interchanges and the block structure of D as determined by _SPTRF or _HPTRF.

anorm $\|A\|_1$ $(= \|A\|_\infty)$ of the original symmetric or Hermitian matrix A. **anorm** ≥ 0.

Working Storage

work, iwork Arrays used for work space.

SSPCON/DSPCON/CHPCON/CSPCON/ZHPCON/ZSPCON Continued

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 10), 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
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 Symmetric or Hermitian Packed Matrix SSPTRF/.../ZSPTRF

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

$$\begin{matrix} 11 & 12 & 13 & 14 \\ & 22 & 23 & 24 \\ & & 33 & 34 \\ & & & 44 \end{matrix}$$

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

$$\begin{matrix} 11 & & & \\ 21 & 22 & & \\ 31 & 32 & 33 & \\ 41 & 42 & 43 & 44 \end{matrix}$$

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:

```
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)
```

Continued SSPTRF/DSPTRF/CHPTRF/CSPTRF/ZHPTRF/ZSPTRF

LAPACK8:

```
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 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. $\mathbf{n} \geq 0$.

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', $\mathbf{ap}(i + (j-1) \times j/2) = A(i,j)$ for $1 \leq i \leq j$;

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

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 On successful exit, details of the interchanges and the block structure of D:

If $\mathbf{ipiv}(k) > 0$, then rows and columns k and $\mathbf{ipiv}(k)$ were interchanged and $D(k,k)$ is a 1-by-1 diagonal block.

If **uplo** = 'U' or 'u' and $\mathbf{ipiv}(k) = \mathbf{ipiv}(k-1) < 0$, then rows and columns $k-1$ and $-\mathbf{ipiv}(k)$ were interchanged and $D(k-1:k,k-1:k)$ is a 2-by-2 diagonal block.

If **uplo** = 'L' or 'l' and $\mathbf{ipiv}(k) = \mathbf{ipiv}(k+1) < 0$, then rows and columns $k+1$ and $-\mathbf{ipiv}(k)$ 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 **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 10), 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'.

Invert Symmetric or Hermitian Packed Matrix — SSPTRI/.../ZSPTRI

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

$$\begin{matrix} 11 & 12 & 13 & 14 \\ & 22 & 23 & 24 \\ & & 33 & 34 \\ & & & 44 \end{matrix}$$

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
ap(k)	11	12	22	13	23	33	14	24	34	44

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

Lower triangular storage. If the lower triangle of A^{-1} is

$$\begin{matrix} 11 & & & \\ 21 & 22 & & \\ 31 & 32 & 33 & \\ 41 & 42 & 43 & 44 \end{matrix}$$

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
ap(k)	11	21	31	41	22	32	42	33	43	44

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

SSPTRI/DSPTRI/CHPTRI/CSPTRI/ZHPTRI/ZSPTRI Continued

Usage

LAPACK:

```
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:

```
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. $\mathbf{n} \geq 0$.

ap The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by _SPTRF or _HPTRF.

ipiv Details of the interchanges and the block structure of D as determined by _SPTRF or _HPTRF.

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', $\mathbf{ap}(i + (j-1) \times j/2) = A^{-1}(i,j)$ for $1 \leq i \leq j$;

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

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 10), may be replaced with a user-supplied version to change the error procedure. Error conditions are

uplo ≠ 'L' or 'l' or 'U' or 'u',
$\mathbf{n} < 0$.

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 the **uplo** argument as 'Lower' for 'L' or 'Upper' for 'U'.

SSPTRS/.../ZSPTRS Solve Symmetric or Hermitian Packed System

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:

```
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:

```
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** ≥ 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 _SPTRF or _HPTRF.

ipiv Details of the interchanges and the block structure of D as determined by _SPTRF or _HPTRF.

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** ≥ 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 10), 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'.

SSYCON/... Condition Number of Symmetric or Hermitian Matrix

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 **rcond** $= (\|A\| \ \|A^{-1}\|)^{-1}$.

Usage

LAPACK:

```
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)
```

Continued SSYCON/DSYCON/CHECON/CSPCON/ZHECON/ZSPCON

```
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:

```
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 SSYCON (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** ≥ 0.

a The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by _SYTRF or _HETRF.

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

ipiv Details of the interchanges and the block structure of D as determined by _SYTRF or _HETRF.

anorm $\|A\|_1$ $(= \|A\|_1)$ of the original symmetric or Hermitian matrix A. **anorm** ≥ 0.

Working Storage

work, iwork Arrays used for work space.

SSYCON/DSYCON/CHECON/CSPCON/ZHECON/ZSPCON Continued

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 10), 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,
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 Symmetric or Hermitian Matrix SSYTRF/.../ZHETRF/ZSYTRF

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:

```
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)
```

SSYTRF/DSYTRF/CHETRF/CSYTRF/ZHETRF/ZSYTRF Continued

```
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:

```
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** ≥ 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).

Continued SSYTRF/DSYTRF/CHETRF/CSYTRF/ZHETRF/ZSYTRF

Working Storage

work An array used for work space. On 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 **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 $D(k\text{-}1{:}k,k\text{-}1{:}k)$ 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.

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 10), 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,
lda < 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 the **uplo** argument as 'Lower' for 'L' or 'Upper' for 'U'.

SSYTRI/.../ZHETRI/ZSYTRI Invert Symmetric or Hermitian Matrix

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:

```
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:

```
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 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** ≥ 0.

a The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by _SYTRF or _HETRF.

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 _SYTRF or _HETRF.

Working Storage

work An array used for work space.

Output

a 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', 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.

If **uplo** = 'L' or 'u', 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.

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 10), 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 Symmetric or Hermitian System SSYTRS/.../ZHETRS/ZSYTRS

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:

```
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:

```
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)
```

