



Scientific Computing Software Library
(SCSL) User's Guide

007-4325-001

© 2003 Silicon Graphics, Inc. All Rights Reserved. This manual or parts thereof may not be reproduced in any form unless permitted by contract or by written permission of Silicon Graphics, Inc.

LIMITED RIGHTS LEGEND

The electronic (software) version of this document was developed at private expense; if acquired under an agreement with the USA government or any contractor thereto, it is acquired as "commercial computer software" subject to the provisions of its applicable license agreement, as specified in (a) 48 CFR 12.212 of the FAR; or, if acquired for Department of Defense units, (b) 48 CFR 227-7202 of the DoD FAR Supplement; or sections succeeding thereto. Contractor/manufacturer is Silicon Graphics, Inc., 1600 Amphitheatre Pkwy 2E, Mountain View, CA 94043-1351.

Silicon Graphics, SGI, the SGI logo, and IRIX are registered trademarks of Silicon Graphics, Inc. in the United States and other countries worldwide.

Cray is a registered trademark of Cray, Inc. Intel is a registered trademark of Intel Corporation. Linux is a registered trademark of Linus Torvalds, used with permission by Silicon Graphics, Inc. MIPS, R4000, R4400, and R8000 are registered trademarks and MIPSpro and R10000 are trademarks of MIPS Technologies, Inc. and are used under license by Silicon Graphics, Inc. Red Hat and all Red Hat-based trademarks are trademarks or registered trademarks of Red Hat, Inc., in the United States and other countries. UNIX and the X device are registered trademarks of The Open Group in the United States and other countries. X/Open is a trademark of X/Open Company Ltd. All other trademarks are the property of their respective owners.

Portions of this product and document are derived from material copyrighted by Kuck and Associates, Inc.

Record of Revision

Version	Description
001	December 2003 Initial release

Contents

About This Guide	xiii
Related publications	xiii
Related Operating System Documentation	xiii
Tuning and Application Guides	xiii
Third Party Documentation	xiv
Conventions	xvi
Obtaining Publications	xvii
Reader Comments	xvii
1. Introduction	1
Parallel Processing Issues	2
2. Basic Linear Algebra Subprogram (BLAS) Routines	5
Data Types	5
C Interface to the BLAS Routines	6
Increment Arguments	8
Array Storage (BLAS 2 and BLAS 3)	8
Level 1 BLAS Routines	8
Level 2 BLAS Routines	10
Level 3 BLAS Routines	12
3. LAPACK	13
LAPACK and SCSL	13
Naming Scheme for Individual Routines	14
Types of Problems Solved by LAPACK	15

Solving Linear Systems	16
Factoring a Matrix	20
Error Codes	24
Solving from the Factored Form	25
Condition Estimation	26
Use in Error Bounds	27
Equilibration	29
Iterative Refinement	31
Error Bounds	32
Inverting a Matrix	33
Solving Least Squares Problems	34
Orthogonal Factorizations	34
Multiplying by the Orthogonal Matrix	37
Generating the Orthogonal Matrix	38
Comparing Answers	39
4. Using Sparse Linear Equation Solvers	41
Sparse Matrices	41
Solution Techniques	43
Direct Methods	43
How Direct Solvers Work	43
Iterative Solvers	44
How Iterative Methods Work	45
5. Signal Processing Routines	47
FFT Routines	47
Data Types	47
Implementation Details	48
Supported Routines	49

Implementation Notes: <i>work</i> and <i>table</i> arrays	51
Implementation Notes: <i>isys</i> Parameter Array	53
Implementation Notes: Scratch Space	53
Convolution and Correlation Routines	54
Appendix A. Supported SCSL Routines	57
Introductory Man Pages	57
BLAS Routines	57
FFT Routines	61
LAPACK Routines	62
Glossary	139
Index	145

Tables

Table 3-1	Factorization forms	21
Table 3-2	Verification tests for LAPACK (all should be $O(1)$)	40

Examples

Example 3-1	LU factorization	21
Example 3-2	Symmetric indefinite matrix factorization	22
Example 3-3	Error conditions	24
Example 3-4	Roundoff errors	26
Example 3-5	Hilbert matrix	31
Example 3-6	Orthogonal factorization	35

About This Guide

This publication describes the SGI Scientific Computing Software Library (SCSL) which runs on SGI IRIX and Linux systems. The information in this manual supplements the man pages provided with the SCSL release.

This document is a user's guide for programmers. Readers should have a working knowledge of the IRIX and Linux operating systems, have an understanding of the Fortran and C programming languages, and have a working familiarity with scientific and mathematical theories.

Related publications

The following publications provide information that can supplement the information in this document.

Release notes for Linux systems are stored in
`/usr/share/doc/sgi-scsl-versionnumber/README.relnotes`.

Related Operating System Documentation

The following documents provide information about IRIX and Linux implementations on SGI systems:

- *Linux Installation and Getting Started*
- *Linux Resource Administration Guide*
- *IRIX Admin: Resource Administration*
- *SGI ProPack for Linux Start Here*
- *Message Passing Toolkit: MPI Programmer's Manual*

Tuning and Application Guides

The following documents provide information about the applications used on IRIX and Linux systems and about tuning issues on those systems:

- *Origin 2000 and Onyx2 Performance Tuning and Optimization Guide*

- *Linux Application Tuning Guide*
- *MIPSpro Fortran 77 Programmer's Guide*
- *MIPSpro Fortran 90 Commands and Directives Reference Manual*
- *C++ Programmer's Guide*
- *Guide to SGI Compilers and Compiling Tools*
- *ProDev WorkShop: Overview*

The following documentation is provided for the compilers and performance tools which run on SGI Linux systems:

- http://sources.redhat.com/gdb/onlinedocs/gdb_toc.html
- <http://intel.com/software/perflib>; documentation for Intel compiler products can be downloaded from this website.
- <http://developer.intel.com/software/products/vtune/vtune61/index.htm/>
- Information about the OpenMP Standard can be found at <http://www.openmp.org/specs>.

Third Party Documentation

The following publications provide detailed information about the topics discussed in this manual. In many cases, these documents are referenced specifically in this manual.

- Anderson, E., Z. Bai, et al. *LAPACK User's Guide*. Philadelphia SIAM, 1999. This manual is available online at <http://www.netlib.org/lapack/lug/index.html>.
- Anderson, Edward, Jack Dongarra, and Susan Blackford. Installation guide for LAPACK. LAPACK Working Note 41, Technical Report CS-91-138. University of Tennessee (Feb. 1992).
- Argham, Nicolas J. *Accuracy and Stability of Numeric Algorithms*. Philadelphia SIAM, 1996.
- Arioli, M., J. W. Demmel, and I. S. Duff. Solving sparse linear systems with sparse backward error. *SIAM J. Matrix Anal. Appl.* 10 (1989).

- Ashcraft, Cleve. A vector implementation of the multifrontal method for large sparse, symmetric positive definite linear systems. Technical Report ETA-TR-51. Boeing Computer Services, 1987.
- Duff, I. S., A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices*. Monographs on Numerical Analysis. New York: Oxford University Press, 1986.
- George, Alan and Joseph W-H Liu. *Computer Solution of Large Sparse Positive Definite Systems*. Prentice-Hall Series in Computational Mathematics. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1981.
- Golub, Gene and James M. Ortega. *Scientific Computing: An Introduction with Parallel Computing*. Boston: Academic Press, 1993.
- Golub, Gene H. and Charles F. Van Loan. *Matrix Computations*. 2nd edition. Baltimore, Maryland: Johns Hopkins University Press, 1989.
- Hageman, Louis A. and David M. Young. *Applied Iterative Methods*. Computer Science and Applied Mathematics. New York and London: Academic Press, 1981.
- Heroux, Michael A. A reverse communication interface for “matrix-free” preconditioned iterative solvers. Edited by C.A. Brebbia, D. Howard, and A. Peters *In Applications of Supercomputers in Engineering II*, 207-213. Boston: Computational Mechanics Publications, 1991.
- Heroux, Michael A., Phuong Vu, and Chao Wu Yang. A parallel preconditioned conjugate gradient package for solving sparse linear systems on a Cray Y-MP. *Applied Numerical Mathematics*, 8 (1991).
- Hestenes, M. R. and E. Stiefel. Methods of conjugate gradients for solving linear systems. *J. Res. National Bureau of Standards* 49 (1952): 409-436.
- Kincaid, David R., Thomas C. Oppe, John R. Respass, and David M. Young. *ITPACKV 2C User's Guide*. Technical Report CNA-191. The University of Texas at Austin: Center for Numerical Analysis, (Nov. 1984).
- Manteuffel, T. A. An incomplete factorization technique for positive definite linear systems. *Math. Comp.* 34 (1980): 473-497.
- Oppe, Thomas C., Wayne D. Joubert, and David R. Kincaid. *NSPCG User's Guide*. The University of Texas at Austin: Center for Numerical Analysis, (Dec. 1988).
- Reid, J. K., editor. On the Method of Conjugate Gradients for the Solution of Large Sparse Systems of Linear Equations. *Large Sparse Sets of Linear Equations*, Academic Press, 1971.

- Saad, Youcef. Practical use of polynomial preconditionings for the conjugate gradient method., 6(4) (Oct. 1985): 865-881.
- Saad, Youcef and Martin H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal of Scientific and Statistical Computing*, 7(3) (Jul. 1986): 856-869.
- Sonneveld, Peter. CGS, a fast lanczos-type solver for nonsymmetric linear systems. *SIAM Journal of Scientific and Statistical Computing*, 10(1) (Jan. 1989): 36-52.
- Stewart, G. W. *Introduction to Matrix Computations*. Orlando, Florida: Academic Press, 1973.
- Wilkinson, J. H. *The Algebraic Eigenvalue Problem*. Oxford, England: Oxford University Press, 1965.
- Yang, Chao W. A parallel multifrontal method for sparse symmetric definite linear systems on the Cray Y-MP. *Proceedings of the Fifth SIAM Conference on Parallel Processing for Scientific Computing*. Houston, Texas (Apr. 1992).

You can find a good general reference on the solution of sparse linear systems in Golub and Van Loan. You can find a good introduction to direct and iterative methods, as well as methods for special linear systems, in these texts. See the special section of the November 1989 issue of the *SIAM Journal of Scientific and Statistical Computing*, pages 1135-1232 for an updated general reference.

See George and Liu, Duff and Erisman, and Reid for classical references that give a thorough and in-depth treatment of sparse direct solvers. Another common reference is Ashcraft.

The original conjugate gradient algorithm was presented in Hestenes and Stiefel; however, Reid presented the first practical application. A classical text in iterative methods is that of Hageman and Young. You can find good discussions of the biconjugate gradient and biconjugate gradient squared methods in Sonneveld. GMRES is presented by Saad and Schultz.

Conventions

The following conventions are used throughout this documentation:

`command`

This fixed-space font denotes literal items, such as pathnames, man page names, commands, and programming language structures.

<i>variable</i>	Italic typeface denotes variable entries and words or concepts being defined.
[]	Brackets enclose optional portions of a command line.

Obtaining Publications

You can obtain SGI documentation as follows:

- See the SGI Technical Publications Library at <http://docs.sgi.com>. Various formats are available. This library contains the most recent and most comprehensive set of online books, release notes, man pages, and other information.
- If it is installed on your SGI system, you can use InfoSearch, an online tool that provides a more limited set of online books, release notes, and man pages. With an IRIX system, enter `infosearch` at a command line or select **Help > InfoSearch** from the Toolchest.
- On IRIX systems, you can view release notes by entering either `grelnotes` or `relnotes` at a command line.
- On Linux systems, you can view release notes on your system by accessing the `README.txt` file for the product. This is usually located in the `/usr/share/doc/productname` directory, although file locations may vary.
- You can view man pages by typing `man title` at a command line.

Reader Comments

If you have comments about the technical accuracy, content, or organization of this publication, contact SGI. Be sure to include the title and document number of the publication with your comments. (Online, the document number is located in the front matter of the publication. In printed publications, the document number is located at the bottom of each page.)

You can contact SGI in any of the following ways:

- Send e-mail to the following address:
`techpubs@sgi.com`
- Use the Feedback option on the Technical Publications Library Web page:

<http://docs.sgi.com>

- Contact your customer service representative and ask that an incident be filed in the SGI incident tracking system.
- Send mail to the following address:

Technical Publications
SGI
1500 Crittenden Lane, M/S 535
Mountain View, California 94043-1351

SGI values your comments and will respond to them promptly.

Introduction

This manual describes the SGI Scientific Computer Software Library, which runs on SGI IRIX and Linux systems. The information in this manual supplements the man pages provided with SCSL and provides details about the implementation and usage of these library routines.

SCSL contains the following groups of routines:

- Vector-vector linear algebra subprograms (Level 1 BLAS routines).
- Matrix-vector linear algebra subprograms (Level 2 BLAS routines).
- Matrix-matrix linear algebra subprograms (Level 3 BLAS routines).
- LAPACK routines for the solution of dense linear systems of equations, linear least-squares problems, eigenvalue problems and singular value decomposition.
- Direct linear solvers for real and complex sparse systems with symmetric non-zero structure, and iterative solvers for real sparse systems with arbitrary structure.
- Signal processing routines, which include Fast Fourier Transform (FFT) routines, convolution routines, and correlation routines.
- 64-bit thread-safe parallel random number generators.

The SCSL routines are loaded by using the `-lscs` option or the `-lscs_mp` options to the compiler command line. The `-lscs_mp` option directs the linker to use the multi-processor version of the library.

When linking with SCSL, the default integer size is 4 bytes (32 bits). Another version of SCSL is available in which integers are 8 bytes (64 bits). This version allows the users access to larger memory sizes. It can be loaded by using the `-lscs_i8` option or the `-lscs_i8_mp` option. A program can use only one of the two versions; 4-byte integer and 8-byte integer library calls cannot be mixed.

Many SCSL routines are multitasked or multithreaded; this means that a program that calls a multitasked routine will run in parallel mode and take advantage of multiple processors whenever possible, even if the program has not specifically requested multitasking. If a significant percentage of time is spent in the routine, this feature can significantly reduce wall-clock time.

Note that most LAPACK routines do not perform multiprocessing, but almost all LAPACK routines call Level 2 BLAS and Level 3 BLAS that do multiprocessing.

This manual includes the following sections:

- Chapter 2, "Basic Linear Algebra Subprogram (BLAS) Routines" on page 5, discusses the Basic Linear Algebra Subprogram (BLAS) routines.
- Chapter 3, "LAPACK" on page 13, discusses the LAPACK routines and their implementation on SGI Linux systems.
- Chapter 4, "Using Sparse Linear Equation Solvers" on page 41, discusses sparse matrices and solution techniques for sparse linear systems.
- Chapter 5, "Signal Processing Routines" on page 47, discusses the Fast Fourier Transform (FFT) routines.

Parallel Processing Issues

Parallel processing is a method of splitting a computational task into subtasks, and then simultaneously performing the subtasks. In many cases, the use of specialized libraries, such as SCSL, is a key component of parallel processing.

Parallel processing can eliminate idle CPU time because the workload is divided among all CPUs; therefore, the amount of work performed per unit time (the *throughput*) increases. However, parallel processing also introduces some overhead into program execution. In some cases, you may be able to reduce wall-clock time, but at the cost of extra CPU time which increases because more machine resources are used.

By using parallel processing, you can alleviate some of the following common problems:

- Maximum-memory jobs: if the memory is occupied by a few large-memory jobs, one or more of the CPUs might be idle even though there are other jobs to run.
- Dedicated machine: if the computer is running a single job, then all other CPUs are idle.
- Light workload: if the amount of jobs waiting for a CPU is less than the total number of CPUs, then one or more of the CPUs becomes idle.

With parallel processing, the additional CPUs reduce the wall-clock time instead of sitting idle. Even when very little idle time exists, using additional CPUs can still lead to benefits.

Parallel processing introduces some overhead into program execution. This subsection discusses some of the common types of overhead introduced by parallel processing:

- Multitasked programs require more memory than unitasked programs, and they can contain more code, more temporary variables, and can require additional stack space.
- Multitasked jobs can be swapped more often, and remain swapped longer, on a heavily loaded production system.
- Processors are forced to wait on semaphores during the process of synchronization.
- Overhead is incurred when slave processors are acquired (on entry to a parallel region) and at synchronization points within parallel regions. Tests show that the overhead of executing extra autotasking code adds a nominal 0% to 5% to the overall execution time.
- If inner-loop autotasking is used, vector performance can decrease because of shorter vector lengths and more vector loop startups.
- Processors are sometimes held for the next parallel region to improve efficiency. While holding a processor can save time, it also costs time to acquire and hold them.

Because overhead is associated with work distribution, jobs with large granularity have less partitioning than smaller jobs. Large jobs, however, may have problems with load balancing.

Parallel processing implementation strategies are discussed in detail in the following books:

- *Linux Application Tuning Guide*
- *Origin 2000 and Onyx2 Performance Tuning and Optimization Guide*

In addition to these books, other documents in the MIPSpro compiler documentation set discuss parallel processing issues that are specific to compiler use. See the *Guide to SGI Compilers and Compiling Tools* for information about those books.

Basic Linear Algebra Subprogram (BLAS) Routines

The SCSL BLAS routines are a library of routines that perform basic operations involving matrices and vectors. The BLAS are used in a wide range of software, including LINPACK, LAPACK, and many other algorithms commonly in use today. They have become a de facto standard for elementary vector and matrix operations.

There are three 'levels' of BLAS routines:

- **Level 1:** these routines perform vector-vector operations such as dot-product and the adding of a multiple of one vector to another.
- **Level 2:** these routines perform matrix-vector operations that occur frequently in the implementation of many of the most common linear algebra algorithms. Note that algorithms that use Level 2 BLAS can be very efficient on vector computers, but are not well suited to computers with a hierarchy of memory (that is, cache memory).
- **Level 3:** these routines are used for matrix-matrix operations.

See the remaining subsections in this chapter for details about each type of BLAS.

BLAS 2 and BLAS 3 modules in SCSL are optimized and parallelized to take advantage of SGI's hardware architecture. Best performance is achieved with BLAS 3 routines where outer-loop unrolling and blocking techniques have been applied to take advantage of the memory cache.

SCSL's LAPACK algorithms make extensive use of BLAS 3 modules and are more efficient than the older, BLAS 1-based LINPACK algorithms.

Data Types

The BLAS routines use the following data types:

- Single precision: Fortran "real" data types, C/C++ "float" data types, 32-bit floating point. These routine names begin with *s*.
- Single precision complex: Fortran "complex" data type, C/C++ "scsl_complex" data type (defined in `<scsl_blas.h>`), C++ STL "complex<float>" data type

(defined in `<complex.h>`), two 32-bit floating point reals. These routine names begin with C.

- Double precision: Fortran “double precision” data type, C/C++ “double” data type, 64-bit floating point. These routine names begin with D.
- Double precision complex: Fortran “double complex” data type, C/C++ “`sctl_zcomplex`” data type (defined in `<sctl_blas.h>`), C++ STL “`complex<double>`” data type (defined in `<complex.h>`), two 64-bit floating point doubles. These routine names begin with Z.

The `man(1)` command can find a man page online by either the single precision, single precision complex, double precision, or double precision complex name, as shown in the following table:

	Single Precision	Double Precision	Single Precision Complex	Double Precision Complex
form:	Sname	Dname	Cname	Zname
example:	SGEMM	DGEMM	CGEMM	ZGEMM

C Interface to the BLAS Routines

SCSL supports two different C interfaces to the BLAS:

- The C interface described in individual BLAS man pages follows the same conventions used for the C interface to the SCSL signal processing library.
- SCSL also supports the C interface to the legacy BLAS set forth by the BLAS Technical Forum. This interface supports row-major storage of multidimensional arrays; see the `INTRO_CBLAS(3S)` man page for details.

By default, the integer arguments are 4 bytes (32 bits) in size; this is the size obtained when the SCSL library is linked with `-lscs` or `lscs_mp`. Another version of SCSL is available, however, in which integers are 8 bytes (64 bits). This version allows the user access to larger memory sizes and helps when porting legacy Cray codes. It can be loaded by using either the `-lscs_i8` or `-lscs_i8_mp` link option. Any program may use only one of the two versions; 4-byte integer and 8-byte integer library calls cannot be mixed.

C/C++ function prototypes for Level 1 BLAS routines are provided in `<scsl_blas.h>`, when using the default 4-byte integers, and in `<scsl_blas_i8.h>` when using 8-byte integers. These header files define the complex types `scsl_complex` and `scsl_zcomplex`, which are used in the prototypes. Alternatively, C++ programs may declare arguments using the types `complex<float>` and `complex<double>` from the standard template library. But if these types are used, `<complex.h>` must be included before `<scsl_blas.h>` (or `<scsl_blas_i8.h>`). Both complex types are equivalent: they simply represent (real, imaginary) pairs of floating point numbers stored contiguously in memory. With the proper casts, you can simply pass arrays of floating point data to the routines where complex arguments are expected.

Casts, however, can be avoided. The header files `<scsl_blas.h>` and `<scsl_blas_i8.h>` directly support the use of user-defined complex types or disabling prototype checking for complex arguments completely. By defining the symbol `SCSL_VOID_ARGS` before including `<scsl_blas.h>` or `<scsl_blas_i8.h>` all complex arguments will be prototyped as `void *`. To define the symbol `SCSL_VOID_ARGS` at compile time use the `-D` compiler option (for example, `-DSCSL_VOID_ARGS`) or use an explicit `#define SCSL_VOID_ARGS` in the source code. This allows the use of any complex data structure without warnings from the compiler, provided the structure is the following:

1. The real and imaginary components must be contiguous in memory.
2. Sequential array elements must also be contiguous in memory

While this allows the use of non-standard complex types without generating compiler warnings, it has the disadvantage that the compiler does not catch type mismatches.

Strong type checking can be enabled employing user-defined complex types instead of SCSL's standard complex types. To do this, define `SCSL_USER_COMPLEX_T=my_complex` and `SCSL_USER_ZCOMPLEX_T=my_zcomplex`, where `my_complex` and `my_zcomplex` are the names of user-defined complex types. These complex types must be defined before including the `<scsl_blas.h>` or `<scsl_blas_i8.h>` header file.

Fortran 90 users on IRIX systems can perform compile-time checking of SCSL BLAS subroutine and function calls by adding `USE SCSL_BLAS` (for 4-byte integer arguments) or `USE SCSL_BLAS_I8` (for 8-byte integer arguments) to the source code from which the BLAS calls are made. Alternatively, the compile-time checking can be invoked without any source code modifications by using the `-auto_use` compiler option, as in the following example:

```
% f90 -auto_use SCSL_BLAS test.f -lscs
% f90 -auto_use SCSL_BLAS_I8 -i8 test.f -lscs_i8
```

Increment Arguments

A vector's description consists of the name of the array (x or y) followed by the storage spacing (increment) in the array of vector elements ($incx$ or $incy$). The increment can be positive or negative. When a vector x consists of n elements, the corresponding actual array arguments must be of a length at least $1 + (n-1) * |incx|$. For a negative increment, the first element of x is assumed to be $x(1 + (n-1) * |incx|)$ for Fortran arrays, $x[(n-1) * |incx|]$ for C/C++ arrays. The standard specification of `_SCAL`, `_NRM2`, `_ASUM`, and `I_AMAX` does not define the behavior for negative increments, so this functionality is an extension to the standard BLAS.

Note that setting an increment argument to 0 can cause unpredictable results.

Array Storage (BLAS 2 and BLAS 3)

Multidimensional arrays passed as arguments to BLAS routines must be stored in column-major order, the storage convention used in Fortran programs. C and C++ users must explicitly store multidimensional arrays column-by-column.

One way to do this is to reverse the order of array dimensions with respect to the Fortran declaration (for example, $x(ldx, n)$ in Fortran versus $x[n][ldx]$ in C/C++). Because of the prototypes used in `<scsl_blas.h>`, the array should be cast as a pointer to the appropriate type when passed as an argument to a BLAS routine in order to avoid potential compiler type mismatch errors or warning messages.

C and C++ users who want to employ row-major storage for multidimensional arrays when calling the BLAS routines should see the `INTRO_CBLAS(3S)` man page for details.

Level 1 BLAS Routines

The Level 1 BLAS routines perform vector-vector linear algebra operations. The following types of vector-vector operations are available:

- Dot products and various vector norms
- Scaling, copying, swapping, and computing linear combinations of vectors

- Generating or applying plane or modified plane rotations.

You should use Fortran type declarations for functions. Declaring the data type of the complex Level 1 BLAS functions is important because, based on the first letter of the name of the routine and the Fortran data typing rules, the default implied data type would be REAL.

Fortran type declarations for function names are as follows:

Type	Function Name
REAL	SASUM, SCASUM, SCNRM2, SDOT, SNRM2, SSUM
COMPLEX	CDOTC, CDOTU, CSUM
DOUBLE PRECISION	DASUM, DZASUM, DDOT, DNRM2, DZNRM2, DSUM
DOUBLE COMPLEX	ZDOTC, ZDOTU, ZSUM
INTEGER	ISAMAX, IDAMAX, ICAMAX, IZAMAX, ISAMIN, IDAMIN, ISMAX, IDMAX, ISMIN, IDMIN

The following routines are available in the SCSL BLAS 1:

- SASUM, DASUM: Sums the absolute values of the elements of a real vector (also called the 1 norm).
- SCASUM, DZASUM: Sums the absolute values of the real and imaginary parts of the elements of a complex vector.
- SAXPBY*, DAXPBY*, CAXPBY*, ZAXPBY*: Adds a scalar multiple of a real or complex vector to a scalar multiple of another vector.
- SAXPY, DAXPY, CAXPY, ZAXPY: Adds a scalar multiple of a real or complex vector to another vector.
- SCOPY, DCOPY, CCOPY, ZCOPY: Copies a real or complex vector into another vector.
- CDOTC, ZDOTC: Computes a dot product of the conjugate of a complex vector and another complex vector.
- SHAD*, DHAD*, CHAD*, ZHAD*: Computes the Hadamard product of two vectors.
- SNRM2, DNRM2: Computes the Euclidean norm (also called l_2 norm) of a real vector.
- SCNRM2, DZNRM2: Computes the Euclidean norm (l_2 norm) of a complex vector. 2
- CSROT*, ZDROT*, CROT*, ZROT*: Applies a real plane rotation to a pair of complex vectors.

- SROT, DROT: Applies an orthogonal plane rotation.
- SROTG, DROTG, CROTG*, ZROTG*: Constructs a Givens plane rotation.
- SROTM, DROTM: Applies a modified Givens plane rotation.
- SROTMG, DROTMG: Constructs a modified Givens plane rotation.
- SSCAL, DSCAL, CSCAL, ZSCAL, CSSCAL, ZDSCAL: Scales a real or complex vector.
- SSUM*, DSUM*, CSUM*, ZSUM*: Sums the elements of a real or complex vector.
- SSWAP, DSWAP, CSWAP, ZSWAP: Swaps two real or two complex vectors.
- ISAMAX, IDAMAX, ICAMAX, IZAMAX: Searches a vector for the first occurrence of the maximum absolute value.
- ISAMIN*, IDAMIN*: Searches a vector for the first occurrence of the minimum absolute value.
- ISMAX*, IDMAX*: Searches a vector for the first occurrence of the maximum value.
- ISMIN*, IDMIN*: Searches a vector for the first occurrence of the minimum value.

Level 2 BLAS Routines

The Level 2 BLAS routines perform matrix-vector linear algebra operations. The following routines are available:

- CHBMV, ZHBMV: Multiplies a complex vector by a complex Hermitian band matrix.
- CHEMV, ZHEMV: Multiplies a complex vector by a complex Hermitian matrix.
- CHER, ZHER: Performs Hermitian rank 1 update of a complex Hermitian matrix.
- CHER2, ZHER2: Performs Hermitian rank 2 update of a complex Hermitian matrix.
- CHPMV, ZHPMV: Multiplies a complex vector by a packed complex Hermitian matrix.
- CHPR, ZHPR: Performs Hermitian rank 1 update of a packed complex Hermitian matrix.
- CHPR2, ZHPR2: Performs Hermitian rank 2 update of a packed complex Hermitian matrix.

- SGBMV, DGBMV, CGBMV, ZGBMV: Multiplies a real or complex vector by a real or complex general band matrix.
- SGEMV, DGEMV, CGEMV, ZGEMV: Multiplies a real or complex vector by a real or complex general matrix.
- SGER, DGER: Performs rank 1 update of a real general matrix.
- CGERC, ZGERC: Performs conjugated rank 1 update of a complex general matrix.
- CGERU, ZGERU: Performs unconjugated rank 1 update of a complex general matrix.
- SGESUM*, DGESUM*, CGESUM*, ZGESUM*: Adds a scalar multiple of a real or complex matrix to a scalar multiple of another real or complex matrix.
- SSBMV, DSBMV: Multiplies a real vector by a real symmetric band matrix.
- SSPMV, DSPMV, CSPMV*, ZSPMV*: Multiplies a real or complex vector by a real or complex symmetric packed matrix.
- SSPR, DSPR, CSPR*, ZSPR*: Performs symmetric rank 1 update of a real or complex symmetric packed matrix.
- SSPR2, DSPR2: Performs symmetric rank 2 update of a real symmetric packed matrix.
- SSMV, DSMV, CSMV*, ZSMV*: Multiplies a real or complex vector by a real or complex symmetric matrix.
- SSYR, DSYR, CSYR*, ZSYR*: Performs symmetric rank 1 update of a real or complex symmetric matrix.
- SSYR2, DSYR2: Performs symmetric rank 2 update of a real symmetric matrix.
- STBMV, DTBMV, CTBMV, ZTBMV: Multiplies a real or complex vector by a real or complex triangular band matrix.
- STBSV, DTBSV, CTBSV, ZTBSV: Solves a real or complex triangular band system of equations.
- STPMV, DTPMV, CTPMV, ZTPMV: Multiplies a real or complex vector by a real or complex triangular packed matrix.
- STPSV, DTPSV, CTPSV, ZTPSV: Solves a real or complex triangular packed system of equations.

- STRMV, DTRMV, CTRMV, ZTRMV: Multiplies a real or complex vector by a real or complex triangular matrix.
- STRSV, DTRSV, CTRSV, ZTRSV: Solves a real or complex triangular system of equations.

Level 3 BLAS Routines

The Level 3 BLAS routines perform matrix-matrix linear algebra operations. The following routines are available:

- SGEMM, DGEMM, CGEMM, ZGEMM: Multiplies a real or complex general matrix by a real or complex general matrix.
- CGEMM3M*, ZGEMM3M*: Multiplies a complex general matrix by a complex general matrix, using 3 real matrix multiplications and 5 matrix additions.
- DGEMMS*: Multiplies a double precision general matrix by a double precision general matrix, using a variation of Strassen's algorithm.
- SSYMM, DSYMM, CSYMM, ZSYMM: Multiplies a real or complex general matrix by a real or complex symmetric matrix.
- CHEMM, ZHEMM: Multiplies a complex general matrix by a Hermitian matrix.
- SSYR2K, DSYR2K, CSYR2K, ZSYR2K: Performs symmetric rank 2k update of a real or complex symmetric matrix.
- CHER2K, ZHER2K: Performs Hermitian rank 2k update of a complex Hermitian matrix.
- SSYRK, DSYRK, CSYRK, ZSYRK: Performs symmetric rank k update of a real or complex symmetric matrix.
- CHERK, ZHERK: Performs Hermitian rank k update of a complex Hermitian matrix.
- STRMM, DTRMM, CTRMM, ZTRMM: Multiplies a real or complex general matrix by a real or complex triangular matrix.

LAPACK

LAPACK is a public domain library of subroutines for solving dense linear algebra problems, including systems of linear equations, linear least squares problems, eigenvalue problems, and singular value decomposition problems. It has been designed for efficiency on high-performance computers.

LAPACK is the successor to LINPACK and EISPACK. It uses today's high-performance computers more efficiently than the older packages and it extends the functionality of these packages by including equilibration, iterative refinement, error bounds, and driver routines for linear systems. It also includes routines for computing and reordering the Schur factorization, and condition estimation routines for eigenvalue problems.

Performance issues are addressed by implementing the most computationally-intensive algorithms using the Level 2 and 3 Basic Linear Algebra Subprograms (BLAS). Because most of the BLAS were optimized in single and multiple-processor environments, these algorithms give near optimal performance.

LAPACK and SCSL

All routines from LAPACK 3.0 are included in SCSL. This includes driver routines, computational routines, and auxiliary routines for solving linear systems, least squares problems, and eigenvalue and singular value problems. See the `INTRO_LAPACK(3S)` man page for details about the routines that are available in the current release of SCSL.

Online man pages are available for individual LAPACK subroutines. For example, to view a description of the calling sequence for the subroutine to perform the LU factorization of a real matrix, see the `DGETRF(3S)` man page. The user interface to all supported LAPACK routines is the same as the standard LAPACK interface.

Several enhancements improve the performance of the LAPACK routines on SGI systems. For example, the LU, Cholesky, and QR factorization routines are redesigned for better performance and scalability when running multiple processes.

Tuning parameters for the block algorithms provided in SCSL are set within the `ILAENV` LAPACK routine. `ILAENV` is an integer function subprogram that accepts information about the problem type and problem dimensions and returns a single integer parameter such as the optimal block size, the minimum block size for which a

block algorithm should be used, or the crossover point (the problem size at which it becomes more efficient to switch to an unblocked algorithm). Setting tuning parameters occurs without user intervention, but users can call `ILAENV` directly to check the values to be used.

Naming Scheme for Individual Routines

The name of each LAPACK routine is a coded specification of its function. All driver and computational routines have five or six character names of the form `XYZZZ` or `XYZZZZ`. The first letter, `X`, indicates the data type:

- **S**: real
- **D**: double precision
- **C**: complex
- **Z**: double complex

The next two letters, `YY`, indicate the type of matrix or the most significant matrix type. Most of these two letter codes apply to both real and complex matrices, but some apply specifically to only one or the other. The following list shows all matrix types:

BD	BiDiagonal
DI	Diagonal
GB	General Band
GE	GEneral (nonsymmetric)
GG	General matrices, Generalized problem
GT	General Tridiagonal
HB	Hermitian Band (complex only)
HE	HErmitian (possibly indefinite) (complex only)
HG	Hessenberg matrix, Generalized problem
HP	Hermitian Packed (possibly indefinite) (complex only)
HS	upper HeSsenberg
OP	Orthogonal Packed (real only)
OR	ORthogonal (real only)
PB	Positive definite Band (symmetric or Hermitian)
PO	POsitive definite (symmetric or Hermitian)
PP	Positive definite Packed (symmetric or Hermitian)
PT	Positive definite Tridiagonal (symmetric or Hermitian)
SB	Symmetric Band (real only)
SP	Symmetric Packed (possibly indefinite)

ST	Symmetric Tridiagonal
SY	SYmmetric (possibly indefinite)
TB	Triangular Band
TG	Triangular matrices, Generalized problem
TP	Triangular Packed
TR	TRiangular
TZ	TrapeZoidal
UN	UNitary (complex only)
UP	Unitary Packed (complex only)

The last letters, ZZ or ZZZ, indicate the computation performed. For example, TRF is a **Triangular Factorization**.

See the `INTRO_LAPACK(3s)` man page for details about the types of computations performed and a list of supported routines.

Types of Problems Solved by LAPACK

LAPACK routines can solve systems of linear equations, linear least squares problems, eigenvalue problems, and singular value problems. LAPACK routines can also handle many associated computations such as matrix factorizations and estimating condition numbers. Dense and banded matrices are provided for, but not general sparse matrices.

This subsection discusses the LAPACK routines for solving the following two basic problems:

- Computing the unique solution to a linear system $AX = B$, where the coefficient matrix A is dense, banded, triangular, or tridiagonal, and the matrix B may contain one or more right-hand sides.
- Computing a least squares solution to an overdetermined system $AX = B$, where A is $m \times n$ with $m \geq n$, or a minimum norm solution to an underdetermined system $AX = B$, where A is $m \times n$ with $m < n$.

See "Solving Linear Systems" on page 16, and "Solving Least Squares Problems" on page 34, for a discussion of the software used to solve these problems. The orthogonal transformation routines described in "Solving Least Squares Problems" on page 34, also have application in eigenvalue and singular value computations.

There are three classes of LAPACK routines: LAPACK *driver routines* solve a complete problem; LAPACK *computational routines* perform one step of the computation; and LAPACK *auxiliary routines* perform certain subtask or low-level computation.

The driver routines generally call the computational routines to do their work, and offer a more convenient interface; therefore, LAPACK users are advised to use a driver routine if there is one that meets their requirements.

Man pages for both driver and computational routines are available with SCSL. A list of the auxiliary routines, with a brief description, can be found in the LAPACK User's Guide.

Solving Linear Systems

Finding the solution of a system of simultaneous linear equations is one of the most frequently encountered problems in scientific computing.