SSYTRS/DSYTRS/CHETRS/CSYTRS/ZHETRS/ZSYTRS Continued

```
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** ≥ 0.

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

a The block diagonal matrix D and the multipliers used to obtain the factor U or L as computed by _SYTRF or _HETRF.

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

ipiv Details of the interchanges and the block structure of D as determined by _SYTRF or _HETRF.

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** ≥ 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.

Continued SSYTRS/DSYTRS/CHETRS/CSYTRS/ZHETRS/ZSYTRS

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 10), 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'.

STBCON/.../ZTBCON Condition Number of Triangular Band Matrix

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 **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 **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 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 **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:

```
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:

```
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 $\| \ \|_1$.
norm = 'I' or 'i': Use $\| \ \|_\infty$.

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** ≥ 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** ≥ 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 10), 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',
uplo ≠ 'L' or 'l' or 'U' or 'u',
diag ≠ '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'.

STBTRS/DTBTRS/.../ZTBTRS Solve Triangular Band Linear System

Purpose

These subprograms solve a system of linear equations $AX = B$, $A^TX = B$, 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:

```
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:

```
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 $A^TX = B$.
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 10), 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'.

Condition Number of Triangular Packed Matrix — STPCON/...

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 **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 **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:

```
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:

```
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 $\| \quad \|_1$.
norm = 'I' or 'i': Use $\| \quad \|_\infty$.

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** ≥ 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.

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 10), 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',
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 **norm** argument as 'Infinity-Norm' for 'I' or '1-norm' for '1'.

STPTRI/DTPTRI/CTPTRI/ZTPTRI Invert Triangular Packed Matrix

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

$$\begin{matrix} 11 & 12 & 13 & 14 \\ 0 & 22 & 23 & 24 \\ 0 & 0 & 33 & 34 \\ 0 & 0 & 0 & 44 \end{matrix}$$

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

$$\begin{matrix} 11 & 0 & 0 & 0 \\ 21 & 22 & 0 & 0 \\ 31 & 32 & 33 & 0 \\ 41 & 42 & 43 & 44 \end{matrix}$$

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:

```
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:

```
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. $\mathbf{n} \geq 0$.

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', $\mathbf{ap}(i + (j-1) \times (2\mathbf{n}-j)/2) = A(i,j)$ for $j \leq i \leq \mathbf{n}$.

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.

STPTRI/DTPTRI/CTPTRI/ZTPTRI Continued

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 10), 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',
diag $\neq$ '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'.

Solve Triangular Packed Linear System STPTRS/DTPTRS/.../ZTPTRS

Purpose

These subprograms solve a system of linear equations $AX = B$, $A^TX = B$, 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:

```
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:

```
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** ≥ 0.

nrhs The number of right hand sides, that is, the number of columns of the matrix B. **nrhs** ≥ 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**$(i + (j-1) \times (2\mathbf{n}-j)/2) = A(i,j)$ for $j \leq i \leq \mathbf{n}$.

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 10), 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'.

STRCON/.../ZTRCON Condition Number of Triangular Matrix

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 **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:

```
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:

```
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 $\| \ \|_1$.
norm = 'I' or 'i': Use $\| \ \|_\infty$.

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** ≥ 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** ≥ max(1,**n**).

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 10), 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',
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 **norm** argument as 'Infinity-Norm' for 'I' or '1-norm' for '1'.

Invert Triangular Matrix — STRTRI/DTRTRI/CTRTRI/ZTRTRI

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:

```
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:

```
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** ≥ 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** ≥ 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 10), 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'.

Solve Triangular Linear System — STRTRS/DTRTRS/.../ZTRTRS

Purpose

These subprograms solve a system of linear equations $AX = B$, $A^TX = B$, 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:

```
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:

```
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 $A^TX = B$.
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** ≥ 0.

nrhs The number of right hand sides, that is, the number of columns of the matrix B. **nrhs** ≥ 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** ≥ 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** ≥ 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 10), may be replaced with a user-supplied version to change the error procedure. Error conditions are

uplo ≠ 'L' or 'l' or 'U' or 'u',
trans ≠ 'N' or 'n' or 'T' or 't' or 'C' or 'c',
diag ≠ '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 the *CONVEX LAPACK User's Guide*

Although CONVEX LAPACK includes all computational subprograms for linear equations, as defined in Table 1-5, the following subprograms are not documented in the *CONVEX LAPACK User's Guide.* Refer to (Anderson, *et al.*) for documentation for these subprograms.

Table 4-1: Subprograms not in the *CONVEX LAPACK User's 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
SGEEQU	Equilibrate a General Full Matrix
DGEEQU	Equilibrate a General Full Matrix
CGEEQU	Equilibrate a General Full Matrix
ZGEEQU	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

Continued **Subprograms not in the *CONVEX LAPACK User's Guide***

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

5

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

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 $A^TAx = A^Tb$. The matrix A^TA is of size only n-by-n, which is attractive if $m \gg n$. If A has full column rank, then A^TA, 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, for example, 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

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, 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 $A^TX = B$ 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 $X = Q_1R^{-*}B$, 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^TX-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 $X = L^{-*}Q_1B$, where L^{-*} is the inverse of the conjugate transpose of L.

SGELS/DGELS/CGELS/ZGELS — Continued

Usage

LAPACK:

```
CHARACTER*1 trans
INTEGER*4   info, lda, ldb, lwork, m, n, nrhs
REAL*4      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:

```
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 $A^TX = B$.
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** ≥ 0.

n The number of columns of the matrix A. **n** ≥ 0.

nrhs The number of right hand sides; that is, the number of columns of the matrix B. **nrhs** ≥ 0.

a The **m**-by-**n** matrix A.

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

lwork The length of array **work**. **lwork** ≥ 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 exit, **work**(1) contains the optimal work space length **lwork** for high performance.

Output

a If **m** ≥ **n**, **a** has been overwritten by the QR factorization of A.

If **m** < **n**, **a** has been overwritten by the LQ factorization of A.

b On successful exit, if **m** ≥ **n** and **trans** = 'N' or 'n', the **n**-by-**nrhs** least squares solution.

On successful exit, if **m** ≥ **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', the **m**-by-**nrhs** least squares solution.

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 10), may be replaced with a user-supplied version to change the error procedure. Error conditions are

trans ≠ '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'.

SGELSS/DGELSS/CGELSS/ZGELSS Using Singular Value Decomposition

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:

```
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)-4)), 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)-4)), 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:

```
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)-4)), 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** ≥ 0.

n The number of columns of the matrix A. **n** ≥ 0.

nrhs The number of right hand sides; that is, the number of columns of the matrix B. **nrhs** ≥ 0.

a The matrix A specifying the least squares problem.

lda The leading dimension of array **a** in the calling program unit. **lda** ≥ 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** ≥ 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

$$x_i = \begin{cases} b_i/s_{ii} & \text{if } s_{ii} > \textbf{rcond} \times \max_j(s_{jj}) \\ 0 & \text{if } s_{ii} \leq \textbf{rcond} \times \max_j(s_{jj}) \end{cases}$$

lwork The length of array **work**.

lwork ≥ 3**n** + max(1,**m**,2**n**–4,**nrhs**) if **m** ≥ **n**.

lwork ≥ 3**m** + max(1,2**m**,**n**–4,**nrhs**) if **m** < **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, rwork Arrays used for work space. On exit, **work**(1) contains the optimal work space length **lwork** for high performance.

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.

SGELSS/DGELSS/CGELSS/ZGELSS Continued

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 10), 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.

Using Complete Orthogonal Factorization SGELSX/DGELSX/.../ZGELSX

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:

```
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)
```

SGELSX/DGELSX/CGELSX/ZGELSX Continued

LAPACK8:

```
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** ≥ 0.

n The number of columns of the matrix A. **n** ≥ 0.

nrhs The number of right hand sides; that is, the number of columns of the matrix B. **nrhs** ≥ 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 An array of length **lwork** = max(min(**m**,**n**)+3**n**, 2min(**m**,**n**)+**nrhs**), used for work space.

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 10), 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**).

6

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 7 describes the LAPACK expert drivers for ordinary eigenvalue problems. Refer to Chapter 8 for software to compute the eigenvalues or eigenvalues and eigenvectors of a real symmetric or complex Hermitian generalized eigenproblem.

Refer to Chapter 7 of the *CONVEX VECLIB User's Guide* 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 $Q^TAQ = R$. 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

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, $i = 1, 2, \ldots, n$, for which there exist corresponding nonzero vectors, z_i, $i = 1, 2, \ldots, n$, 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.

Usage

LAPACK:

```
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)
```

SGEES/DGEES/CGEES/ZGEES Continued

```
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:

```
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 A. **n** ≥ 0.

a The **n**-by-**n** matrix A.

lda The leading dimension of array **a** in the calling program unit. **lda** ≥ 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** ≥ **n**.

lwork The length of array **work**. **lwork** ≥ max(1,3**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 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 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**–**sdim**), and A_{22} is (**n**–**sdim**)-by-(**n**–**sdim**), and where the eigenvalues λ_j, j = 1, 2, ..., **sdim**, of A_{11} satisfy **select**(λ_j) = .TRUE. and the eigenvalues λ_j, j = **sdim**+1, **sdim**+2, ..., **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.

SGEES/DGEES/CGEES/ZGEES Continued

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, $j = 1, 2, ..., \mathbf{n}$. 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, $j = 1, 2, ..., \mathbf{n}$. 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 10), 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'
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, $i = 1, 2, \ldots, n$, for which there exist corresponding nonzero vectors, z_i, $i = 1, 2, \ldots, n$, 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:

```
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:

```
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)
```

SGEEV/DGEEV/CGEEV/ZGEEV — Continued

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** ≥ 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** ≥ max(1,**n**).

ldvl The leading dimension of array **vl** in the calling program unit. **ldvl** ≥ 1, and if **jobvl** = 'V' or 'v', then **ldvl** ≥ **n**.

ldvr The leading dimension of array **vr** in the calling program unit. **ldvr** ≥ 1, and if **jobvr** = 'V' or 'v', then **ldvr** ≥ **n**.

lwork The length of array **work**. **lwork** ≥ max(1,3**n**) if **jobvl** = 'N' or 'n' and **jobvr** = 'N' or 'n', and **lwork** ≥ max(1,4**n**) 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 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, ..., **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, ..., **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, ..., **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.

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, $j = 1, 2, \ldots, \mathbf{n}$, 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.

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 10), 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**),
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'.

SSBEV/DSBEV/CHBEV/ZHBEV Symmetric or Hermitian Band Matrix

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, $i = 1, 2, \ldots, n$, for which there exist corresponding nonzero vectors, z_i, $i = 1, 2, \ldots, n$, 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 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 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 $\mathbf{ab}(kd+1+i-j, j)$. 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 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:

```
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:

```
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. $\mathbf{n} \geq 0$.

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'. $\mathbf{kd} \geq 0$.

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', $\mathbf{ab}(\mathbf{kd}+1+i-j,j) = A(i,j)$ for $\max(1,j-\mathbf{kd}) \leq i \leq j$;

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

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

ldz The leading dimension of array **z** in the calling program unit. $\mathbf{ldz} \geq 1$, and if **jobz** = 'V' or 'v', then $\mathbf{ldz} \geq \mathbf{n}$.

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 10), 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 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, $i = 1, 2, \ldots, n$, for which there exist corresponding nonzero vectors, z_i, $i = 1, 2, \ldots, n$, 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

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**$(i + (j-1) \times (2n-j)/2)$.

SSPEV/DSPEV/CHPEV/ZHPEV Continued

Usage

LAPACK:

```
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:

```
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. $\mathbf{n} \geq 0$.

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', $\mathbf{ap}(i + (j-1) \times j/2) = A(i,j)$ for $1 \leq i \leq j$;

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

	ldz	The leading dimension of array **z** in the calling program unit. **ldz** ≥ 1. If eigenvectors are desired, then **ldz** ≥ max(1,**n**).
Working Storage	**work, rwork**	Arrays used for work space.
Output	**a**	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 10), 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'.

SSTEV/DSTEV — Real Symmetric Tridiagonal Matrix

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, $i = 1, 2, \ldots, n$, for which there exist corresponding nonzero vectors, z_i, $i = 1, 2, \ldots, n$, 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	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:

```
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 SSTEV (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 DSTEV (jobz, n, d, e, z, ldz, work, info)
```

LAPACK8:

```
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 SSTEV (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 10), 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'.

SSYEV/DSYEV/CHEEV/ZHEEV Symmetric or Hermitian Matrix

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, $i = 1, 2, \ldots, n$, for which there exist corresponding nonzero vectors, z_i, $i = 1, 2, \ldots, n$, 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:

```
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:

```
CHARACTER*1 jobz, uplo
INTEGER*8   info, lda, lwork, n
REAL*8      a(lda, n), w(n), work(lwork)
CALL SSYEV (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 CHEEV (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 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. $\mathbf{n} \geq 0$.

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** ≥ max(1,**n**).

lwork The length of array **work**. **lwork** ≥ max(1,3**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, rwork Arrays used for work space. On 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.

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. 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.

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.

SSYEV/DSYEV/CHEEV/ZHEEV Continued

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 10), 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(1,3**n**–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'.

7

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 6 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 8 for software to compute the eigenvalues or eigenvalues and eigenvectors of a real symmetric or complex Hermitian generalized eigenproblem.

Refer to Chapter 7 of the *CONVEX VECLIB User's Guide* 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 6.

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.* Philadelphia, PA: Society for Industrial and Applied Mathematics. 1992.

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

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, $i = 1, 2, \ldots, n$, for which there exist corresponding nonzero vectors, z_i, $i = 1, 2, \ldots, n$, 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:

```
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)
```

SGEESX/DGEESX/CGEESX/ZGEESX Continued

```
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 ZGEESX (jobvs, sort, select, sense, n, a, lda, sdim, w,
             vs, ldvs, rconde, rcondv, work, lwork, rwork,
             bwork, info)
```

LAPACK8:

```
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** ≥ 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** ≥ **n**.

lwork The length of array **work**. **lwork** ≥ max(1,3**n**). If **sense** = 'E' or 'e' or 'V' or 'v' or 'B' or 'b', then **lwork** ≥ **n** + 2**sdim**×(**n**–**sdim**) where **sdim** is the total number of eigenvalues selected. Note that **n** + 2**sdim**×(**n**–**sdim**) ≤ **n**×(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 **lwork** ≥ **sdim**×(**n**–**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 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**–**sdim**), and A_{22} is (**n**–**sdim**)-by-(**n**–**sdim**), and where the eigenvalues λ_j, j = 1, 2, ..., **sdim**, of A_{11} satisfy **select**(λ_j) = .TRUE. and the eigenvalues λ_j, j = **sdim**+1, **sdim**+2, ..., **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, ..., **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, $j = 1, 2, \ldots, \mathbf{n}$. 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}:

$$(\lambda_1 + \lambda_2 + \cdots + \lambda_{\mathbf{sdim}})/\mathbf{sdim}$$

$0 \leq$ **rconde** ≤ 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 $\| \ \|_F$ is the Frobenius norm. **rconde** underestimates the true reciprocal condition number, but not by more than a factor of $\sqrt{\mathbf{n}}$.

An approximate bound on the error between the computed average of the eigenvalues of A_{11} is $\epsilon \|A\|/$**rconde**, where ϵ is the machine epsilon.

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}:

$$\mathrm{sep}(A_{11}, A_{22}) = \sigma_{\min}(K)$$

where $\sigma_{\min}$ is the smallest singular value of the **sdim**$\times$(**n**–**sdim**)-by-**sdim**$\times$(**n**–**sdim**) matrix

$$K = I_{\mathbf{n-sdim}} \otimes A_{11} - A_{22}^T \otimes I_{\mathbf{sdim}}$$

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 $\|K^{-1}\|_1$. **rconde** cannot differ from the true reciprocal condition number by more than a factor of $(\mathbf{sdim} \times (\mathbf{n}\text{–}\mathbf{sdim}))^{1/2}$. 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 $\epsilon \|A\|/$**rcondv**, where ϵ is the machine epsilon.

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 10), 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, $i = 1, 2, \ldots, n$, for which there exist corresponding nonzero vectors, z_i, $i = 1, 2, \ldots, n$, 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:

```
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)
```

SGEEVX/DGEEVX/CGEEVX/ZGEEVX Continued