A linear system of equations (n equations with n unknowns) can be written as follows:

$$\begin{aligned} a_{11} x_1 + a_{12} x_2 + \dots + a_{1n} x_n &= b_1 \\ a_{12} x_1 + a_{22} x_2 + \dots + a_{2n} x_n &= b_2 \\ &\dots\dots\dots \\ a_{n1} x_1 + a_{n2} x_2 + \dots + a_{nn} x_n &= b_n \end{aligned}$$

Equation 3-1

This system can also be written in the form $Ax = b$ where A is a square matrix of order n , b is a given column vector of n components and x is an unknown column vector of n components.

For example, the following linear equations:

$$\begin{aligned} 2x + y &= 4 \\ -x + y &= 1 \end{aligned}$$

Equation 3-2

can be written in matrix/vector notation as the following:

$$\begin{bmatrix} 2 & 1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 4 \\ 1 \end{bmatrix}$$

Equation 3-3

The solution to this system of equations is the set of vectors $[x, y]^T$ that satisfy both equations. The physical interpretation of this system of equations is that it represents two lines in the (x, y) plane, which may intersect in one point, no points (if they are parallel), or an infinite number of points (if the two equations are multiples of each other).

To solve the system, it is helpful to simplify the system as much as possible. A standard method for doing this (Gaussian elimination) is to use the following elementary operations:

- interchange any two equations
- multiply an equation by a nonzero constant
- add a multiple of one equation to any other one

This reduces the system to a triangular form. The system obtained after each operation will be equivalent to the original system and therefore have the same solution.

For illustration consider the following system of linear equations:

$$\begin{aligned}x + y + z &= 0 \\x + 2y + 4z &= -1 \\x + 3y + 9z &= 2\end{aligned}$$

Equation 3-4

Subtract the first equation from the second equation and subtract the first equation from the third one. The result is the following system:

$$\begin{aligned}x + y + z &= 0 \\y + 3z &= -1 \\2y + 8z &= 2\end{aligned}$$

Equation 3-5

Note the first variable x no longer appears in the second or third equation. Similarly, the second variable y can be eliminated from the third equation by subtracting 2 times the second equation from the third equation to obtain the following:

$$\begin{aligned}x + y + z &= 0 \\y + 3z &= -1 \\2z &= 4\end{aligned}$$

Equation 3-6

The resulting system is upper triangular and it can easily be deduced that:

$$\begin{aligned}2z = 4 &\Rightarrow z = 2 \\y + 3z = -1 &\Rightarrow y = (-1 - 3 \times 2) = -7 \\x + y + z = 0 &\Rightarrow x = 7 - 2 = 5\end{aligned}$$

Equation 3-7

To represent these steps in matrix form, let $A=A1$, $A2$, and $A3$ represent the matrices of the systems (Equation 3-4, Equation 3-5, Equation 3-6, respectively), for example:

$$A2 = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 3 \\ 0 & 2 & 8 \end{bmatrix}$$

Equation 3-8

Then $A2$ is obtained from $A1$ by subtracting the first row from the second row and subtracting the first row from the third row. This means that $A2$ can be obtained by pre-multiplying $A1$ by a suitable matrix, and in this example, it is easy to verify that

$$A2 = M1 \times A1$$

Equation 3-9

that is:

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 3 \\ 0 & 2 & 8 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \\ -1 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 3 & 9 \end{bmatrix}$$

Equation 3-10

In a similar way

$$A3 = M2 \times A2$$

Equation 3-11

or

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 3 \\ 0 & 0 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 3 \\ 0 & 2 & 8 \end{bmatrix}$$

Equation 3-12

Combining Equation 3-9 and Equation 3-11 we have

$$A3 = M2 \times M1 \times A$$

Equation 3-13Therefore, to solve the system $Ax=b$, we only need to calculate the following:

$$c = M2 \times M1 \times b$$

Equation 3-14

and solve the upper triangular system

$$A3 \times x = M2 \times M1 \times A \times x = M2 \times M1 \times b = c$$

Equation 3-15Moreover, $M1$ and $M2$ are nonsingular matrices so:

$$A = M1^{-1} \times M2^{-1} \times A3$$

Equation 3-16

Because $M1$ and $M2$ are unit lower triangular, so is the product of their inverses. Therefore, A can be written as the product of a lower triangular matrix $L=M1^{-1} \times M2^{-1}$ and an upper triangular matrix $U=A3$. Equation 3-16 becomes $A=LU$, which is the factorization of the coefficient matrix A .

The LAPACK routines for solving linear systems assume the system is already in a matrix form. The data type (real or complex), characteristics of the coefficient matrix (general or symmetric, and positive definite or indefinite if symmetric), and the storage format of the matrix (dense, band, or packed), determine the routines that should be used.

Factoring a Matrix

Most of the techniques in LAPACK are based on a matrix factorization as the first step. There are two main types of factorization forms:

- explicit: The actual factors are returned. For example, the Cholesky factorization routine `DPOTRF(3S)`, with `UPLO = 'L'`, returns a matrix L such that $A = LL^T$.
- factored form: The factorization is returned as a product of permutation matrices and triangular matrices that are low-rank modifications of the identity. For example, the diagonal pivoting factorization routine `DSYTRF(3S)`, with `UPLO = 'L'`, computes a factorization of the following form:

$$A = (P_1 L_1 P_2 L_2 \dots P_n L_n) D (P_1 L_1 P_2 L_2 \dots P_n L_n)^T$$

Equation 3-17

where each P_i is a rank-1 permutation, each L_i is a rank-1 or rank-2 modification of the identity, and D is a diagonal matrix with 1-by-1 and 2-by-2 diagonal blocks.

Generally, users do not have to know the details of how the factorization is stored, because other LAPACK routines manipulate the factored form.

Regardless of the form of the factorization, it reduces the solution phase to one that requires only permutations and the solution of triangular systems. For example, the LU factorization of a general matrix, $A = PLU$, is used to solve for X in the system of equations $AX = B$ by successively applying the inverses of P , L , and U to the right-hand side:

1. $X \leftarrow PB$
2. $X \leftarrow L^{-1}X$
3. $X \leftarrow U^{-1}X$

In the last two steps, the inverse of the triangular factors is not computed, but triangular systems of the form $LY = Z$ and $UX = Y$ are solved instead.

The following table lists the factorization forms for each of the factorization routines for real matrices. The factorization forms differ for DGETRF and DGBTRF, even though both compute an LU factorization with partial pivoting. You can also obtain the same factorizations through the LAPACK driver routines (for instance, DGESV or DGESVX).

Table 3-1 Factorization forms

Name	Form	Equation	Notes
DGBTRF	Factored form	$A = LU$	L is a product of permutations and unit lower triangular matrices L_i ; L_i differs from the identity matrix only in column i .
DGTRF	Factored form	$A = LU$	
DGETRF	Explicit	$A = PLU$	
DPBTRF	Explicit	$A = LL^T$ or $A = U^T U$	
DPOTRF	Explicit	$A = LL^T$ or $A = U^T U$	
DPPTRF	Explicit	$A = LL^T$ or $A = U^T U$	
DPTTRF	Explicit	$A = LDL^T$ or $A = U^T D U$	
DSPTRF	Factored form	$A = LDL^T$ or $A = UDU^T$	L (or U) is a product of permutations and block unit lower (upper) triangular matrices L_i (U_i); L_i (U_i) differs from the identity matrix only in the one or two columns that correspond to the 1-by-1 or 2-by-2 diagonal block D_i .
DSYTRF	Factored form	$A = LDL^T$ or $A = UDU^T$	

Example 3-1 LU factorization

The DGETRF subroutine performs an LU factorization with partial pivoting ($A = PLU$) as the first step in solving a general system of linear equations $AX = B$. If DGETRF is called with the following:

$$A = \begin{bmatrix} 4. & 9. & 2. \\ 3. & 5. & 7. \\ 8. & 1. & 6. \end{bmatrix}$$

Equation 3-18

details of the factorization are returned, as follows:

$$A = \begin{bmatrix} 8. & 1. & 6. \\ 0.5 & 8.5 & -1.0 \\ 0.375 & 0.5441 & 5.294 \end{bmatrix}$$

$$IPIV = [3, 3, 3]$$

Equation 3-19

Matrices L and U are given explicitly in the lower and upper triangles, respectively, of A :

$$L = \begin{bmatrix} 1. & & \\ 0.5 & 1. & \\ 0.375 & 0.5441 & 1. \end{bmatrix}, U = \begin{bmatrix} 8. & 1. & 6. \\ & 8.5 & -1.0 \\ & & 5.4294 \end{bmatrix}$$

Equation 3-20

The $IPIV$ vector specifies the row interchanges that were performed. $IPIV(1) = 3$ implies that the first and third rows were interchanged when factoring the first column; $IPIV(2) = 3$ implies that the second and third rows were interchanged when factoring the second column. In this case, $IPIV(3)$ must be 3 because there are only three rows. Thus, the permutation matrix is the following:

$$P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

Equation 3-21

Generally, the pivot information is used directly from $IPIV$ without constructing matrix P .

Example 3-2 Symmetric indefinite matrix factorization

`DSYTRF` factors a symmetric indefinite matrix A into one of the forms $A = LDL^T$ or $A = UDU^T$, where L and U are lower triangular and upper triangular matrices, respectively, in factored form, and D is a diagonal matrix with 1-by-1 and 2-by-2

diagonal blocks. To illustrate this factorization, choose a symmetric matrix that requires both 1-by-1 and 2-by-2 pivots:

$$A = \begin{bmatrix} 9. & & & \\ 5. & 7. & & \\ -16. & 12. & 3. & \\ 4. & -2. & 10. & 8. \end{bmatrix}$$

Equation 3-22

Only the lower triangle of A is specified because the matrix is symmetric, but you could have specified the upper triangle instead. The output from DSYTRF is the following:

$$A = \begin{bmatrix} 9. & & & \\ -16. & 3. & & \\ -0.9039 & -0.8210 & 21.37 & \\ -0.7511 & -0.6725 & 0.4597 & 13.21 \end{bmatrix}$$

$$IPIV = [-3, -3, 3, 4]$$

Equation 3-23

The signs of the indices in the `IPIV` vector indicate that a 2-by-2 pivot block was used for the first two columns, and 1-by-1 pivots were used for the third and fourth columns. Therefore, D must be the following:

$$D = \begin{bmatrix} 9. & & & \\ -16. & 3. & & \\ & & 21.37 & \\ & & & 13.21 \end{bmatrix}$$

Equation 3-24

Matrix L is supplied in factored form as $L = P_1 L_1 P_2 L_2$, where the parts of each L_i that differ from the identity are stored in A below their corresponding blocks D_i :

$$P_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, L_1 = \begin{bmatrix} 1. & & & \\ 0. & 1. & & \\ -0.9039 & -0.8210 & 1 & \\ -0.7511 & -0.6725 & 0.0 & 1.0 \end{bmatrix}$$

Equation 3-25

$$P_2 = I, L_2 = \begin{bmatrix} 1 & & & \\ 0 & 1 & & \\ 0 & 0 & 1.0 & \\ 0 & 0 & 0.4597 & 1 \end{bmatrix}$$

Equation 3-26

Error Codes

The LAPACK routines always check the arguments on entry for incorrect values. If an illegal argument value is detected, the error-handling subroutine XERBLA is called. XERBLA prints a message similar to the following to standard error, and then it aborts:

```
** On entry to DGETRF parameter number 4 had an illegal value
```

All other errors in the LAPACK routines are described by error codes returned in *info*, the last argument. The values returned in *info* are routine-specific, except for *info* = 0, which always means that the requested operation completed successfully.

For example, an error code of *info* > 0 from DGETRF means that one of the diagonal elements of the factor *U* from the factorization $A = PLU$ is exactly 0. This indicates that one of the rows of *A* is a linear combination of the other rows, and the linear system does not have a unique solution.

Example 3-3 Error conditions

If DGETRF is given the matrix

$$A = \begin{bmatrix} 1. & 4. & 7. \\ 2. & 5. & 8. \\ 3. & 6. & 9. \end{bmatrix}$$

Equation 3-27

it returns

$$A = \begin{bmatrix} 3. & 6. & 9. \\ 0.3333 & 2. & 4. \\ 0.6667 & 0.5 & 0. \end{bmatrix}$$

$$IPIV = [3, 3, 3]$$

Equation 3-28

which corresponds to the factorization

$$P = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}, L = \begin{bmatrix} 1.0 & & \\ 0.3333 & 1.0 & \\ 0.6667 & 0.5 & 1. \end{bmatrix}, U = \begin{bmatrix} 3. & 6. & 9. \\ & 2. & 4. \\ & & 0. \end{bmatrix}$$

Equation 3-29

On exit from DGETRF, $info = 3$, indicating that $U(3,3)$ is exactly 0. This is not an error condition for the factorization because the factors that were computed satisfy $A = PLU$, but the factorization cannot be used to solve the system.

Solving from the Factored Form

In LAPACK, the solution step is generally separated from the factorization. This allows the matrix factorization to be reused if the same coefficient matrix appears in several systems of equations with different right-hand sides. If the number of right-hand sides is also large, it is often more efficient to separate the solve from the factorization. The typical usage is found in the driver routine DGESV, which solves a general system of equations $AX = B$ by using two subroutine calls, the first to factor the matrix A and the second to solve the system, using the factored form:

```
CALL DGETRF( N, N, A, LDA, IPIV, INFO )
IF( INFO.EQ.0 ) THEN
CALL DGETRS( 'No transpose', N, NRHS, A, LDA,
$           IPIV, B, LDB, INFO )
END IF
```

As shown, you should always check the return code from the factorization to see whether it completed successfully and did not produce any singular factors. To obtain further information about proceeding with the solve, estimate the condition number (see "Condition Estimation" on page 26, for details).

Because most of the LAPACK driver routines do their work in the LAPACK computational routines, a call to a driver routine gives the same performance as separate calls to the computational routines. The exceptions are the simple driver routines used for solving tridiagonal systems: `SGTSV`, `SPTS`, `CGTSV`, and `CPTS`. These routines compute the solution while performing the factorization for certain numbers of right-hand sides. Because the amount of work in each loop is small, some reloading of constants and loop overhead is saved by combining the factorization with part of the solve.

Condition Estimation

A return code of `info = 0` from a factorization routine indicates that the triangular factors have nonzero diagonals. The linear system still may be too ill-conditioned to give a meaningful solution.

One indicator that you can examine before computing the solution is the reciprocal condition number, `RCOND`. The condition number, defined as $\kappa(A) = \|A\| \|A^{-1}\|$, tells how much the relative errors in A and b are magnified in the solution x . `DGECN` and the other condition estimation routines compute `RCOND = 1/κ(A)` by using the exact 1-norm or infinity-norm of A and an estimate for the norm of A^{-1} , because computing the inverse explicitly would be very expensive, and the inverse may not even exist. By convention, if A^{-1} does not exist, $\kappa(A) = \infty$ and `RCOND` should be computed as 0 or a small number on the order of ϵ , the machine epsilon.

If the condition number is large (that is, if `RCOND` is small), small errors in A and b may lead to large errors in the solution x . The rule of thumb is that the solution loses one digit of accuracy for every power of 10 in the condition number, assuming that the elements of A all have about the same magnitude. For example, a condition number of 100 (`RCOND = 0.01`) implies that the last 2 digits are inaccurate; a condition number of $1/\epsilon$ (`RCOND < ε`, the machine epsilon which is approximately 1.1×10^{-14} on Altix systems) implies that all of the digits have been lost. This value for the machine epsilon assumes a model of rounded floating-point arithmetic with base $\beta = 2$ and $t = 53$ mantissa digits, and $\epsilon = \beta^{(1-t)}$.

The expert driver routine `DGESVX` uses this rule of thumb to decide whether the solution and error bounds should be computed. If `RCOND` is less than the machine epsilon, `DGESVX` returns `info = N+1`, indicating that the matrix A is singular to working precision, and it does not compute the solution.

Example 3-4 Roundoff errors

The matrix

$$A = \begin{bmatrix} 1. & 2. & 3. \\ 4. & 5. & 6. \\ 7. & 8. & 9. \end{bmatrix}$$

Equation 3-30

is singular in exact arithmetic, but on Altix systems, DGETRF returns

$$A = \begin{bmatrix} 7. & 8. & 9. \\ 0.1429 & 0.8571 & 1.714 \\ 0.5714 & 0.5 & -1.586. \times 10^{-16} \end{bmatrix}$$

Equation 3-31

where $IPIV=[3,3,3]$ and $info = 0$. In exact arithmetic, $A(3,3)$ would have been 0, but roundoff error has made this entry -1.586×10^{-16} instead. The reciprocal condition number computed by DGECON is 2.20×10^{-18} , which is less than the machine epsilon of 1.11×10^{-16} . Therefore, DGESVX returns $info=4$ and does not try to solve any systems with this A .

Use in Error Bounds

You can use the condition number to compute a simple bound on the relative error in the computed solution to a system of equations $Ax = b$ (see *Introduction to Matrix Computations*, by Stewart). If x is the exact solution and \hat{x} is the computed solution, let r be the residual $r = b - Ax = A(x - \hat{x})$. If A is nonsingular:

$$x - \hat{x} = A^{-1}r$$

Equation 3-32

and

$$\|x - \hat{x}\| \leq \|A^{-1}\| \|r\|$$

Equation 3-33

From Equation 3-33 we can already see that if $\|A\|$ is large, then the error in \hat{x} may be significantly greater than the residual r .

Since $Ax = b$, it follows that $\|x\| \geq \|b\| / \|A\|$, therefore

$$\frac{\|x - \hat{x}\|}{\|x\|} \leq \|A\| \|A - 1\| \frac{\|r\|}{\|b\|}$$

Equation 3-34

Define the condition number $\kappa(A) = \|A\| \|A - 1\|$. This gives the bound

$$\frac{\|x - \hat{x}\|}{\|x\|} \leq \kappa(A) \frac{\|r\|}{\|b\|}$$

Equation 3-35

For a description of a more precise error bound based on a component-wise error analysis, see "Error Bounds" on page 32.

Another application of the condition number is to consider the computed solution $\hat{x} = x + \Delta x$ to be the exact solution of a slightly perturbed linear system:

$$(A + \Delta A)(x + \Delta x) = (b + \Delta b)$$

Equation 3-36

where ΔA is small in norm with respect to A , and Δb is small in norm with respect to b . For Gaussian elimination with partial pivoting, it has been shown that $\|\Delta A\| \leq \omega \|A\|$ and $\|\Delta b\| \leq \omega \|b\|$, where ω is the product of ε and a slowly growing function of n (see *The Algebraic Eigenvalue Problem*, by Wilkinson). Proving that an algorithm has this property is the stuff of backward error analysis, and it ensures that, if the problem is well-conditioned, the computed solution is near the exact solution of the original problem. Because $Ax = b$ (Equation 3-34) simplifies to

$$A\Delta x = \Delta b - \Delta A(x + \Delta x)$$

Equation 3-37

Assuming A is nonsingular,

$$\Delta x = A^{-1}(\Delta b - \Delta A(x + \Delta x))$$

Equation 3-38

Taking norms and dividing by $\|x\|$,

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

Equation 3-39

Using the inequality $\|b\| \leq \|A\| \|x\|$,

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

Equation 3-40

and substituting $\kappa(A) = \|A\| \|A^{-1}\|$,

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

Equation 3-41

provided $\kappa(A) \|\Delta A\| / \|A\| < 1$. In terms of the relative backward error ω ,

$$\frac{\|\Delta x\|}{\|x\|} \leq \frac{2\omega\kappa(A)}{1 - \omega\kappa(A)}$$

Equation 3-42

In "Error Bounds" on page 32, the backward error is defined slightly differently to obtain a component-wise error bound.

Equilibration

The condition number defined in the last section is sensitive to the scaling of A . For example, the matrix

$$A = \begin{bmatrix} 1. & 1 \\ 0 & 1. \times 10^{16} \end{bmatrix}$$

Equation 3-43

has as its inverse

$$A^{-1} = \begin{bmatrix} 1. & -1 \times 10^{-16} \\ 0 & 1. \times 10^{-16} \end{bmatrix}$$

Equation 3-44

and so $\text{RCOND} = 1 / (\|A\| \|A^{-1}\|) = 1. \times 10^{-16}$. If this value of RCOND is less than the machine epsilon on a system, `DGESVX` (with `FACT = 'N'`) does not try to solve a system with this matrix. However, A has elements that vary widely in magnitude, so the bounds on the relative error in the solution may be pessimistic. For example, if the right-hand side in $Ax = b$ is the following:

$$b = \begin{bmatrix} 2.0 \\ 1.0 \times 10^{16} \end{bmatrix}$$

Equation 3-45

`DGETRF` followed by `DGETRS` produces the exact answer, $x = [1., 1.]^T$.

You can improve the condition of this example by a simple row scaling. Scaling a problem before computing its solution is known as *equilibration*, and it is an option to some of the expert driver routines (those for general or positive definite matrices). Enabling equilibration does not necessarily mean it will be done; the driver routine will choose to do row scaling, column scaling, both row and column scaling, or no scaling, depending on the input data. The usage of this option is as follows:

```
CALL DGESVX('E', 'N', N, NRHS, A, LDA, AF,
$         LDAF, IPIV, EQUED, R, C, B, LDB, X, LDX,
$         RCOND, FERR, BERR, WORK, IWORK, INFO)
```

The `'E'` in the first argument enables equilibration. For this example, `EQUED = 'R'` on return, indicating that only row scaling was done, and the vector R contains the scaling constants:

$$R = \begin{bmatrix} 1.0 \\ 1.0 \times 10^{-16} \end{bmatrix}$$

Equation 3-46

The form of the equilibrated problem is:

$$(\text{diag}(R) A \text{diag}(C)) (\text{diag}(C)^{-1} X) = \text{diag}(R) B$$

Equation 3-47

or $A_E X_E = B_E$, where A_E is returned in A :

$$A \leftarrow \begin{bmatrix} 1. & 0. \\ 0. & 1.0 \times 10^{-16} \end{bmatrix} \begin{bmatrix} 1. & .1 \\ 0. & 1.0 \times 10^{-16} \end{bmatrix} = \begin{bmatrix} 1. & 1. \\ 0. & 1. \end{bmatrix}$$

Equation 3-48

and B_E is returned in B :

$$B \leftarrow \begin{bmatrix} 1. & 0. \\ 0. & 1.0 \times 10^{-16} \end{bmatrix} \begin{bmatrix} 2. \\ 1.0 \times 10^{16} \end{bmatrix} = \begin{bmatrix} 2. \\ 1. \end{bmatrix}$$

Equation 3-49

The factored form, `AF`, returns the same matrix as A in this example, because A is upper triangular, and `RCOND` = 0.3, which is the estimated reciprocal condition number for the equilibrated matrix. The only output quantity that pertains to the original problem before equilibration is the solution matrix X . In this example, X is also the solution to the equilibrated problem because no column scaling was done, but if `EQUED` had returned 'C' or 'B' and the solution to the equilibrated system were desired, it could be computed from $X_E = \text{diag}(C)^{-1} X$.

Iterative Refinement

Iterative refinement in LAPACK uses all same-precision arithmetic, a recent innovation, because it was long believed that a successful algorithm would require the residual to be computed in double precision. The following example illustrates the results of iterative refinement; "Error Bounds" on page 32, discusses the error bounds computed in the course of the algorithm.

One possible use of iterative refinement is to smooth out numerical differences between floating-point number representations. For example, a result computed on a system, which has about 13 digits of accuracy, may be improved on IEEE system, which has about 15 digits of accuracy, through the $O(n^2)$ process of iterative refinement, instead of the $O(n^3)$ process of recomputing the solution.

Example 3-5 Hilbert matrix

The classic example of an ill-conditioned matrix is the Hilbert matrix, defined by $A_{i,j} = 1/(i + j - 1)$. For example, the 5-by-5 Hilbert matrix is

$$\begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} \\ \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} & \frac{1}{9} \end{bmatrix}$$

Equation 3-50

which has a condition number of 4.77×10^5 . A rule of thumb ("Condition Estimation" on page 26) suggests almost 8 digits of accuracy in the solution are possible on systems, because $\epsilon = 1.4 \times 10^{-13}$ and $\epsilon\kappa(A) \approx 0.5 \times 10^{-8}$. If the matrix is factored using DGETRF and DGETRS is used to solve $Ax = b$, where $b = [1, 0, 0, 0, 0]^T$.

Error Bounds

In addition to performing iterative refinement on each column of the solution matrix, DGERFS and the other *xx*RFS routines also compute error bounds for each column of the solution. These bounds are returned in the real arrays FERR (forward error) and BERR (backward error), both of length NRHS. For a computed solution \hat{x} to the system of equations $Ax = b$, the forward error bound f is the following:

$$\frac{\|x - \hat{x}\|}{\|x\|} \leq f$$

Equation 3-51

and the backward error bound ω bounds the relative errors in each component of A and b in the perturbed equation 2 from "Use in Error Bounds" on page 27.

$$\|\Delta A_{i,j}\| \leq \omega \|A_{i,j}\|, \|\Delta b_i\| \leq \omega \|b_i\|$$

Equation 3-52

In Example 3-5 on page 31, the first column of the inverse of the Hilbert matrix of order 5 is computed by solving $H_5 x = e_1$, and DGERFS computed error bounds of $f \approx 1.4 \times 10^{-8}$ and $\omega \approx 9.1 \times 10^{-16}$ on systems. This provides direct information about the solution; its relative error is at most $0(10^{-8})$; therefore, the largest components of the solution should exhibit about 8 digits of accuracy, and the system for which this

solution is an exact solution is within a factor of epsilon ($\epsilon \approx 1.4 \times 10^{-14}$) from the system whose solution was attempted, so the solution is as good as the data warrants.

The component-wise relative backward error bound (equation 3) is more restrictive than the classical backward error bound $\|\Delta A\| \leq \omega \|A\|$, because it assumes $\|\Delta A\|$ has the same sparsity structure as $\|A\|$, because if $A_{i,j}$ is 0, so must be $\Delta A_{i,j}$. The backward error for the solution \hat{x} is computed from the equation

$$\omega = \max_i \frac{|r|_i}{(|A| |\hat{x}| + |b|)_i}$$

Equation 3-53

where $r = b - A\hat{x}$, and the forward error bound is computed from the equation

$$f = \frac{\| |A^{-1}| (|r| + (\gamma + 1) \epsilon (|A| |x| + |b|)) \|}{\|x\|}$$

Equation 3-54

where γ is the maximum number of nonzeros in any row of A . To avoid computing A^{-1} an approximation is used for $\| |A^{-1}| g \|$, where g is the nonnegative vector in parentheses (see Arioli, et al., for details).

Inverting a Matrix

Subroutines to compute a matrix inverse are provided in LAPACK, but they are not used in the driver routines. The inverse routines sometimes use extra workspace and always require more operations than the solve routines. For example, if there is one right-hand side in the equation $Ax = b$, where A is a square general matrix, the solves following the factorization require $2n^2$ operations, while inverting the matrix A requires $4/3n^3$ operations, plus another $2n^2$ to multiply b by A^{-1} . The inverse must be computed only once, however, and the cost can be amortized over the number of right-hand sides. Because multiplying by the inverse may be more efficient than doing a triangular solve, the extra cost to compute the inverse may be overcome if the number of right-hand sides is large.

Solving Least Squares Problems

In some applications, the best solution to a system of equations $AX = B$ that does not have a unique solution is required. If A is overdetermined, that is, A is $m \times n$ with $m \geq n$ and rank at least n , the system does not have a solution, but the linear least squares problem may have to be solved:

$$\text{minimize } \|B - AX\|_2$$

Equation 3-55

If A is underdetermined, that is, A is $m \times n$ with $m < n$, generally many solutions exist, and you may want to find the solution X with minimum 2-norm. Solving these problems requires that you first obtain a basis for the range of A , and several orthogonal factorization routines are provided in LAPACK for this purpose.

An orthogonal factorization decomposes a general $m \times n$ matrix A into a product of an orthogonal matrix Q and a triangular or trapezoidal matrix. A real matrix Q is orthogonal if $Q^T Q = I$, and a complex matrix Q is unitary if $Q^H Q = I$. The key property of orthogonal matrices for least squares problems is that multiplying a vector by an orthogonal matrix does not change its 2-norm, because

$$\|Qx\|_2 = \sqrt{x^T Q^T Q x} = \sqrt{x^T x} = \|x\|_2$$

Equation 3-56

Orthogonal Factorizations

LAPACK provides four different orthogonal factorizations for each data type. For real data, they are as follows:

- DGELQF: LQ factorization
- DGEQLF: QL factorization
- DGEQRF: QR factorization
- DGERQF: RQ factorization

Each of these factorizations can be done regardless of whether m or n is larger, but the QR and QL factorizations are most often used when $m \geq n$, and the LQ and RQ factorizations are most often used when $m \leq n$.

The QR factorization of an m -by- n matrix A for $m \geq n$ has the form

$$A = Q \begin{pmatrix} R \\ 0 \end{pmatrix}$$

Equation 3-57

where Q is an m -by- m orthogonal matrix, and R is an n -by- n upper triangular matrix. If $m > n$, it is convenient to write the factorization as

$$A = \left(Q^{(1)} \ Q^{(2)} \right) \begin{pmatrix} R \\ 0 \end{pmatrix}$$

Equation 3-58

or simply

$$A = Q^{(1)} R$$

Equation 3-59

where $Q^{(1)}$ consists of the first n columns of Q , and $Q^{(2)}$ consists of the remaining $m-n$ columns. The LAPACK routine `SGEQRF` computes this factorization. See the *LAPACK User's Guide* for details.

Example 3-6 Orthogonal factorization

The result of calling `DGEQRF` with the matrix A equal to

$$A = \begin{bmatrix} 1. & 2. & 3. \\ -3. & 2. & 1. \\ 2. & 0. & -1. \\ 3. & -1. & 2. \end{bmatrix}$$

Equation 3-60

is a compact representation of Q and R consisting of

$$A = \begin{bmatrix} -4.796 & 1.460 & -0.8341 \\ -0.5176 & -2.621 & -2.754 \\ 0.3451 & -0.03805 & 2.593 \\ 0.5176 & -0.2611 & -0.3223 \end{bmatrix}$$

$$TAU = [1.2085, 1.8698, 1.8118]$$

Equation 3-61

The matrix R appears in the upper triangle of A explicitly:

$$R = \begin{bmatrix} -4.796 & 1.460 & -0.8341 \\ 0. & -2.621 & -2.754 \\ 0. & 0. & 2.593 \end{bmatrix}$$

Equation 3-62

while the matrix Q is stored in a factored form $Q = Q_3 Q_2 Q_1$ where each Q_i is an elementary Householder transformation of the following form: $Q_i = I - \tau_i v_i v_i^T$. Each vector v_i has $v_i[0 : i - 1] = 0$, $v_i[i] = 1$, and $v[i + 1 : n]$ is stored below the diagonal in the i^{th} column of A . Therefore,

$$Q_1 = I - 1.2085 \begin{bmatrix} 1. \\ -0.5176 \\ 0.3451 \\ 0.5176 \end{bmatrix} [1. \quad -0.5176 \quad 0.3451 \quad 0.5176]$$

Equation 3-63

$$Q_2 = I - 1.8698 \begin{bmatrix} 0. \\ 1. \\ -0.03805 \\ -0.2611 \end{bmatrix} [0 \quad 1. \quad -0.03805 \quad -0.2611]$$

Equation 3-64

$$Q_3 = I - 1.8118 \begin{bmatrix} 0. \\ 0. \\ 1. \\ -0.3223 \end{bmatrix} [0 \ 0 \ 1 \ -0.3223]$$

Equation 3-65

Each transformation is orthogonal (to machine precision) because τ is chosen to have the value $2/(v^T v)$, so that

$$(I - \tau v v^T) (I - \tau v v^T) = I - 2\tau v v^T + \tau^2 (v^T v) v v^T = I$$

Equation 3-66

Multiplying by the Orthogonal Matrix

In the example of the last subsection, the elementary orthogonal matrices Q_i were expressed in terms of the scalar τ and the vector v that defines them, without multiplying out the expression. This was done to make the point that the elementary transformation is most often used in its factored form. This subsection describes one application in which multiplication by the orthogonal matrix Q is required. An alternative interface for this application is the LAPACK driver routine `DGELS`.

Given the QR factorization of a $m \times n$ matrix A with $m > n$ (Equation 3-57), if R has rank n , the solution to the linear least squares problem (Equation 3-55) is obtained by the following steps:

$$\begin{bmatrix} X \\ Y \end{bmatrix} \leftarrow Q^T B$$

$$X \leftarrow R^{-1} X$$

Equation 3-67

The LAPACK routine `DORMQR` is used in step 1 to multiply the right-hand side matrix by the transpose of the orthogonal matrix Q , using Q in its factored form. The triangular system solution in step 2 can be done using the LAPACK routine `DTRTRS`.

Continuing the example of "Orthogonal Factorizations" on page 34, suppose the right-hand side vector is $b = [1. \ 2. \ 3. \ 4.]^T$. Multiplying b by Q^T by using the LAPACK routine `DORMQR`, you get

$$\begin{bmatrix} x \\ y \end{bmatrix} = Q^T b = \begin{bmatrix} -2.711 \\ -2.273 \\ 0.5712 \\ 4.143 \end{bmatrix}$$

Equation 3-68

and after solving the triangular system with the 3-by-3 matrix R ,

$$x = \begin{bmatrix} 0.7203 \\ 0.6356 \\ 0.2203 \end{bmatrix}$$

Equation 3-69

The last $m-n$ elements of $Q^T b$ can be used to compute the 2-norm of the residual, because $\|r\|_2 \approx \|y\|_2$. Here, $\|r\|_2 = 4.143$.

Generating the Orthogonal Matrix

The `DORMxx` routines described in the previous subsection are useful for operating with the orthogonal matrix Q in its factored form, but sometimes it is the matrix Q itself that is of interest.

For example, the first step in the computational procedure for the generalized eigenvalue problem $AX = \Lambda BX$ is to find orthogonal matrices U and Z such that $U^T AZ$ is upper Hessenberg and $U^T BZ$ is upper triangular (see Golub and Van Loan for details). First, the matrix B is reduced to upper triangular form by the `QR` factorization, and A is overwritten by $Q^T A$, using the subroutines of the previous section for multiplying by the orthogonal matrix.

Next, the updated A is reduced to Hessenberg form while preserving the triangular structure of B by applying Givens rotations to A and B , alternately from the left and the right. In order to obtain the orthogonal matrices U and Z that reduce the original problem, it is necessary to keep a copy of the matrix Q from $B = QR$ and continually update it. This requires that Q be generated after the `QR` factorization.

The `DORGxx` routines generate the orthogonal matrix Q as a full matrix by expanding its factored form. Using the example of "Orthogonal Factorizations" on page 34, Q can be generated from its factored form by using the following Fortran code:

```

DO J = 1, N
  DO I = J+1, M
    Q(I, J) = A(I, J)
  END DO
END DO
CALL DORGQR(M, M, N, Q, LDQ, TAU, WORK, LWORK, INFO)

```

where the data from the QR factorization of A has been copied into a separate matrix Q because `DORGQR` overwrites its input data with the expanded matrix. The orthogonal matrix is returned in Q :

$$Q = \begin{bmatrix} -0.2085 & -0.8792 & 0.1562 & -0.3989 \\ 0.6255 & -0.4147 & 0.1465 & 0.6444 \\ -0.4170 & -0.2322 & -0.7665 & 0.4296 \\ -0.6255 & 0.03318 & 0.6054 & 0.4910 \end{bmatrix}$$

Equation 3-70

The matrix $Q^{(1)}$, consisting of only the first n columns of Q , could be generated by specifying N , rather than M , as the second argument in the call to `DORGQR`.

Comparing Answers

The results obtained by LAPACK routines should be deterministic; that is, if the same input is provided to the same subroutine in the same system environment, the output should be the same. However, because all computers operate in finite-precision arithmetic, a different order of operations may produce a different set of rounding errors, so results of the same operation obtained from different subroutines, or from the same subroutine with different numbers of processors, are not guaranteed to agree to the last decimal place.

In testing LAPACK, the test ratios in the following table were used to verify the correctness of different operations. All of these ratios give a measure of relative error. Residual tests are scaled by $1/\varepsilon$, the reciprocal of the machine precision, to make the test ratios $O(1)$, and results that are sensitive to conditioning are scaled by $1/\kappa$, where $\kappa = \|A\| \|A^{-1}\|$ is the condition number of the matrix A , as computed from the norms of A and its computed inverse A^{-1} . If a given result has a test ratio less than 30, it is judged to be as accurate as the machine precision and the conditioning of the problem will allow. See the *Installation Guide for LAPACK* for further details on the testing strategy used for LAPACK.