```
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:

```
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, wr, wi,
             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** ≥ 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** ≥ max(1,**n**).

ldvl The leading dimension of array **vl** in the calling program unit. **ldvl** ≥ 1, and if **jobvl** = 'V' or 'v', then **ldvl** ≥ **n**.

ldvr The leading dimension of array **vr** in the calling program unit. **ldvr** ≥ 1, and if **jobvr** = 'V' or 'v', then **ldvr** ≥ **n**.

lwork The length of array **work**. For SGEEVX and DGEEVX, if **sense** = 'N' or 'n' or 'E' or 'e', **lwork** ≥ max(1,2**n**), and if **jobvl** = 'V' or 'v' or **jobvr** = 'V' or 'v', **lwork** ≥ max(1,3**n**). If **sense** = 'V' or 'v' or 'B' or 'b', **lwork** ≥ **n**×(**n**+6).

For CGEEVX and ZGEEVX, if **sense** = 'N' or 'n' or 'E' or 'e', **lwork** ≥ max(1,2**n**). If **sense** = 'V' or 'v' or 'B' or 'b', **lwork** ≥ **n**×(**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 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, ..., **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, ..., **n**. The eigenvalues are not in any particular order.

vl On successful exit, if **jobvl** = 'V' or 'v', the left eigenvectors, y_j, $j = 1, 2, \ldots, \mathbf{n}$, 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.

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, $j = 1, 2, \ldots, \mathbf{n}$, 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.

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:

$$P = (\mathbf{ilo}\text{-}1, P_{\mathbf{ilo}-1}) \cdots (2, P_2)\,(1, P_1)\,(\mathbf{ihi}+1, P_{\mathbf{ihi}+1}) \cdots (\mathbf{n}\text{-}1, P_{\mathbf{n}-1})\,(\mathbf{n}, P_{\mathbf{n}})$$

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:

$$\mathbf{scale}(j) = \begin{cases} P_j & \text{for } j = 1, 2, \ldots, \mathbf{ilo}-1 \\ D_{jj} & \text{for } j = \mathbf{ilo}, \mathbf{ilo}+1, \ldots, \mathbf{ihi} \\ P_j & \text{for } j = \mathbf{ihi}+1, \mathbf{ihi}+2, \ldots, \mathbf{n}. \end{cases}$$

abnrm 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

$$\mathbf{rconde}(j) = |y_j^* z_j|$$

where y_j and z_j are left and right unit eigenvectors, respectively, corresponding to λ_j. $0 \leq \mathbf{rconde}(j) \leq 1$, with $\mathbf{rconde}(j) \approx 0$ indicating that λ_j is very poorly conditioned, and $\mathbf{rconde}(j) \approx 1$ 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

$$|\tilde{\lambda}_j - \lambda_j| \lesssim \epsilon \|A\|_1 / \mathbf{rconde}(j)$$

where ϵ is the machine epsilon.

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

$$\mathbf{rcondv}(j) = \sigma_{\min}(T_{22} - \lambda_j I)$$

where $\sigma_{\min}$ 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

$$\theta(\tilde{z}_j, z_j) \lesssim \epsilon \, \mathbf{abnrm} / \mathbf{rcondv}(j)$$

where $\theta(x,y)$ is the angle between vectors x and y, and ϵ is the machine epsilon.

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, ..., **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 10), may be replaced with a user-supplied version to change the error procedure. Error conditions are

balanc ≠ 'N' or 'n' or 'P' or 'p' or 'S' or 's' or 'B' or 'b',
jobvl ≠ 'N' or 'n' or 'V' or 'v',
jobvr ≠ 'N' or 'n' or 'V' or 'v',
sense ≠ '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** ≠ 'V' or 'v',
sense = 'E' or 'e' or 'B' or 'b' and **jobvr** ≠ '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, $i = 1, 2, \ldots, n$, for which there exist corresponding nonzero vectors, z_i, $i = 1, 2, \ldots, n$, 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 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 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 **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**.

SSBEVX/DSBEVX/CHBEVX/ZHBEVX Continued

Usage

LAPACK:

```
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:

```
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)
```

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 $\mathbf{vl} < \lambda \leq \mathbf{vu}$ are to be computed.
range = 'I' or 'i': Eigenvalues λ_{il} 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. $\mathbf{n} \geq 0$.

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'. $\mathbf{kd} \geq 0$.

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', $\mathbf{ab}(\mathbf{kd}+1+i-j,j) = A(i,j)$ for $\max(1,j-\mathbf{kd}) \leq i \leq j$;

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

ldab The leading dimension of array **ab** in the calling program unit. $\mathbf{ldab} \geq \max(1,\mathbf{n})$.

ldq The leading dimension of array **q** in the calling program unit. If **jobz** = 'V' or 'v', then $\mathbf{ldq} \geq \max(1,\mathbf{n})$.

vl If **range** = 'V' or 'v', the left endpoint of the interval of selected eigenvalues. Only eigenvalues $\lambda > \mathbf{vl}$ are returned. $\mathbf{vl} < \mathbf{vu}$.

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

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 $b-a \leq$ **abstol** $+ \epsilon \max(|a|,|b|)$, where ϵ is the machine epsilon. If **abstol** ≤ 0, then $\epsilon \|T\|_1$ is used in its place, where T is the matrix obtained by reducing A to tridiagonal form. For most problems, this is the appropriate level of accuracy to request. For certain strongly graded matrices, greater accuracy can be obtained in very small eigenvalues by setting **abstol** to some very small positive number.

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

ab Destroyed.

q If **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. $0 \leq$ **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 10), 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',
range $\neq$ 'A' or 'a' or 'V' or 'v' or 'I' or 'i',
uplo $\neq$ 'L' or 'l' or 'U' or 'u',
n < 0,
ldab $<$ **kd**+1,
range $=$ 'V' or 'v' and **vu** $\leq$ **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'.

SSPEVX/DSPEVX/.../ZHPEVX Symmetric or Hermitian Packed Matrix

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, $i = 1, 2, \ldots, n$, for which there exist corresponding nonzero vectors, z_i, $i = 1, 2, \ldots, n$, 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

$$\begin{matrix} 11 & 12 & 13 & 14 \\ & 22 & 23 & 24 \\ & & 33 & 34 \\ & & & 44 \end{matrix}$$

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 $\mathbf{ap}(i + j \times (j-1)/2)$.