Table 3-2 Verification tests for LAPACK (all should be $O(1)$)

Operation or result	Test ratio
Factorization $A = LU$	$\ LU - A\ / (n \ A\ \epsilon)$
Solution \hat{x} to $Ax = b$	$\ b - A\hat{x}\ / (\ A\ \ \hat{x}\ \epsilon)$
Compared to exact soln x	$\ x - \hat{x}\ / (\ \hat{x}\ \kappa \epsilon)$
Reciprocal condition number RCOND	$\max(\text{RCOND}, \kappa) / \min(\text{RCOND}, \kappa)$
Forward error bound f	$\ x - \hat{x}\ / (\ \hat{x}\ f)$
Backward error bound ω	ω / ϵ
Computation a A^{-1}	$\ I - AA^{-1}\ / (n \ A\ \ A^{-1}\ \epsilon)$
Orthogonality check for Q	$\ I - Q^H Q\ / (n \epsilon)$

ϵ = Machine epsilon
 n = Order of matrices
 $\kappa = \kappa = \|A\| \|A^{-1}\|$

Using Sparse Linear Equation Solvers

Many techniques exist for solving sparse linear systems. The appropriate technique depends on many factors, including the mathematical and structural properties of matrix A , the dimension of A , and the number of right-hand sides b .

SGI provides two direct solvers, PSLDLT and PSLDU, and one iterative solver, DITERATIVE, for sparse linear systems of equations. These solvers are optimized and parallelized for SGI platforms.

Direct solvers of dense linear systems of equations are described on the INTRO_LAPACK(3s) man page.

This section describes some of the properties that are useful in determining a good solution technique, with some common sources of matrices with these properties.

Sparse Matrices

A *linear system* can be described as $Ax = b$, where A is an n -by- n matrix, and x and b are n dimensional vectors. A system of this kind is considered *sparse* if the matrix A has a small percentage of nonzero terms (less than 10%, often less than 1%). Large sparse linear systems occur frequently in engineering and scientific applications, and the solution of these systems (finding x given A and b) is an important and costly step.

If matrix A has a regular pattern, such as a banded or block structure, good performance can easily be obtained when solving the linear system. Good performance is much more difficult to obtain in problems in which A has no discernible pattern. The goal of the routines described in this section is to solve sparse linear systems efficiently, especially those where matrix A has no known regular pattern.

The following list defines the different types of sparse matrices:

- *symmetric positive definite matrix*: A matrix A is *symmetric* if $A = A^T$ (that is, if the coefficients of A are such that $a_{ij} = a_{ji}$ for all i and j).

A matrix A is defined to be *symmetric positive definite (SPD)* if A is symmetric and $y^T Ay > 0$ for all vectors $y \neq 0$. It is usually difficult to directly verify that a matrix is SPD; however, it can often be determined by considering the problem source. For example, many finite difference or finite volume approximations of *partial*

differential equations (PDEs) with Dirichlet or mixed boundary conditions and other mild assumptions generate SPD matrices. Also, finite element approximations with a symmetric bilinear form and equivalent test and basis functions generate SPD matrices.

SPD matrices occur frequently in applications. Optimal techniques exist to solve the related linear systems. Common sources of SPD matrices are finite element analysis of structures, the pressure correction phase of a segregated fluid dynamics simulation, and the analysis of electrical networks.

- **Diagonally dominant matrix:** A matrix A is (*strictly*) *diagonally dominant* if $|a_{ii}| > \sum_{i \neq j} |a_{ij}|$ for all i . If A is diagonally dominant, operations that involve A are often numerically stable. Common sources of diagonally-dominant matrices are simple reservoir simulation models and the velocity equations of a segregated fluid dynamics solver.
- **Structurally symmetric matrix:** If the nonzero pattern of A is symmetric, a matrix A is *structurally symmetric*; that is, $a_{ij} \neq 0$ if and only if $a_{ji} \neq 0$. The integer complexity of a solver for a structurally symmetric matrix is greatly reduced compared to a more general solver. If A is diagonally dominant, many of the optimal solution techniques for SPD matrices can be used. Common sources of these matrices are the same as those for diagonally dominant matrices.
- **Banded matrix:** If $a_{ij} = 0$ for $|i - j| > k$, matrix A is *banded*. If k is small in relation to the problem dimension n , special techniques exist for solving the related linear systems. Systems of this form usually occur in special domains with a particular ordering of the grid or node points.
- **Tridiagonal matrix:** If $a_{ij} = 0$ for $|i - j| > 1$, matrix A is *tridiagonal*. Tridiagonal matrices occur frequently in fluid dynamics and reservoir simulation.

During a simulation, an application often generates many sparse linear systems that must be solved. These are usually related to some linear approximation of a nonlinear function or some time-marching scheme for a time-dependent problem. In these cases, the linear systems are often related and information from the previous solution can be used to solve the next linear system.

For example, consider Newton's method for a nonlinear PDE. In this case, the linear system A_n is generated by evaluating the Jacobian at a certain point. A subsequent matrix, A_{n+1} , is generated using the Jacobian at a nearby point. Thus, the two matrices A_n and A_{n+1} are close to each other, and this fact is used in the solver technique. The structure of A_n and A_{n+1} is usually identical and all structural preprocessing can be done once for all related matrices.

Solution Techniques

Solution techniques for sparse linear systems can usually be divided into two broad classes: *direct methods* and *iterative methods*. The following subsections provide an overview of both classes and provide brief algorithmic descriptions.

Direct Methods

The `DPSLDT` and `ZPSLDT` routines solve sparse symmetric linear systems of the form $AX=b$ where A is a symmetric input matrix, b is an input vector of length n , and x is a vector of unknowns of length n . This solver uses a direct method. A is factored into $A = L D^{LT}$ where L is a lower triangular matrix with unit diagonal and D is a diagonal matrix.

This solver supports both real and complex double precision data types and is available in the multi-processing versions of SCSL. See the man pages for details.

`DPSLDU` and `ZPSLDU` solve sparse unsymmetric linear systems of the form $Ax=b$ where A is an input matrix with symmetric non-zero pattern but unsymmetric non-zero values, b is an input vector of length n , and x is a vector of unknowns of length n .

The unsymmetric solver uses a direct method. A is factored into $A = L D U$ where L is a lower triangular matrix with unit diagonal, D is a diagonal matrix, and U is an upper triangular matrix with unit diagonal.

The unsymmetric solver supports both real and complex double precision data types and is available in the multi-processing versions of SCSL. See the man pages for details.

How Direct Solvers Work

Direct solution methods transform matrix A into a product of several other operators so that each of the resulting operators is easy to invert for a given right-hand side b . For example, the LU factorization of A generates lower and upper triangular matrices, L and U , respectively, such that $A = LU$.

To find $x = A^{-1}b$, compute $y = L^{-1}b$ followed by $x = U^{-1}y$, both of which are straightforward computations. Direct methods are usually popular because they are considered to be very reliable. This is true if the problem dimension and the condition number of A are not too large. See *Matrix Computations*, by Golub and Van Loan for details about error bounds.

Direct methods for dense, tridiagonal, and banded matrices are quite straightforward to implement and use the three basic steps of LU factorization as mentioned previously; they may also solve for a given right-hand side without factorization. However, for general sparse matrices, the situation is considerably more complicated; in particular, the factors L and U can become extremely dense. If pivoting is required, implementing sparse factorization can use a lot of time searching lists of numbers and creating a great deal of computational overhead.

Efficient implementations can be developed, especially for SPD and symmetric pattern, positive definite matrices. See *Computer Solution of Large Sparse Positive Definite Systems*, by George and Liu, for details.

The following algorithm shows the basic steps of the sparse Cholesky solver:

Structural preprocessing phase:

1. Find P so that $\bar{A} = PAP^T$ has factor L with near minimal fill.
2. Compute symbolic factorization, that is, find structure of L .
3. Find optimal memory use and node execution sequence.

Numerical factorization phase:

4. Compute L such that $\bar{A} = LDL^T$.

Solution phase:

5. Given b , compute $y = L^{-1}b$, $z = D^{-1}y$, $x = L^{-T}z$.

In this case, A is SPD, and L and D can be found such that $A = LDL^T$, where L is a lower triangle with unit diagonal, and D is diagonal.

Steps 1 through 3 require only the nonzero structure of A . Therefore, if two matrices A_1 and A_2 have the same structure, these steps can be skipped for A_2 . Furthermore, if the same matrix A is used for more than one right-hand side b , step 5 can be repeated (or if all right-hand sides are available at once, they can be solved for simultaneously).

Iterative Solvers

The DITERATIVE solver solves sparse linear systems of the form $Ax=b$ where A is a sparse input matrix in Compressed Sparse Column (CSC) format or Compressed Sparse Row (CSR) format, b is an input vector of length n , and x is a vector of unknowns in length n .

The iterative solver uses one of four preconditioned iterative methods:

- conjugate gradient (CG) and conjugate residual (CR) for symmetric systems
- conjugate gradient squared (CGS) and BiCGSTAB, a variant of CGS with smoother convergence properties, for unsymmetric systems.

Four different types of preconditioners are available:

- Jacobi
- symmetric successive over-relaxation (SSOR)
- ILDLT (incomplete LDLT) by pattern
- ILDLT by value

The ILDLT preconditioners are only available for symmetric matrices and ILDLT by value is currently not parallelized.

Note that the iterative solver supports only real double precision data. See the `ITERATIVE(3s)` man page for details.

How Iterative Methods Work

Iterative solution methods comprise a wide variety of techniques. The solvers presented in this subsection are all in the general class of preconditioned conjugate gradient (CG) methods. These methods attempt to solve $Ax = b$ by solving an equivalent system $M^{-1}Ax = M^{-1}b$, where M is some approximation to A which is inexpensive to construct and can be easily used to compute z such that $z = M^{-1}r$.

This is left preconditioning. It is also possible to apply the preconditioner on the right side of A or on both sides. After the preconditioner is constructed and an initial approximation, x_0 to x is given, the iterative method generates a sequence of vectors $\{x_i\}$ such that x_i converges to x . Each x_i is chosen to satisfy some orthogonality or minimization condition, or both.

Unlike direct methods, iterative methods are more special-purpose. No general, effective iterative algorithms exist for an arbitrary sparse linear system. However, for certain classes of problems, an appropriate iterative method can be used to yield an approximate solution significantly faster than direct methods. Also, iterative methods usually require less memory than direct methods, thus making them the only feasible approach for large problems. See *Matrix Computations*, by Golub and Van Loan, for an introduction to these methods.

The standard preconditioned CG method, shown in the following algorithm, illustrates the basic phases common to all CG type methods used in the solvers:

Preprocessing phase:

1. Compute structure of preconditioner M .
2. Compute values of preconditioner M .
3. Analyze structure of A to optimize performance of sparse matrix vector product, $q = Ap$.

Iterative phase:

4. $r^0 = b - Ax^0$
Do $k=0, \dots$
5. $z^k = Mr^k$
6. $\gamma_k = (r^k, z^k)$
7. $\beta_k = \gamma_k / \gamma_{k-1}, \beta_0 = 0$
8. $p^k = z^k + \beta_k p^{k-1}$
9. $q^k = Ap^k$
10. $\delta_k = (q^k, p^k)$
11. $\alpha_k = \gamma_k / \delta_k$
12. $x^{k+1} = x^k + \alpha_k p^k$
13. $r^{k+1} = r^k - \alpha_k p^k$

End Do

Iterative methods are very flexible. Like direct solvers, if two matrices A_1 and A_2 have the same structure, the structural preprocessing needs to be done only once. If there are multiple right-hand sides, Steps 1 through 3 can be skipped after the first right-hand side.

Signal Processing Routines

SCSL provides three types of signal processing routines:

- Fast Fourier Transform (FFT) routines
- Convolution routines
- Correlation routines

The SCSL FFT interfaces are incompatible with those of the IRIX CHALLENGEcomplib scientific library, which has been superseded by SCSL. See the release notes that are provided with your SCSL distribution for details about converting CHALLENGEcomplib FFT calls to SCSL FFT calls.

FFT Routines

The FFT routines have been highly optimized for single-processor use. The two-dimensional, three-dimensional, and one-dimensional multiple routines are also multitasked (multithreaded) for all sizes for which there is a performance benefit. The one-dimensional routines are multitasked if the data size exceeds the size of the largest processor cache. Each routine can compute either a forward or an inverse Fourier transform.

Data Types

The following data types are used in these routines:

- Single precision: Fortran "real" data type, C/C++ "float" data type, 32-bit floating point; these routine names begin with S.
- Single precision complex: Fortran "complex" data type, C/C++ "scsl_complex" data type (defined in `<scsl_fft.h>`), C++ STL "complex<float>" data type (defined in `<complex.h>`), two 32-bit floating point reals; these routine names begin with C.
- Double precision: Fortran "double precision" data type, C/C++ "double" data type, 64-bit floating point; these routine names begin with D.
- Double precision complex: Fortran "double complex" data type, C/C++ "scsl_zomplex" data type (defined in `<scsl_fft.h>`), C++ STL

"complex<double>" data type (defined in `<complex.h>`), two 64-bit floating point doubles; these routine names begin with `Z`.

When using the C++ Standard Template Library (STL) to define complex types, the include files must be used in the following order:

```
#include <complex.h>
#include <scsl_fft.h>
```

Implementation Details

Often little or no difference exists between these versions, other than the data types of some inputs and outputs. In this case, the routines are described on the same man page, and that man page is named after the real or complex routine.

The `man(1)` command can find a man page online by either the real, complex, double precision, or double complex name.

The data types for the *scale*, *table*, and *work* arguments in these routines vary, depending on the function which is called. In the `CC`, `SC`, and `CS` routines, the arguments are single precision. In the `ZZ`, `DZ` and `ZD` routines, the arguments are double precision.

By default, the integer arguments are 4 bytes (32 bits) in size; this is the size obtained when one links to the SCSL library with `-lscs` or `-lscs_mp`. Another version of SCSL is available, however, in which integers are 8 bytes (64 bits). This version allows the user access to larger memory sizes and helps when porting legacy codes. It can be loaded by using either the `-lscs_i8` or `-lscs_i8_mp` link option. Note that any program may use only one of the two versions; 4-byte integer and 8-byte integer library calls cannot be mixed.

C/C++ function prototypes for the signal processing routines are provided in `<scsl_fft.h>`, when using the default 4-byte integers, and `<scsl_fft_i8.h>` when using 8-byte integers. These header files define the complex types `scsl_complex` and `scsl_zcomplex`, which are used in the prototypes. Alternatively, C++ programs may declare arguments using the types `complex<float>` and `complex<double>` from the standard template library (STL). But if these types are used, `<complex.h>` must be included before `<scsl_fft.h>` (or `<scsl_fft_i8.h>`). Note, though, that both complex types are equivalent: they simply represent (real, imaginary) pairs of floating point numbers stored contiguously in memory. With the proper casts, you can simply pass arrays of floating point data to the routines where complex arguments are expected.

Casts, however, can be avoided. The header files `<scsl_fft.h>` and `<scsl_fft_i8.h>` directly support the use of user-defined complex types or disabling prototype checking for complex arguments completely. By defining the symbol `SCSL_VOID_ARGS` before including `<scsl_fft.h>` or `<scsl_fft_i8.h>` all complex arguments will be prototyped as `void *`. To define the symbol `SCSL_VOID_ARGS` at compile time use the `-D` compiler option (for example, `-DSCSL_VOID_ARGS`) or use an explicit `#define SCSL_VOID_ARGS` in the source code. This allows the use of any complex data structure without warnings from the compiler, provided the structure is as described above:

1. The real and imaginary components must be contiguous in memory.
2. Sequential array elements must also be contiguous in memory.

While this allows the use of non-standard complex types without generating compiler warnings, it has the disadvantage that the compiler will not catch type mismatches.

Strong type checking can be enabled employing user-defined complex types instead of SCSL's standard complex types. To do this, define `SCSL_USER_COMPLEX_T=my_complex` and `SCSL_USER_ZOMPLEX_T=my_zomplex`, where *my_complex* and *my_zomplex* are the names of user-defined complex types. These complex types must be defined before including the `<scsl_fft.h>` (or `<scsl_fft_i8.h>`) header file.

Fortran 90 users on IRIX systems can perform compile-time checking of SCSL FFT subroutine calls by adding `USE SCSL_FFT` (for 4-byte integer arguments) or `USE SCSL_FFT_I8` (for 8-byte integer arguments) to the source code from which the FFT calls are made. Alternatively, the compile-time checking can be invoked without any source code modifications by using the `-auto_use` compiler option. For example:

```
% f90 -auto_use SCSL_FFT test.f -lscs
% f90 -auto_use SCSL_FFT_I8 -i8 test.f -lscs_i8
```

Fortran 90 users on SGI Altix systems can also perform compile-time argument checking, but in this case the `USE` statements must be explicitly incorporated into the source code.

Supported Routines

The following list describes the supported FFT routines. Rows of the table represent input and output data types for the routines in each column:

- C->C implies 32-bit complex input and output.

- Z->Z implies 64-bit double complex input and output. Each routine in this row is documented with the complex routine in the prior row.
- S->C implies 32-bit real input and 32-bit complex output.
- D->Z implies 64-bit double precision real input and 64-bit double precision complex output. Each routine in this row is documented with the real-to-complex routine in the prior row.
- C->S implies 32-bit complex input and 32-bit real output.
- Z->D implies 64-bit double complex input and 64-bit double precision output. Each routine named in this row is documented with the complex-real routine in the prior row.

Columns of the table represent the number of dimensions for which the FFT is calculated for the routines in each row:

- One-dimensional (single) calculates one FFT in one dimension.
- One-dimensional (multiple) calculates an FFT in one dimension for each column (FFTM) or row (FFTMR) of a two-dimensional matrix.
- Two-dimensional calculates one FFT in two dimensions.
- Three-dimensional calculates one FFT in three dimensions.

	1-dimensional (single)	1-dimensional (multiple)	2-dimensional	3-dimensional
C->C	CCFFT	CCFFTM	CCFFTMR	CCFFT2D CCFFT3D
Z->Z	ZZFFT	ZZFFTM	ZZFFTMR	ZZFFT2D ZZFFT3D
S->C	SCFFT	SCFFTM		SCFFT2D SCFFT3D
D->Z	DZFFT	DZFFTM		DZFFT2D DZFFT3D
C->S	CSFFT	CSFFTM		CSFFT2D CSFFT3D
Z->D	ZDFFT	ZDFFTM		ZDFFT2D ZDFFT3D

Implementation Notes: *work* and *table* arrays

The FFT routines were designed so that they can be implemented efficiently on many different architectures. The calling sequence is the same in any implementation. Certain details, however, depend on the particular implementation.

One area of difference is the size of the *table* and *work* arrays. Different systems may need different sizes. The subroutine call requires no change, but you may have to change array sizes in the `DIMENSION` or `type` statements that declare the arrays. The following are the required array sizes for the Origin and Altix series (n , n_1 , n_2 , and n_3 are transform sizes; the values of NF and NFR are explained below):

- CCFFT
 - table*: $2n + NF$ REAL
 - work*: $2n$ REAL WORDS

- ZZFFT
 - table*: $2n + NF$ DBL PREC WORDS
 - work*: $2n$ DBL PREC WORDS

- CCFFTMR
 - table*: $2n + NF$ REAL WORDS
 - work*: $2n$ REAL WORDS

- ZZFTMR
 - table*: $2n + NF$ DBL PREC WORDS
 - work*: $2n$ DBL PREC WORDS

- CCFFT2D
 - table*: $(2*n_1+NF) + (2*n_2+NF)$ REAL WORDS
 - work*: $2*MAX(n_1, n_2)$ REAL WORDS

- ZZFFT2D
 - table*: $(2*n_1+NF) + (2*n_2+NF)$ DBL PREC WORDS
 - work*: $2*MAX(n_1, n_2)$ DBL PREC WORDS

- CCFFT3D
 - table*: $(2*n_1+NF) + (2*n_2+NF) + (2*n_3+NF)$ REAL WORDS
 - work*: $2*MAX(n_1, n_2, n_3)$ REAL WORDS

- ZZFFT3D
table: $(2*n1+NF) + (2*n2+NF) + (2*n3+NF)$ DBL PREC WORDS
work: $2*MAX(n1, n2, n3)$ DBL PREC WORDS
- CCFFTM
table: $(NF + 2 * n)$ REAL
work: $2n$ REAL WORDS
- ZZFFTM
table: $(NF + 2 * n)$ DBL PREC
work: $2n$ DBL PREC WORDS
- SCFFT, CSFFT
table: $(n+NFR)$ REAL
work: $n+2$ REAL WORDS
- DZFFT, ZDFFT
table: $(n+NFR)$ DBL PREC
work: $n + 2$ DBL PREC WORDS
- SCFFT2D, CSFFT2D
table: $(n+NFR) + (2*n2+NF)$ REAL
work: $n1+4*n2$ REAL WORDS
- DZFFT2D, ZDFFT2D
table: $(n1+NFR) + (2*n2+NF)$ DBL PREC WORDS
work: $n1 + 4 * n2$ DBL PREC WORDS
- SCFFT3D, CSFFT3D
table: $(n1+NFR) + (2*n2+NF) + (2*n3+NF)$ REAL WORDS
work: $n1 + 4 * n3$ REAL WORDS
- DZFFT3D, ZDFFT3D
table: $(n1+NFR) + (2*n2+NF) + (2*n3+NF)$ DBL PREC WORDS
work: $n1 + 4 * n3$ DBL PREC WORDS

- SCFFTM, CSFFTM

table: (n+NFR) REAL WORDS

work: n + 2 REAL WORDS

- DZFFTM, ZDFFTM

table: (n+NFR) DBL PREC

work: n + 2 DBL PREC WORDS

Implementation Notes: *isys* Parameter Array

The second area of difference is the *isys* parameter array, an array that gives certain implementation-specific information. All features and functions of the FFT routines specific to any particular implementation are confined to this *isys* array. On any implementation, you can use the default values by using an argument value of 0.

In the Origin and Altix series implementations, $isys(0)=0$ and $isys(0)=1$ are supported. In SCSL versions prior to 1.3, only $isys(0)=0$ was allowed. For $isys(0)=0$, $NF=30$ and $NFR=15$, and for $isys(0)=1$, $NF=NFR=256$. The $NF(R)$ words of storage in the *table* array contain a factorization of the length of the transform.

The smaller values of NF and NFR for $isys(0)=0$ are historical. They are too small to store all the required factors for the highest performing FFT, so when $isys(0)=0$, extra space is allocated when the *table* array is initialized. To avoid memory leaks, this extra space must be deallocated when the *table* array is no longer needed. The routines CCFFTF, CCFFTMF, etc., are used to release this memory. Due to the potential for memory leaks, the use of $isys(0)=0$ should be avoided.

For $isys(0)=1$, the values of NF and NFR are large enough so that no extra memory needs to be allocated, and there is no need to call CCFFTF, etc. to release memory. (If called, these routines do nothing.) $isys(0) = 1$ means that *isys* is an integer array with two elements. The second element, $isys(1)$, will not be accessed.

Implementation Notes: Scratch Space

Finally, in addition to the *work* array, the FFT routines also dynamically allocate scratch space from the stack. The amount of space allocated can be slightly bigger than the size of the largest processor cache. For single processor runs, the default stack size is large enough that these allocations generally cause no problems. But for parallel runs, you need to ensure that the stack size of slave threads is big enough to hold this scratch space. Failure to reserve sufficient stack space will cause programs to dump core due to stack overflows. The stack size of MP library slave threads on

Origin systems is controlled via the `MP_SLAVE_STACKSIZE` environment variable or the `mp_set_slave_stacksize()` library routine. See the `mp(3c)`, `mp(3F)` and `pe_environ(5)` man pages for more information on controlling the slave stack size.

On Altix systems, the OpenMP thread stack size is controlled by the `KMP_STACKSIZE` environment variable or by the `kmp_set_stacksize_s()` library routine; see the Intel compiler documentation for additional details. SCSL versions 1.4.1, or later, will automatically attempt to increase the OpenMP thread stack size limit to a safe value if the library detects an initial setting that is too low. Because this action is performed at DSO initialization time, it is not possible for the library to detect subsequent `kmp_set_stacksize_s()` calls within user code that may be below the recommended threshold.

For pthreads applications, the thread's stack size is specified as one of many creation attributes provided in the `pthread_attr_t` argument to `pthread_create(3P)`. The `stacksize` attribute should be set explicitly to a non-default value using the `pthread_attr_setstacksize(3P)` call, described in the `pthread_attr_init(3P)` man page.

Convolution and Correlation Routines

The convolution routines feature convolution for Finite Impulse Response (FIR) filters, as well as, correlations. Each routine is highly optimized for single-processor use. The routines which use two-dimensional input sequences are multitasked (multi-threaded).

The convolution and correlation routines are very general. To achieve this generality and maximum flexibility, one-dimensional sequences are defined by 3 parameters. Six parameters are necessary for two-dimensional sequences. One drawback of this generality is the long subroutine argument list.

The following table contains a summary of the filter and correlation routines. In this table, rows of the table represent data types for the routines in each column:

- C implies 32-bit complex data.
- Z implies 64-bit double complex data
- S implies 32-bit real data.
- D implies 64-bit double precision real data.

Columns of the table represent the type of computation as well as the number of dimensions for which the convolution or correlation is calculated for the routines in each row:

- One-dimensional FIR applies a Finite Impulse Response filter to one-dimensional signals.
- One-dimensional (multiple) FIR applies a Finite Impulse Response filter to multiple one-dimensional signals.
- Two-dimensional FIR applies a Finite Impulse Response filter to two-dimensional signals.
- One-dimensional COR calculates the correlation of one-dimensional sequences.
- One-dimensional (multiple) COR calculates the correlation of multiple one-dimensional sequences.
- Two-dimensional COR calculates the correlation of two-dimensional sequences.

Type	1D (single)	1D (multiple)	2D
C	CFIR1D	CFIRM1D	CFIR2D
Z	ZFIR1D	ZFIRM1D	ZFIR2D
S	SFIR1D	SFIRM1D	SFIR2D
D	DFIR1D	DFIRM1D	DFIR2D
C	CCOR1D	CCORM1D	CCOR2D
Z	ZCOR1D	ZCORM1D	ZCOR2D
S	SCOR1D	SCORM1D	SCOR2D
D	DCOR1D	DCORM1D	DCOR2D

Supported SCSL Routines

This appendix lists all supported SCSL routines and a brief description of each.

For details, see the individual man pages.

Introductory Man Pages

The following man pages provide an introduction to the different types of routines supported in SCSL.

- `INTRO_BLAS1` - Introduction to vector-vector linear algebra subprograms
- `INTRO_BLAS2` - Introduction to matrix-vector linear algebra subprograms
- `INTRO_BLAS3` - Introduction to matrix-matrix linear algebra subprograms
- `INTRO_BLAS` - Introduction to SCSL Basic Linear Algebra Subprograms
- `INTRO_CBLAS` - Introduction to the C interface to Fortran 77 Basic Linear Algebra Subprograms (legacy BLAS)
- `INTRO_FFT` - Introduction to signal processing routines
- `INTRO_LAPACK` - Introduction to LAPACK solvers for dense linear systems
- `INTRO_SCSL` - Introduction to Scientific Computing Software Library (SCSL) routines
- `INTRO_SOLVERS` - Introduction to SGI-developed linear equation solvers

BLAS Routines

The following is a list of all BLAS 1, BLAS 2, and BLAS 3 supported routines.

- `CGEMM3M`, `ZGEMM3M` - Multiplies a complex general matrix by a complex general matrix
- `CHBMV`, `ZHBMV` - Multiplies a complex vector by a complex Hermitian band matrix

- CHEMM, ZHEMM - Multiplies a complex general matrix by a complex Hermitian matrix
- CHEMV, ZHEMV - Multiplies a complex vector by a complex Hermitian matrix
- CHER2, ZHER2 - Performs Hermitian rank 2 update of a complex Hermitian matrix
- CHER2K, ZHER2K - Performs Hermitian rank 2k update of a complex Hermitian matrix
- CHER, ZHER - Performs Hermitian rank 1 update of a complex Hermitian matrix
- CHERK, ZHERK - Performs Hermitian rank k update of a complex Hermitian matrix
- CHPMV, ZHPMV - Multiplies a complex vector by a packed complex Hermitian matrix
- CHPR2, ZHPR2 - Performs Hermitian rank 2 update of a packed complex Hermitian matrix
- CHPR, ZHPR - Performs Hermitian rank 1 update of a packed complex Hermitian matrix
- CSROT, ZDROT - applies a real plane rotation to a pair of complex vectors
- DGEMMS - Multiplies a real general matrix by a real general matrix, using Strassen's algorithm
- ISAMAX, IDAMAX, ICAMAX, IZAMAX - Searches a vector for the first occurrence of the maximum absolute value
- ISAMIN, IDAMIN - Searches a vector for the first occurrence of the minimum absolute value
- ISMAX, IDMAX - Searches a real vector for the first occurrence of the maximum value
- ISMIN, IDMIN - Searches a real vector for the first occurrence of the minimum value
- SASUM, DASUM, SCASUM, DZASUM - Sums the absolute value of elements in a real or complex vector
- SAXPBY, DAXPBY, CAXPBY, ZAXPBY - Adds a scalar multiple of a Single precision or complex vector x to a scalar multiple of another Single precision or complex vector y

- SAXPY, CAXPY, DAXPY, ZAXPY - Adds a scalar multiple of a real or complex vector to another real or complex vector
- SCOPY, DCOPY, CCOPY, ZCOPY - Copies a real or complex vector into another real or complex vector
- SDOT, DDOT, CDOTC, ZDOTC, CDOTU, ZDOTU - Computes a dot product (inner product) of two real or complex vectors
- SGBMV, DGBMV, CGBMV, ZGBMV - Multiplies a real or complex vector by a real or complex general band matrix
- SGEMM, DGEMM, CGEMM, ZGEMM - Multiplies a real or complex general matrix by a real or complex general matrix
- SGEMV, DGEMV, CGEMV, ZGEMV - Multiplies a real or complex vector by a real or complex general matrix
- SGER, DGER, CGERC, ZGERC, CGERU, ZGERU - Performs rank 1 update of a real or complex general matrix
- SGESUM, DGESUM, CGESUM, ZGESUM - Adds a scalar multiple of a real or complex matrix to a scalar multiple of another real or complex matrix
- SHAD, DHAD, CHAD, ZHAD - Computes the Hadamard product of two vectors
- SNRM2, DNRM2, SCNRM2, DZNRM2 - Computes the Euclidean norm of a vector
- SROT, DROT, CROT, ZROT - applies a real plane rotation or complex coordinate rotation
- SROTG, DROTG, CROTG, ZROTG - Constructs a Givens plane rotation
- SROTM, DROTM - applies a modified Givens plane rotation
- SROTMG, DROTMG - Constructs a modified Givens plane rotation
- SSBMV, DSBMV - Multiplies a real vector by a real symmetric band matrix
- SSCAL, DSCAL, CSSCAL, ZDSCAL, CSCAL, ZSCAL - Scales a real or complex vector
- SSPMV, DSPMV, CSPMV, ZSPMV - Multiplies a real or complex symmetric packed matrix by a real or complex vector
- SSPR2, DSPR2 - Performs symmetric rank 2 update of a real symmetric packed matrix

- SSPR, DSPR, CSPR, ZSPR - Performs symmetric rank 1 update of a real or complex symmetric packed matrix
- SSUM, DSUM, CSUM, ZSUM - Sums the elements of a real or complex vector
- SSWAP, DSWAP, CSWAP, ZSWAP - Swaps two real or complex vectors
- SSYM, DSYM, CSYM, ZSYM - Multiplies a real or complex general matrix by a real or complex symmetric matrix
- SSYMV, DSYMV, CSYMV, ZSYMV - Multiplies a real or complex vector by a real or complex symmetric matrix
- SSYR2, DSYR2 - Performs symmetric rank 2 update of a real symmetric matrix
- SSYR2K, DSYR2K, CSYR2K, ZSYR2K - Performs symmetric rank 2k update of a real or complex symmetric matrix
- SSYR, DSYR, CSYR, ZSYR - Performs symmetric rank 1 update of a real or complex symmetric matrix
- SSYRK, DSYRK, CSYRK, ZSYRK - Performs symmetric rank k update of a real or complex symmetric matrix
- STBMV, DTBMV, CTBMV, ZTBMV - Multiplies a real or complex vector by a real or complex triangular band matrix
- STBSV, DTBSV, CTBSV, ZTBSV - Solves a real or complex triangular banded system of equations
- STPMV, DTPMV, CTPMV, ZTPMV - Multiplies a real or complex vector by a real or complex triangular packed matrix
- STPSV, DTPSV, CTPSV, ZTPSV - Solves a real or complex triangular packed system of equations
- STRMM, DTRMM, CTRMM, ZTRMM - Multiplies a real or complex general matrix by a real or complex triangular matrix
- STRMV, DTRMV, CTRMV, ZTRMV - Multiplies a real or complex vector by a real or complex triangular matrix
- STRSM, DTRSM, CTRSM, ZTRSM - Solves a real or complex triangular system of equations with multiple right-hand sides

- STRSV, DTRSV, CTRSV, ZTRSV - Solves a real or complex triangular system of equations

FFT Routines

The following is a list of all supported Fast Fourier Transform (FFT) routines.

- CCFFT2D, ZFFT2D - applies a two-dimensional complex-to-complex Fast Fourier Transform (FFT)
- CCFFT3D, ZFFT3D - applies a three-dimensional complex-to-complex Fast Fourier Transform (FFT)
- CCFFT, ZFFT - applies a complex-to-complex Fast Fourier Transform (FFT)
- CCFFTF, CCFFTMF, CCFFTMRF, CCFFT2DF, CCFFT3DF, ZFFTF, ZFFTMF, ZFFTMRF, ZFFT2DF, ZFFT3DF - deallocates memory tacked on to the table array during initialization
- CCFFTM, ZFFTM - applies multiple complex-to-complex Fast Fourier Transforms (FFTs)
- CCFFTMR, ZFFTMR - applies multiple complex-to-complex Fast Fourier Transforms (FFTs) to the rows of a two-dimensional (2D) array
- CCOR1D, ZCOR1D, SCOR1D, DCOR1D - computes the one-dimensional (1D) correlation of two sequences.
- CCOR2D, ZCOR2D, SCOR2D, DCOR2D - computes the two-dimensional (2D) correlation of two two-dimensional (2D) arrays
- CCORM1D, ZCORM1D, SCORM1D, DCORM1D - computes multiple 1D correlations
- CFIR1D, ZFIR1D, SFIR1D, DFIR1D - computes the 1D convolution of a sequence
- CFIR2D, ZFIR2D, SFIR2D, DFIR2D - computes the two-dimensional (2D) convolution of two 2D arrays
- CFIRM1D, ZFIRM1D, SFIRM1D, DFIRM1D - computes multiple 1D convolutions
- SCFFT2D, DZFFT2D, CSFFT2D, ZDFFT2D - applies a two-dimensional real-to-complex or complex-to-real Fast Fourier Transform (FFT)

- SCFFT3D, DZFFT3D, CSFFT3D, ZDFFT3D - applies a three-dimensional real-to-complex Fast Fourier Transform (FFT)
- SCFFT, DZFFT, CSFFT, ZDFFT - computes a real-to-complex or complex-to-real Fast Fourier Transform (FFT)
- SCFFTF, SCFFTMF, SCFFT2DF, SCFFT3DF, DZFFTF, DZFFTMF, DZFFT2DF, DZFFT3DF - Deallocate memory tacked on to the table array during initialization
- SCFFTM, DZFFTM, CSFFTM, ZDFFTM - applies multiple real-to-complex or complex-to-real Fast Fourier Transforms (FFTs)

LAPACK Routines

The following is a list of all supported LAPACK routines.