Lower triangular storage. If the lower triangle of A is

$$\begin{matrix} 11 & & & \\ 21 & 22 & & \\ 31 & 32 & 33 & \\ 41 & 42 & 43 & 44 \end{matrix}$$

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 $\mathbf{ap}(i + (j-1) \times (2n-j)/2)$.

Usage

LAPACK:

```
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:

```
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)
```

SSPEVX/DSPEVX/CHPEVX/ZHPEVX 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.

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 $\mathbf{vl} < \lambda \leq \mathbf{vu}$ are to be computed.
range = 'I' or 'i': Eigenvalues λ_{il} 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. $\mathbf{n} \geq 0$.

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', $\mathbf{ap}(i + (j-1) \times j/2) = A(i,j)$ for $1 \leq i \leq j$;

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

vl If **range** = 'V' or 'v', the left endpoint of the interval of selected eigenvalues. Only eigenvalues $\lambda > \mathbf{vl}$ are returned. $\mathbf{vl} < \mathbf{vu}$.

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 $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$, 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 $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$, 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 $b-a \leq$ **abstol**$+\epsilon\max(|a|,|b|)$, where ϵ is the machine epsilon. If **abstol** ≤ 0, then $\epsilon\|T\|_1$ is used in its place, where T is the matrix obtained by reducing A to tridiagonal form. For most problems, this is the appropriate level of accuracy to request. For certain strongly graded matrices, greater accuracy can be obtained in very small eigenvalues by setting **abstol** to some very small positive number.

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**.

SSPEVX/DSPEVX/CHPEVX/ZHPEVX Continued

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 10), 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 — SSTEVX/DSTEVX

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, $i = 1, 2, \ldots, n$, for which there exist corresponding nonzero vectors, z_i, $i = 1, 2, \ldots, n$, 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	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:

```
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 SSTEVX (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 DSTEVX (jobz, range, n, d, e, vl, vu, il, iu, abstol, m, w,
            z, ldz, work, iwork, ifail, info)
```

SSTEVX/DSTEVX Continued

LAPACK8:

```
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 SSTEVX (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 $\mathbf{vl} < \lambda \leq \mathbf{vu}$ are to be computed.
range = 'I' or 'i': Eigenvalues λ_{il} through λ_{iu} are to be computed.

n The order in the matrix A. $\mathbf{n} \geq 0$.

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 $\lambda > \mathbf{vl}$ are returned. $\mathbf{vl} < \mathbf{vu}$.

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 $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$, 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 $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$, only eigenvalues λ_i with $i \leq \mathbf{iu}$ are returned. $\min(\mathbf{il},\mathbf{n}) \leq \mathbf{iu} \leq \mathbf{n}$.

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 epsilon. If $\mathbf{abstol} \leq 0$, then $\epsilon \|A\|_1$ is used in its place, where A is the input tridiagonal matrix. For most problems, this is the appropriate level of accuracy to request. For certain strongly graded matrices, greater accuracy can be obtained in very small eigenvalues by setting **abstol** to some very small positive number.

	ldz	The leading dimension of array **z** in the calling program unit. **ldz** ≥ 1, and if **jobz** = 'V' or 'v', then **ldz** ≥ **n**.
Working Storage	**work, iwork**	Arrays used for work space.
Output	**d**	Destroyed.
	e	Destroyed.
	m	The total number of selected eigenvalues found. 0 ≤ **m** ≤ **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 10), 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'.

SSYEVX/DSYEVX/CHEEVX/ZHEEVX Symmetric or Hermitian Matrix

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, $i = 1, 2, \ldots, n$, for which there exist corresponding nonzero vectors, z_i, $i = 1, 2, \ldots, n$, 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:

```
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)
```

LAPACK8:

```
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_{il} 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.

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**).

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 $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$, only eigenvalues λ_i with $i \geq$ **il** are returned. **il** ≥ 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 $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$, only eigenvalues λ_i with $i \leq$ **iu** are returned. min(**il**,**n**) ≤ **iu** ≤ **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 $b-a \leq$ **abstol**$+\epsilon\max(|a|,|b|)$, where ϵ is the machine epsilon. If **abstol** ≤ 0, then $\epsilon\|T\|_1$ is used in its place, where T is the matrix obtained by reducing A to tridiagonal form. For most problems, this is the appropriate level of accuracy to request. For certain strongly graded matrices, greater accuracy can be obtained in very small eigenvalues by setting **abstol** to some very small positive number.

ldz The leading dimension of array **z** in the calling program unit. **ldz** ≥ max(1,**n**).

lwork The length of array **work**. For subprograms SSYEVX and DSYEVX, **lwork** ≥ max(1,7**n**). For subprograms CHEEVX and ZHEEVX, **lwork** ≥ max(1,2**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 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 ≤ **m** ≤ **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 10), 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'.

8

Drivers for Generalized Eigenvalues Problems

Overview

This chapter explains how to use LAPACK subprograms to compute the eigenvalues or eigenvalues and eigenvectors of a generalized symmetric or Hermitian definite eigenproblem 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.

Refer to Chapters 6 and 7 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 *CONVEX VECLIB User's Guide* 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 $\lambda \epsilon \mathbb{C}$ 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 $i = 1, 2, ..., n$.

Then λ_i is an eigenvalue and x_i is an eigenvector of the generalized eigenproblem $Ax = \lambda Bx$. The eigenvectors are B-orthogonal: $x_i^*Bx_i = 1$ and $x_i^*Bx_j = 0$ 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.

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

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 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 $x^T Bx$ 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 eigenvalues of A are scalars λ_i, $i = 1, 2, \ldots, n$, for which there exist corresponding nonzero vectors, z_i, $i = 1, 2, \ldots, n$, such that

$$Ax = \lambda Bx \quad \text{or} \quad ABx = \lambda x \quad \text{or} \quad BAx = \lambda x.$$

Optionally, the 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

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**$(i + (j-1) \times (2n-j)/2)$.

SSPGV/DSPGV/CHPGV/ZHPGV Continued

Usage

LAPACK:

```
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:

```
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** ≥ 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 $1 \le i \le j$;

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

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 $1 \le i \le j$;

If **uplo** = 'L' or 'l', **bp**($i + (j-1) \times (2n-j)/2$) = $B(i,j)$ for $j \le i \le n$.

ldz The leading dimension of array **z** in the calling program unit. **ldz** ≥ 1, and if **jobz** = 'V' or 'v', then **ldz** ≥ **n**.

Working Storage

work, rwork Arrays used for work space.

Output

ap Destroyed.

bp On exit with **info** ≤ **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** ≤ **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 ≤ **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.

SSPGV/DSPGV/CHPGV/ZHPGV Continued

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 10), 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 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 eigenvalues of A are scalars λ_i, $i = 1, 2, \ldots, n$, for which there exist corresponding nonzero vectors, z_i, $i = 1, 2, \ldots, n$, such that

$$Ax = \lambda Bx \quad \text{or} \quad ABx = \lambda x \quad \text{or} \quad BAx = \lambda x$$

Optionally, the 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:

```
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)
```

SSYGV/DSYGV/CHEGV/ZHEGV Continued

LAPACK8:

```
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** ≥ 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** ≥ 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** ≥ max(1,**n**).

lwork The length of array **work**. **lwork** ≥ max(1,3**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, rwork Arrays used for work space. On 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** ≤ **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** ≤ **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** = $-k$, the k-th argument had an invalid value.
info > 0: If **info** = k ≤ **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 10), may be replaced with a user-supplied version to change the error procedure. Error conditions are

itype ≠ 1, 2, or 3,
jobz ≠ 'N' or 'n' or 'V' or 'v',
uplo ≠ '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'.

9

Drivers for the Singular Value Decomposition

Overview

This chapter explains how to use LAPACK subprograms to compute the singular value decomposition of a matrix.

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

SGESVD/DGESVD/CGESVD/ZGESVD SVD of a General Matrix

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:

```
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*max(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*max(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:

```
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*max(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)
```

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 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** ≥ **m**.

If **jobu** = 'S' or 's', then min(**m**,**n**) left singular vectors are returned and **ucol** ≥ 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 10), may be replaced with a user-supplied version to change the error procedure. Error conditions are

jobu ≠ 'A' or 'a' or 'S' or 's' or 'O' or 'o' or 'N' or 'n',
jobvt ≠ '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'.

10

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 CONVEX LAPACK and will exist in subsequent releases of the product. The operations covered are:

- choosing 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 $\mathbb{R}^n$, the vector space of n-dimensional real vectors, is a function $f:\mathbb{R}^n \rightarrow \mathbb{R}$ that has the following properties:

$f(x) \geq 0$	$x \in \mathbb{R}^n$
$f(x) = 0$	if and only if $x = 0$
$f(x+y) \leq f(x)+f(y)$	$x, y \in \mathbb{R}^n$
$f(\alpha x) = \lvert\alpha\rvert f(x)$	$\alpha \in \mathbb{R}, x \in \mathbb{R}^n$

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 10-1.

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 $\mathbb{R}^{m\times n}$, the vector space of m-by-n real matrices, is a function $f:\mathbb{R}^{m\times n} \to \mathbb{R}$ that has the following properties:

$$\begin{array}{ll} f(A) \geq 0 & A \in \mathbb{R}^{m\times n} \\ f(A) = 0 & \text{if and only if } A = 0 \\ f(A+B) \leq f(A)+f(B) & A, B \in \mathbb{R}^{m\times n} \\ f(\alpha A) = |\alpha| f(A) & \alpha \in \mathbb{R}, A \in \mathbb{R}^{m\times n} \end{array}$$

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 $\|\ \|_\alpha$ be a vector norm on $\mathbb{R}^m$, $\|\ \|_\beta$ be a vector norm on $\mathbb{R}^n$, and $\|\ \|_{\alpha,\beta}$ be a matrix norm on $\mathbb{R}^{m\times n}$. The matrix norm is said to be ***consistent with*** the vector norm if

$$\|Ax\|_\beta \leq \|A\|_{\alpha,\beta} \|x\|_\alpha.$$

Let $\|\ \|_\alpha$ be a vector norm on $\mathbb{R}^m$, $\|\ \|_\beta$ be a vector norm on $\mathbb{R}^n$, and $\|\ \|_{\alpha,\beta}$ be a matrix norm on $\mathbb{R}^{m\times n}$. 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 $\|\ \|_\alpha$, $\|\ \|_\beta$, and $\|\ \|_\gamma$ be matrix norms on $\mathbb{R}^{m\times q}$, $\mathbb{R}^{m\times n}$, and $\mathbb{R}^{n\times q}$, respectively. Then the norms are *consistent* if the submultiplicative property

$$\|AB\|_\alpha \leq \|A\|_\beta \|B\|_\gamma$$

is satisfied for all $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{n\times q}$.

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 10-1: 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 a 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

ILAENV — Choose Problem-Dependent Parameters

Purpose

ILAENV is called by various LAPACK subprograms to set problem-dependent parameters.

Usage

LAPACK:

```
CHARACTER*(*) name, opts
INTEGER*4     ispec, n1, n2, n3, n4
INTEGER*4     ivalue, ILAENV
ivalue = ILAENV (ispec, name, opts, n1, n2, n3, n4)
```

LAPACK8:

```
CHARACTER*(*) name, opts
INTEGER*8     ispec, n1, n2, n3, n4
INTEGER*8     ivalue, ILAENV
ivalue = ILAENV (ispec, name, opts, n1, n2, n3, n4)
```

Input

ispec Specifies which parameter is to be returned as the value of ILAENV, as follows:

- **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(**m**,**n**)/min(**m**,**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.

name The name of the calling subprogram in either all uppercase or all lowercase characters.

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'.

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.

Output

ivalue The function value is the value of the requested problem-dependent parameter, or an error code, as follows:

- **ivalue** ≥ 0: The value of the problem parameter specified by **ispec**.
- **ivalue** < 0: If **ivalue** = $-k$, the k-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)
```

SLANGB/DLANGB/.../ZLANGB Compute Norm of General Band Matrix

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:

```
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:

```
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, SLANGB
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** ≥ 0.

kl The number of subdiagonals within the band of A. **kl** ≥ 0.

ku The number of superdiagonals within the band of A. **ku** ≥ 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: **ab**(**ku**+1+i-j,j) = $A(i,j)$ for $\max(1, j-\mathbf{ku}) \le i \le \min(\mathbf{n}, j+\mathbf{kl})$

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

SLANGB/DLANGB/CGBCON/ZLANGB Continued

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'.

Compute Norm of General Matrix — SLANGE/DLANGE/.../ZLANGE

Purpose These subprograms compute a norm of a general m-by-n matrix A.

Usage

LAPACK:

```
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:

```
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** ≥ 0.

n The number of columns of the matrix A. **n** ≥ 0.

SLANGE/DLANGE/CLANGE/ZLANGE Continued

	a	The **m**-by-**n** matrix A.
	lda	The leading dimension of array **a** in the calling program unit. **lda** ≥ 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'.

Compute Norm of General Tridiagonal Matrix — SLANGT/.../ZLANGT

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:

```
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)
```

SLANGT/DLANGT/CLANGT/ZLANGT — Continued

LAPACK8:

```
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** ≥ 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'.

Compute Norm of Symmetric or Hermitian Band Matrix SLANSB/...

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 **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 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	*	*

SLANSB/DLANSB/CLANHB/CLANSB/ZLANHB/ZLANSB Continued

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**$(kd+1+i-j,j)$. 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:

```
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:

```
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)
```

Continued SLANSB/DLANSB/CLANHB/CLANSB/ZLANHB/ZLANSB

```
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 $\max(1,j\text{-}\mathbf{kd}) \leq i \leq j$;

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

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'.

SLANSP/... Compute Norm of Symmetric or Hermitian Packed Matrix

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:

SLANSP or DLANSP	A is a real symmetric packed matrix.
CLANSP or ZLANSP	A is a complex symmetric packed matrix.
CLANHP or ZLANHP	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

$$\begin{matrix} 11 & 12 & 13 & 14 \\ & 22 & 23 & 24 \\ & & 33 & 34 \\ & & & 44 \end{matrix}$$

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

$$\begin{matrix} 11 & & & \\ 21 & 22 & & \\ 31 & 32 & 33 & \\ 41 & 42 & 43 & 44 \end{matrix}$$

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:

```
CHARACTER*1 norm, uplo
INTEGER*4   n
REAL*4      ap((n*(n+1))/2), work(n)
REAL*4      anorm, SLANSP
anorm = SLANSP (norm, uplo, n, ap, work)
```

```
CHARACTER*1 norm, uplo
INTEGER*4   n
REAL*8      ap((n*(n+1))/2), work(n)
REAL*8      anorm, DLANSP
anorm = DLANSP (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, CLANHP
anorm = CLANHP (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, CLANSP
anorm = CLANSP (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, CLANSP
anorm = ZLANSP (norm, uplo, n, ap, rwork)
```

LAPACK8:

```
CHARACTER*1 norm, uplo
INTEGER*8   n
REAL*8      ap((n*(n+1))/2), work(n)
REAL*8      anorm, SLANSP
anorm = SLANSP (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, CLANHP
anorm = CLANHP (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, CLANSP
anorm = CLANSP (norm, uplo, n, ap, rwork)
```

SLANSP/DLANSP/CLANHP/CLANSP/ZLANHP/ZLANSP Continued

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. $\mathbf{n} \geq 0$.

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', $\mathbf{ap}(i + (j-1) \times j/2) = A(i,j)$ for $1 \leq i \leq j$;

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

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'.

Compute Norm of Symmetric or Hermitian Tridiagonal Matrix SLANST/...

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	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:

```
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)
```

SLANST/DLANST/CLANHT/ZLANHT Continued

```
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:

```
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** ≥ 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'.

Compute Norm of Symmetric or Hermitian Matrix SLANSY/.../ZLANSY

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:

```
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)
```

SLANSY/DLANSY/CLANHE/CLANSY/ZLANHE/ZLANSY Continued

```
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:

```
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.

Continued SLANSY/DLANSY/CLANHE/CLANSY/ZLANHE/ZLANSY

	lda	The leading dimension of array **a** in the calling program unit. **lda** ≥ 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'.

XERBLA — Error Handler

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, a call traceback is also written onto the standard error file. 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:

```
CHARACTER*6 name
INTEGER*4   iarg
CALL XERBLA (name, iarg)
```

LAPACK8:

```
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

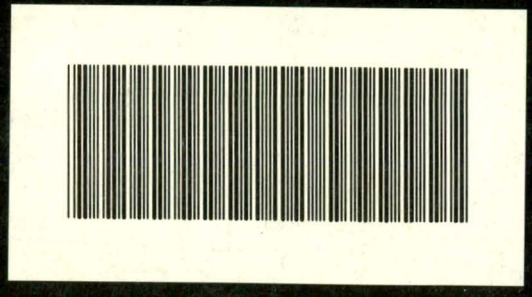
D

E

F

G

H



Order Number
DSW-036

Document Number
720-005630-000