- CBDSQR - computes the singular value decomposition (SVD) of a real N-by-N (upper or lower) bidiagonal matrix B
- CGBBRD - reduces a complex general m-by-n band matrix A to real upper bidiagonal form B by a unitary transformation
- CGBCON - estimates the reciprocal of the condition number of a complex general band matrix A
- CGBEQU - computes row and column scalings intended to equilibrate an M-by-N band matrix A and reduce its condition number
- CGBRFS - improves the computed solution to a system of linear equations when the coefficient matrix is banded
- CGBSV - computes the solution to a complex system of linear equations
- CGBSVX - uses the LU factorization to compute the solution to a complex system of linear equations
- CGBTF2 - computes an LU factorization of a complex m-by-n band matrix A using partial pivoting with row interchanges
- CGBTRF - computes an LU factorization of a complex m-by-n band matrix A using partial pivoting with row interchanges
- CGBTRS - solves a system of linear equations with a general band matrix A using the LU factorization computed by CGBTRF

- CGEBAK - forms the right or left eigenvectors of a complex general matrix by backward transformation on the computed eigenvectors of the balanced matrix output by CGEBAL
- CGEBAL - balances a general complex matrix A
- CGEBD2 - reduces a complex general m by n matrix A to upper or lower real bidiagonal form B by a unitary transformation
- CGEBRD - reduces a general complex M-by-N matrix A to upper or lower bidiagonal form B by a unitary transformation
- CGECON - estimates the reciprocal of the condition number of a general complex matrix A using the LU factorization computed by CGETRF
- CGEEQU - computes row and column scalings intended to equilibrate an M-by-N matrix A and reduce its condition number
- CGEES - computes the eigenvalues, the Schur form T, and, optionally, the matrix of Schur vectors Z
- CGEESX - computes the eigenvalues, the Schur form T, and, optionally, the matrix of Schur vectors Z
- CGEEV - computes the eigenvalues and, optionally, the left and/or right eigenvectors
- CGEEVX - computes the eigenvalues and, optionally, the left and/or right eigenvectors
- CGEGS - routine is deprecated and has been replaced by routine CGGES
- CGEGV - routine is deprecated and has been replaced by routine CGGEV
- CGEHD2 - reduces a complex general matrix A to upper Hessenberg form H by a unitary similarity transformation
- CGEHRD - reduces a complex general matrix A to upper Hessenberg form H by a unitary similarity transformation
- CGELQ2 - computes an LQ factorization of a complex m by n matrix A
- CGELQF - computes an LQ factorization of a complex M-by-N matrix A
- CGELS - solves overdetermined or underdetermined complex linear systems

- CGELSD - computes the minimum-norm solution to a real linear least squares problem
- CGELSS - computes the minimum norm solution to a complex linear least squares problem
- CGELSX - routine is deprecated and has been replaced by routine CGELSY
- CGELSY - computes the minimum-norm solution to a complex linear least squares problem
- CGEQL2 - computes a QL factorization of a complex m by n matrix A
- CGEQLF - computes a QL factorization of a complex M-by-N matrix A
- CGEQP3 - computes a QR factorization with column pivoting of a matrix A
- CGEQPF - routine is deprecated and has been replaced by routine CGEQP3
- CGEQR2 - computes a QR factorization of a complex m by n matrix A
- CGEQRF - computes a QR factorization of a complex M-by-N matrix A
- CGERFS - improves the computed solution to a system of linear equations
- CGERQ2 - computes an RQ factorization of a complex m by n matrix A
- CGERQF - computes an RQ factorization of a complex M-by-N matrix A
- CGESC2 - solves a system of linear equations with a general N-by-N matrix A using the LU factorization with complete pivoting computed by CGETC2
- CGESDD - computes the singular value decomposition (SVD) of a complex M-by-N matrix A
- CGESV - computes the solution to a complex system of linear equations
- CGESVD - computes the singular value decomposition (SVD) of a complex M-by-N matrix A, optionally computing the left and/or right singular vectors
- CGESVX - uses the LU factorization to compute the solution to a complex system of linear equations
- CGETC2 - computes an LU factorization, using complete pivoting, of the n-by-n matrix A

- CGETF2 - computes an LU factorization of a general m-by-n matrix A using partial pivoting with row interchanges
- CGETRF - computes an LU factorization of a general M-by-N matrix A using partial pivoting with row interchanges
- CGETRI - computes the inverse of a matrix using the LU factorization computed by CGETRF
- CGETRS - solves a system of linear equations with a general N-by-N matrix A using the LU factorization computed by CGETRF
- CGGBAK - forms the right or left eigenvectors of a complex generalized eigenvalue problem by backward transformation on the computed eigenvectors of the balanced pair of matrices output by CGGBAL
- CGGBAL - balances a pair of general complex matrices (A,B)
- CGGES - computes the generalized eigenvalues, the generalized complex Schur form (S, T), and optionally left and/or right Schur vectors (VSL and VSR)
- CGGESX - computes the generalized eigenvalues, the complex Schur form (S,T),
- CGGEV - computes the generalized eigenvalues, and optionally, the left and/or right generalized eigenvectors
- CGGEVX - computes the generalized eigenvalues, and optionally, the left and/or right generalized eigenvectors
- CGGGLM - solves a general Gauss-Markov linear model (GLM) problem
- CGGHRD - reduces a pair of complex matrices (A,B) to generalized upper Hessenberg form using unitary transformations, where A is a general matrix and B is upper triangular
- CGGLSE - solves the linear equality-constrained least squares (LSE) problem
- CGGQRF - computes a generalized QR factorization of an N-by-M matrix A and an N-by-P matrix B
- CGGRQF - computes a generalized RQ factorization of an M-by-N matrix A and a P-by-N matrix B
- CGGSVD - computes the generalized singular value decomposition (GSVD) of an M-by-N complex matrix A and P-by-N complex matrix B

- CCGSVP - computes unitary matrices
- CGTCON - estimates the reciprocal of the condition number of a complex tridiagonal matrix A using the LU factorization as computed by CGTTRF
- CGTRFS - improves the computed solution to a system of linear equations when the coefficient matrix is tridiagonal
- CGTSV - solves the equation $AX = B$,
- CGTSVX - uses the LU factorization to compute the solution to a complex system of linear equations
- CGTTRF - computes an LU factorization of a complex tridiagonal matrix A using elimination with partial pivoting and row interchanges
- CGTTRS - solves systems of equations
- CGTTS2 - solves systems of equations
- CHBEV - computes all the eigenvalues and, optionally, eigenvectors of a complex Hermitian band matrix A
- CHBEVD - computes all the eigenvalues and, optionally, eigenvectors of a complex Hermitian band matrix A
- CHBEVX - computes selected eigenvalues and, optionally, eigenvectors of a complex Hermitian band matrix A
- CHBGST - reduces a complex Hermitian-definite banded generalized eigenproblem
- CHBGV - computes all the eigenvalues and optionally, the eigenvectors of a complex generalized Hermitian-definite banded eigenproblem
- CHBGVD - computes all the eigenvalues, and optionally, the eigenvectors of a complex generalized Hermitian-definite banded eigenproblem
- CHBGVX - computes all the eigenvalues, and optionally, the eigenvectors of a complex generalized Hermitian-definite banded eigenproblem
- CHBTRD - reduces a complex Hermitian band matrix A to real symmetric tridiagonal form T by a unitary similarity transformation
- CHECON - estimates the reciprocal of the condition number of a complex Hermitian matrix A

- CHEEV - computes all eigenvalues and, optionally, eigenvectors of a complex Hermitian matrix A
- CHEEVD - computes all eigenvalues and, optionally, eigenvectors of a complex Hermitian matrix A
- CHEEVR - computes selected eigenvalues and, optionally, eigenvectors of a complex Hermitian matrix T
- CHEEVX - computes selected eigenvalues and, optionally, eigenvectors of a complex Hermitian matrix A
- CHEGS2 - reduces a complex Hermitian-definite generalized eigenproblem to standard form
- CHEGST - reduces a complex Hermitian-definite generalized eigenproblem to standard form
- CHEGV - computes all the eigenvalues, and optionally, the eigenvectors of a complex generalized Hermitian-definite eigenproblem
- CHEGVD - computes all the eigenvalues, and optionally, the eigenvectors of a complex generalized Hermitian-definite eigenproblem
- CHEGVX - computes selected eigenvalues, and optionally, eigenvectors of a complex generalized Hermitian-definite eigenproblem
- CHERFS - improves the computed solution to a system of linear equations when the coefficient matrix is Hermitian indefinite
- CHESV - computes the solution to a complex system of linear equations
- CHESVX - uses the diagonal pivoting factorization to compute the solution to a complex system of linear equations
- CHETD2 - reduces a complex Hermitian matrix A to real symmetric tridiagonal form T by a unitary similarity transformation
- CHETF2 - computes the factorization of a complex Hermitian matrix A using the Bunch-Kaufman diagonal pivoting method
- CHETRD - reduces a complex Hermitian matrix A to real symmetric tridiagonal form T by a unitary similarity transformation
- CHETRF - computes the factorization of a complex Hermitian matrix A using the Bunch-Kaufman diagonal pivoting method

- CHETRI - computes the inverse of a complex Hermitian indefinite matrix A using the factorization computed by CHETRF
- CHETRS - solves a system of linear equations with a complex Hermitian matrix A using the factorization computed by CHETRF
- CHGEQZ - implements a single-shift version of the QZ method for finding the generalized eigenvalues
- CHPCON - estimates the reciprocal of the condition number of a complex Hermitian packed matrix A using the factorization computed by CHPTRF
- CHPEV - computes all the eigenvalues and, optionally, eigenvectors of a complex Hermitian matrix in packed storage
- CHPEVD - computes all the eigenvalues and, optionally, eigenvectors of a complex Hermitian matrix A in packed storage
- CHPEVX - computes selected eigenvalues and, optionally, eigenvectors of a complex Hermitian matrix A in packed storage
- CHPGST - reduces a complex Hermitian-definite generalized eigenproblem to standard form, using packed storage
- CHPGV - computes all the eigenvalues and, optionally, the eigenvectors of a complex generalized Hermitian-definite eigenproblem
- CHPGVD - computes all the eigenvalues and, optionally, the eigenvectors of a complex generalized Hermitian-definite eigenproblem
- CHPGVX - computes selected eigenvalues and, optionally, eigenvectors of a complex generalized Hermitian-definite eigenproblem
- CHPRFS - improves the computed solution to a system of linear equations when the coefficient matrix is Hermitian indefinite and packed
- CHPSV - computes the solution to a complex system of linear equations
- CHPSVX - uses diagonal pivoting factorization to compute the solution to a complex system of linear equations
- CHPTRD - reduces a complex Hermitian matrix A stored in packed form to real symmetric tridiagonal form T by a unitary similarity transformation
- CHPTRF - computes the factorization of a complex Hermitian packed matrix A using the Bunch-Kaufman diagonal pivoting method

- CHPTRI - computes the inverse of a complex Hermitian indefinite matrix A in packed storage using the factorization computed by CHPTRF
- CHPTRS - solves a system of linear equations with a complex Hermitian matrix A stored in packed format using the factorization computed by CHPTRF
- CHSEIN - uses inverse iteration to find specified right and/or left eigenvectors of a complex upper Hessenberg matrix H
- CHSEQR - computes the eigenvalues of a complex upper Hessenberg matrix H, and, optionally, the matrices T and Z from the Schur decomposition
- CLABRD - reduces the first NB rows and columns of a complex general m by n matrix A to upper or lower real bidiagonal form
- CLACGV - conjugates a complex vector of length N
- CLACON - estimates the 1-norm of a square, complex matrix A
- CLACP2 - copies all or part of a real two-dimensional matrix A to a complex matrix B
- CLACPY - copies all or part of a two-dimensional matrix A to another matrix B
- CLACRM - performs a very simple matrix-matrix multiplication
- CLACRT - perform the operation $(c\ s)(x) \gg (x)(-s\ c)(y)(y)$ where c and s are complex and the vectors x and y are complex
- CLADIV - := X / Y, where X and Y are complex
- CLAE0 - computes all eigenvalues of a symmetric tridiagonal matrix which is one diagonal block
- CLAE7 - computes the updated eigensystem of a diagonal matrix after modification by a rank-one symmetric matrix
- CLAE8 - merges the two sets of eigenvalues together into a single sorted set
- CLAEIN - uses inverse iteration to find a right or left eigenvector corresponding to the eigenvalue W of a complex upper Hessenberg matrix H
- CLAESY - computes the eigendecomposition of a 2-by-2 symmetric matrix
- CLAEV2 - computes the eigendecomposition of a 2-by-2 Hermitian matrix
- CLAGS2 - computes 2-by-2 unitary matrices U, V and Q

- CLAGTM - performs a matrix-vector product
- CLAHF - computes a partial factorization of a complex Hermitian matrix A using the Bunch-Kaufman diagonal pivoting method
- CLAHQR - an auxiliary routine called by CHSEQR to update the eigenvalues and Schur decomposition already computed by CHSEQR
- CLAHRD - reduces the first NB columns of a complex general matrix so that elements below the k-th subdiagonal are zero
- CLAI1 - applies one step of incremental condition estimation in its simplest version
- CLALS0 - applies back the multiplying factors of either the left or the right singular vector matrix of a diagonal matrix
- CLALSA - an intermediate step in solving the least squares problem by computing the SVD of the coefficient matrix in compact form
- CLALSD - uses the singular value decomposition of A to solve the least squares problem of finding X to minimize the Euclidean norm of each column of AX-B
- CLANBB - returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of an n by n band matrix A
- CLANBE - returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a complex matrix A
- CLANBT - returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a complex tridiagonal matrix A
- CLANHB - returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of an n by n hermitian band matrix A, with k super-diagonals
- CLANHE - returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a complex hermitian matrix A
- CLANHP - returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a complex hermitian matrix A, supplied in packed form
- CLANHS - returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a Hessenberg matrix A

- CLANHT - returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a complex Hermitian tridiagonal matrix A
- CLANSB - returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of an n by n symmetric band matrix A, with k super-diagonals
- CLANSP - returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a complex symmetric matrix A, supplied in packed form
- CLANSY - returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a complex symmetric matrix A
- CLANTB - returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of an n by n triangular band matrix A, with (k + 1) diagonals
- CLANTP - returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a triangular matrix A, supplied in packed form
- CLANTR - returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a trapezoidal or triangular matrix A
- CLAPLL - computes the QR factorization of $A=QR$
- CLAPMT - rearranges the columns of the M by N matrix X
- CLAQGB - equilibrates a general M by N band matrix A
- CLAQGE - equilibrates a general M by N matrix A using the row and scaling factors in the vectors R and C
- CLAQHB - equilibrates a symmetric band matrix A using the scaling factors in the vector S
- CLAQHE - equilibrates a Hermitian matrix A using the scaling factors in the vector S
- CLAQHP - equilibrates a Hermitian matrix A using the scaling factors in the vector S
- CLAQP2 - computes a QR factorization with column pivoting

- **CLAQPS** - computes a step of QR factorization with column pivoting of a complex M-by-N matrix A
- **CLAQSB** - equilibrates a symmetric band matrix A using the scaling factors in the vector S
- **CLAQSP** - equilibrates a symmetric matrix A using the scaling factors in the vector S
- **CLAQSY** - equilibrates a symmetric matrix A using the scaling factors in the vector S
- **CLAR1V** - computes the (scaled) r^{th} column of the inverse of the submatrix in rows B1 through BN of a tridiagonal matrix
- **CLAR2V** - applies a vector of complex plane rotations with real cosines from both sides to a sequence of 2-by-2 complex Hermitian matrices,
- **CLARCM** - performs a very simple matrix-matrix multiplication
- **CLARF** - applies a complex elementary reflector H to a complex M-by-N matrix C, from either the left or the right
- **CLARFB** - applies a complex block reflector H or its transpose H' to a complex M-by-N matrix C, from either the left or the right
- **CLARFG** - generates a complex elementary reflector H of order n
- **CLARFT** - forms the triangular factor T of a complex block reflector H of order n, which is defined as a product of k elementary reflectors
- **CLARFX** - applies a complex elementary reflector H to a complex m by n matrix C, from either the left or the right
- **CLARGV** - generates a vector of complex plane rotations with real cosines, determined by elements of the complex vectors x and y
- **CLARNV** - returns a vector of n random complex numbers from a uniform or normal distribution
- **CLARRV** - computes the eigenvectors of a tridiagonal matrix
- **CLARTG** - generates a plane rotation
- **CLARTV** - applies a vector of complex plane rotations with real cosines to elements of the complex vectors x and y

- CLARZ - applies a complex elementary reflector H to a complex M-by-N matrix C, from either the left or the right
- CLARZB - applies a complex block reflector H or its transpose to a complex distributed M-by-N C from the left or the right
- CLARZT - forms the triangular factor T of a complex block reflector H
- CLASCL - multiplies the M by N complex matrix A by the real scalar CTO/CFROM
- CLASET - initializes a 2-D array A to BETA on the diagonal and ALPHA on the offdiagonals
- CLASR - performs a transformation $A := PA$
- CLASSQ - returns the values scl and ssq
- CLASWP - performs a series of row interchanges on the matrix A
- CLASYF - computes a partial factorization of a complex symmetric matrix A using the Bunch-Kaufman diagonal pivoting method
- CLATBS - solves a triangular system
- CLATDF - computes the contribution to the reciprocal Dif-estimate
- CLATPS - solves a triangular system
- CLATRD - reduces NB rows and columns of a complex Hermitian matrix A
- CLATRS - solves a triangular system
- CLATRZ - factors a M-by-(M+L) complex upper trapezoidal matrix
- CLATZM - routine is deprecated and has been replaced by routine CUNMRZ
- CLAUU2 - computes the product $U \times U'$ or $L' \times L$
- CLAUUM - computes the product $U \times U'$ or $L' \times L$, where the triangular factor U or L is stored in the upper or lower triangular part of the array A
- CPBCON - estimates the reciprocal of the condition number (in the 1-norm) of a complex Hermitian positive definite band matrix using the Cholesky factorization computed by CPBTRF

- CPBEQU - computes row and column scalings intended to equilibrate a Hermitian positive definite band matrix A and reduce its condition number (with respect to the two-norm)
- CPBRFS - improves the computed solution to a system of linear equations when the coefficient matrix is Hermitian positive definite and banded
- CPBSTF - computes a split Cholesky factorization of a complex Hermitian positive definite band matrix A
- CPBSV - computes the solution to a complex system of linear equations
- CPBSVX - uses the Cholesky factorization to compute the solution to a complex system of linear equations
- CPBTF2 - computes the Cholesky factorization of a complex Hermitian positive definite band matrix A
- CPBTRF - computes the Cholesky factorization of a complex Hermitian positive definite band matrix A
- CPBTRS - solves a system of linear equations with a Hermitian positive definite band matrix A
- CPOCON - estimates the reciprocal of the condition number (in the 1-norm) of a complex Hermitian positive definite matrix
- CPOEQU - computes row and column scalings intended to equilibrate a Hermitian positive definite matrix A and reduce its condition number (with respect to the two-norm)
- CPORFS - improves the computed solution to a system of linear equations when the coefficient matrix is Hermitian positive definite
- CPOSV - computes the solution to a complex system of linear equations
- CPOSVX - uses the Cholesky factorization to compute the solution to a complex system of linear equations
- CPOTF2 - computes the Cholesky factorization of a complex Hermitian positive definite matrix A
- CPOTRF - computes the Cholesky factorization of a complex Hermitian positive definite matrix A
- CPOTRI - computes the inverse of a complex Hermitian positive definite matrix A

- CPOTRS - solves a system of linear equations with a Hermitian positive definite matrix A
- CPPCON - estimates the reciprocal of the condition number (in the 1-norm) of a complex Hermitian positive definite packed matrix
- CPPEQU - computes row and column scalings intended to equilibrate a Hermitian positive definite matrix A in packed storage and reduce its condition number (with respect to the two-norm)
- CPPRFS - improves the computed solution to a system of linear equations when the coefficient matrix is Hermitian positive definite and packed, and provides error bounds and backward error estimates for the solution
- CPPSV - computes the solution to a complex system of linear equations
- CPPSVX - uses the Cholesky factorization to compute the solution to a complex system of linear equations
- CPPTRF - computes the Cholesky factorization of a complex Hermitian positive definite matrix A stored in packed format
- CPPTRI - computes the inverse of a complex Hermitian positive definite matrix A using the Cholesky factorization computed by CPPTRF
- CPPTRS - solves a system of linear equations with a Hermitian positive definite matrix A in packed storage using the Cholesky factorization computed by CPPTRF
- CPTCON - computes the reciprocal of the condition number (in the 1-norm) of a complex Hermitian positive definite tridiagonal matrix using the factorization computed by CPTTRF
- CPTEQR - computes all eigenvalues and, optionally, eigenvectors of a symmetric positive definite tridiagonal matrix
- CPTRFS - improves the computed solution to a system of linear equations when the coefficient matrix is Hermitian positive definite and tridiagonal
- CPTSV - computes the solution to a complex system of linear equations
- CPTSVX - computes the solution to a complex system of linear equations
- CPTTRF - computes the factorization of a complex Hermitian positive definite tridiagonal matrix A
- CPTTRS - solves a tridiagonal system using the factorization computed by CPTTRF

- CPTTS2 - solves a tridiagonal system using the factorization computed by CPTTRF
- CSPCON - estimates the reciprocal of the condition number (in the 1-norm) of a complex symmetric packed matrix A
- CSPRFS - improves the computed solution to a system of linear equations when the coefficient matrix is symmetric indefinite and packed
- CSPSV - computes the solution to a complex system of linear equations
- CSPSVX - uses diagonal pivoting factorization to compute the solution to a complex system of linear equations
- CSPTRF - computes the factorization of a complex symmetric matrix A stored in packed format using the Bunch-Kaufman diagonal pivoting method
- CSPTRI - computes the inverse of a complex symmetric indefinite matrix A in packed storage using the factorization computed by CSPTRF
- CSPTRS - solves a system of linear equations with a complex symmetric matrix A stored in packed format using the factorization computed by CSPTRF
- CSRSCCL - multiplies an n-element complex vector x by the real scalar 1/a
- CSTEDC - computes all eigenvalues and, optionally, eigenvectors of a symmetric tridiagonal matrix using the divide and conquer method
- CSTEGR - computes selected eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix T
- CSTEIN - computes the eigenvectors of a real symmetric tridiagonal matrix T corresponding to specified eigenvalues, using inverse iteration
- CSTEQR - computes all eigenvalues and, optionally, eigenvectors of a symmetric tridiagonal matrix using the implicit QL or QR method
- CSYCON - estimates the reciprocal of the condition number (in the 1-norm) of a complex symmetric matrix A using the factorization computed by CSYTRF
- CSYRFS - improves the computed solution to a system of linear equations when the coefficient matrix is symmetric indefinite, and provides error bounds and backward error estimates for the solution
- CSYSV - computes the solution to a complex system of linear equations

- CSYSVX - uses the diagonal pivoting factorization to compute the solution to a complex system of linear equations
- CSYTF2 - computes the factorization of a complex symmetric matrix A using the Bunch-Kaufman diagonal pivoting method
- CSYTRF - computes the factorization of a complex symmetric matrix A using the Bunch-Kaufman diagonal pivoting method
- CSYTRI - computes the inverse of a complex symmetric indefinite matrix A using the factorization computed by CSYTRF
- CSYTRS - solves a system of linear equations with a complex symmetric matrix A using the factorization computed by CSYTRF
- CTBCON - estimates the reciprocal of the condition number of a triangular band matrix A, in either the 1-norm or the infinity-norm
- CTBRFS - provides error bounds and backward error estimates for the solution to a system of linear equations with a triangular band coefficient matrix
- CTBTRS - solves a triangular system
- CTGEVC - computes some or all of the right and/or left generalized eigenvectors of a pair of complex upper triangular matrices (A,B)
- CTGEX2 - swaps adjacent diagonal 1 by 1 blocks (A11,B11) and (A22,B22)
- CTGEXC - reorders the generalized Schur decomposition of a complex matrix pair (A,B), using a unitary equivalence transformation
- CTGSEN - reorders the generalized Schur decomposition of a complex matrix pair (A, B)
- CTGSJA - computes the generalized singular value decomposition (GSVD) of two complex upper triangular (or trapezoidal) matrices A and B
- CTGSNA - estimates reciprocal condition numbers for specified eigenvalues and/or eigenvectors of a matrix pair (A, B)
- CTGSY2 - solves the generalized Sylvester equation using Level 1 and 2 BLAS
- CTGSYL - solves the generalized Sylvester equation
- CTPCON - estimates the reciprocal of the condition number of a packed triangular matrix A, in either the 1-norm or the infinity-norm

- CTPRFS - provides error bounds and backward error estimates for the solution to a system of linear equations with a triangular packed coefficient matrix
- CTPTRI - computes the inverse of a complex upper or lower triangular matrix A stored in packed format
- CTPTRS - solves a triangular system
- CTRCON - estimates the reciprocal of the condition number of a triangular matrix A, in either the 1-norm or the infinity-norm
- CTREVC - computes some or all of the right and/or left eigenvectors of a complex upper triangular matrix T
- CTREXC - reorders the Schur factorization of a complex matrix so that the diagonal element of T with row index IFST is moved to row ILST
- CTRID - computes the solution to a complex system of linear equations
- CTRRFS - provides error bounds and backward error estimates for the solution to a system of linear equations with a triangular coefficient matrix
- CTRSEN - reorders the Schur factorization of a complex matrix
- CTRSNA - estimates reciprocal condition numbers for specified eigenvalues and/or right eigenvectors of a complex upper triangular matrix T
- CTRSYL - solves the complex Sylvester matrix equation
- CTRTI2 - computes the inverse of a complex upper or lower triangular matrix
- CTRTRI - computes the inverse of a complex upper or lower triangular matrix A
- CTRTRS - solves a triangular system
- CTZRQF - routine is deprecated and has been replaced by routine CTZRZF
- CTZRZF - reduces the M-by-N ($M \leq N$) complex upper trapezoidal matrix A to upper triangular form by means of unitary transformations
- CUNG2L - generates an m by n complex matrix Q with orthonormal columns,
- CUNG2R - generates an m by n complex matrix Q with orthonormal columns,
- CUNGBR - generates one of the complex unitary matrices Q or P^H determined by CGEBRD when reducing a complex matrix A to bidiagonal form

- CUNGHR - generates a complex unitary matrix Q which is defined as the product of IHI-ILO elementary reflectors of order N , as returned by CGEHRD
- CUNGL2 - generates an m -by- n complex matrix Q with orthonormal rows,
- CUNGLQ - generates an M -by- N complex matrix Q with orthonormal rows,
- CUNGQL - generates an M -by- N complex matrix Q with orthonormal columns,
- CUNGQR - generates an M -by- N complex matrix Q with orthonormal columns,
- CUNGR2 - generates an m by n complex matrix Q with orthonormal rows,
- CUNGRQ - generates an M -by- N complex matrix Q with orthonormal rows,
- CUNGTR - generates a complex unitary matrix Q which is defined as the product of $n-1$ elementary reflectors of order N , as returned by CHETRD
- CUNM2L - overwrites the general complex m -by- n matrix C
- CUNM2R - overwrites the general complex m -by- n matrix C
- CUNMBR - overwrites the general complex M -by- N matrix C
- CUNMHR - overwrites the general complex M -by- N matrix C
- CUNML2 - overwrites the general complex m -by- n matrix C
- CUNMLQ - overwrites the general complex M -by- N matrix C
- CUNMQL - overwrites the general complex M -by- N matrix C
- CUNMQR - overwrites the general complex M -by- N matrix C
- CUNMR2 - overwrites the general complex m -by- n matrix C
- CUNMR3 - overwrites the general complex m by n matrix C
- CUNMRQ - overwrites the general complex M -by- N matrix C
- CUNMRZ - overwrites the general complex M -by- N matrix C
- CUNMTR - overwrites the general complex M -by- N matrix C
- CUPGTR - generates a complex unitary matrix Q
- CUPMTR - overwrites the general complex M -by- N matrix C

- DBDSDC - computes the singular value decomposition (SVD) of a real N-by-N (upper or lower) bidiagonal matrix B
- DBDSQR - computes the singular value decomposition (SVD) of a real N-by-N (upper or lower) bidiagonal matrix B
- DDISNA - computes the reciprocal condition numbers for the eigenvectors of a real symmetric or complex Hermitian matrix or for the left or right singular vectors of a general m-by-n matrix
- DGBBRD - reduces a real general m-by-n band matrix A to upper bidiagonal form B by an orthogonal transformation
- DGBCON - estimates the reciprocal of the condition number of a real general band matrix A
- DGBEQU - computes row and column scalings intended to equilibrate an M-by-N band matrix A and reduce its condition number
- DGBRFS - improves the computed solution to a system of linear equations when the coefficient matrix is banded
- DGBSV - computes the solution to a real system of linear equations
- DGBSVX - uses the LU factorization to compute the solution to a real system of linear equations
- DGBTF2 - computes an LU factorization of a real m-by-n band matrix A using partial pivoting with row interchanges
- DGBTRF - computes an LU factorization of a real m-by-n band matrix A using partial pivoting with row interchanges
- DGBTRS - solves a system of linear equations with a general band matrix A using the LU factorization computed by DGBTRF
- DGEBAK - forms the right or left eigenvectors of a real general matrix by backward transformation on the computed eigenvectors of the balanced matrix output by DGEBAL
- DGEBAL - balances a general real matrix A
- DGEBD2 - reduces a real general m by n matrix A to upper or lower bidiagonal form B by an orthogonal transformation

- DGEBRD - reduces a general real M-by-N matrix A to upper or lower bidiagonal form B by an orthogonal transformation
- DGECON - estimates the reciprocal of the condition number of a general real matrix A, in either the 1-norm or the infinity-norm, using the LU factorization computed by DGETRF
- DGEEQU - computes row and column scalings intended to equilibrate an M-by-N matrix A and reduce its condition number
- DGEES - computes for an N-by-N real nonsymmetric matrix A, the eigenvalues, the real Schur form T, and, optionally, the matrix of Schur vectors Z
- DGEESX - computes for an N-by-N real nonsymmetric matrix A, the eigenvalues, the real Schur form T, and, optionally, the matrix of Schur vectors Z
- DGEEV - computes for an N-by-N real nonsymmetric matrix A, the eigenvalues and, optionally, the left and/or right eigenvectors
- DGEEVX - computes for an N-by-N real nonsymmetric matrix A, the eigenvalues and, optionally, the left and/or right eigenvectors
- DGEES - routine is deprecated and has been replaced by routine DGGES
- DGEESV - routine is deprecated and has been replaced by routine DGGEV
- DGEHD2 - reduces a real general matrix A to upper Hessenberg form H by an orthogonal similarity transformation
- DGEHRD - reduces a real general matrix A to upper Hessenberg form H by an orthogonal similarity transformation
- DGELQ2 - computes an LQ factorization of a real m by n matrix A
- DGELQF - computes an LQ factorization of a real M-by-N matrix A
- DGELS - solves overdetermined or underdetermined real linear systems involving an M-by-N matrix A, or its transpose, using a QR or LQ factorization of A
- DGELSD - computes the minimum-norm solution to a real linear least squares problem
- DGELSS - computes the minimum norm solution to a real linear least squares problem
- DGELSX - routine is deprecated and has been replaced by routine DGELSY

- DGELSY - computes the minimum-norm solution to a real linear least squares problem
- DGEQL2 - computes a QL factorization of a real m by n matrix A
- DGEQLF - computes a QL factorization of a real M-by-N matrix A
- DGEQP3 - computes a QR factorization with column pivoting of a matrix A
- DGEQPF - routine is deprecated and has been replaced by routine DGEQP3
- DGEQR2 - computes a QR factorization of a real m by n matrix A
- DGEQRF - computes a QR factorization of a real M-by-N matrix A
- DGERFS - improves the computed solution to a system of linear equations and provides error bounds and backward error estimates for the solution
- DGERQ2 - computes an RQ factorization of a real m by n matrix A
- DGERQF - computes an RQ factorization of a real M-by-N matrix A
- DGES2 - solves a system of linear equations with a general N-by-N matrix A using the LU factorization with complete pivoting computed by DGETC2
- DGESDD - computes the singular value decomposition (SVD) of a real M-by-N matrix A
- DGESV - computes the solution to a real system of linear equations
- DGESVD - computes the singular value decomposition (SVD) of a real M-by-N matrix A
- DGESVX - uses the LU factorization to compute the solution to a real system of linear equations
- DGETC2 - computes an LU factorization with complete pivoting of the n-by-n matrix A
- DGETF2 - computes an LU factorization of a general m-by-n matrix A using partial pivoting with row interchanges
- DGETRF - computes an LU factorization of a general M-by-N matrix A using partial pivoting with row interchanges
- DGETRI - computes the inverse of a matrix using the LU factorization computed by DGETRF

- DGETRS - solves a system of linear equations with a general N-by-N matrix A using the LU factorization computed by DGETRF
- DGGBAK - forms the right or left eigenvectors of a real generalized eigenvalue problem by backward transformation on the computed eigenvectors of the balanced pair of matrices output by DGGBAL
- DGGBAL - balances a pair of general real matrices (A,B)
- DGGES - computes for a pair of N-by-N real nonsymmetric matrices (A,B),
- DGGESX - computes for a pair of N-by-N real nonsymmetric matrices (A,B), the generalized eigenvalues and the real Schur form (S,T)
- DGGEV - computes for a pair of N-by-N real nonsymmetric matrices (A,B) the generalized eigenvalues
- DGGEVX - computes for a pair of N-by-N real nonsymmetric matrices (A,B) the generalized eigenvalues
- DGGGLM - solves a general Gauss-Markov linear model (GLM) problem
- DGGHRD - reduces a pair of real matrices (A,B) to generalized upper Hessenberg form using orthogonal transformations, where A is a general matrix and B is upper triangular
- DGGLSE - solves the linear equality-constrained least squares (LSE) problem
- DGGQRF - computes a generalized QR factorization of an N-by-M matrix A and an N-by-P matrix B
- DGGRQF - computes a generalized RQ factorization of an M-by-N matrix A and a P-by-N matrix B
- DGGSV - computes the generalized singular value decomposition (GSVD) of an M-by-N real matrix A and P-by-N real matrix B
- DGGSV - computes orthogonal matrices U, V and Q
- DGTCON - estimates the reciprocal of the condition number of a real tridiagonal matrix A using the LU factorization as computed by DGTTRF
- DGTRFS - improves the computed solution to a system of linear equations when the coefficient matrix is tridiagonal
- DGTSV - solves the equation $AX = B$

- DGTSVX - uses the LU factorization to compute the solution to a real system of linear equations
- DGTTRF - computes an LU factorization of a real tridiagonal matrix A using elimination with partial pivoting and row interchanges
- DGTTRS - solves one of the systems of equations $AX = B$ or $A'X = B$
- DGTTS2 - solves one of the systems of equations $AX = B$ or $A'X = B$
- DHGEQZ - implements a single-/double-shift version of the QZ method for finding generalized eigenvalues
- DHSEIN - uses inverse iteration to find specified right and/or left eigenvectors of a real upper Hessenberg matrix H
- DHSEQR - computes the eigenvalues of a real upper Hessenberg matrix H
- DLABAD - returns the square root of values
- DLABRD - reduces the first NB rows and columns of a real general m by n matrix A to upper or lower bidiagonal form by an orthogonal transformation
- DLACON - estimates the 1-norm of a square, real matrix A
- DLACPY - copies all or part of a two-dimensional matrix A to another matrix B
- DLADIV - performs complex division in real arithmetic
- DLAE2 - computes the eigenvalues of a 2-by-2 symmetric matrix
- DLAE2Z - contains the iteration loops which compute and use the function $N(w)$
- DLAED0 - computes all eigenvalues and corresponding eigenvectors of a symmetric tridiagonal matrix using the divide and conquer method
- DLAED1 - computes the updated eigensystem of a diagonal matrix after modification by a rank-one symmetric matrix
- DLAED2 - merges the two sets of eigenvalues together into a single sorted set
- DLAED3 - finds the roots of the secular equation, as defined by the values in D, W, and RHO, between 1 and K
- DLAED4 - computes the I-th updated eigenvalue of a symmetric rank-one modification to a diagonal matrix

- DLAED5 - computes the I-th eigenvalue of a symmetric rank-one modification of a 2-by-2 diagonal matrix
- DLAED6 - computes the positive or negative root (closest to the origin)
- DLAED7 - computes the updated eigensystem of a diagonal matrix after modification by a rank-one symmetric matrix
- DLAED8 - merges the two sets of eigenvalues together into a single sorted set
- DLAED9 - finds the roots of the secular equation, as defined by the values in D, Z, and RHO, between KSTART and KSTOP
- DLAEDA - computes the Z vector corresponding to the merge step in the CURLVLth step of the merge process with TLVLS steps for the CURPBMth problem
- DLAEIN - uses inverse iteration to find a right or left eigenvector corresponding to the eigenvalue (WR,WI) of a real upper Hessenberg matrix H
- DLAEV2 - computes the eigendecomposition of a 2-by-2 symmetric matrix
- DLAEXC - swaps adjacent diagonal blocks T11 and T22 of order 1 or 2 in an upper quasi-triangular matrix T by an orthogonal similarity transformation
- DLAG2 - computes the eigenvalues of a 2 x 2 generalized eigenvalue problem with scaling as necessary to avoid over-/underflow
- DLAGS2 - computes 2-by-2 orthogonal matrices U, V and Q
- DLAGTF - factorizes a matrix
- DLAGTM - performs a matrix-vector product
- DLAGTS - solves one of two systems of equations
- DLAGV2 - computes the Generalized Schur factorization of a real 2-by-2 matrix pencil (A,B) where B is upper triangular
- DLAHQQR - updates the eigenvalues and Schur decomposition already computed by DHSEQR
- DLAHRD - reduces the first NB columns of a real general n-by-(n-k+1) matrix A so that elements below the kth subdiagonal are zero
- DLAIC1 - applies one step of incremental condition estimation in its simplest version

- DLALN2 - solves a system with possible scaling and perturbation of A
- DLALSO - applies back the multiplying factors of either the left or the right singular vector matrix of a diagonal matrix appended by a row to the right hand side matrix B in solving the least squares problem using the divide-and-conquer SVD approach
- DLALSA - an intermediate step in solving the least squares problem by computing the SVD of the coefficient matrix in compact form
- DLALSD - uses the singular value decomposition of A to solve the least squares problem
- DLAMCH - determines double precision machine parameters
- DLAMRG - creates a permutation list which merges the elements of A
- DLANGB - returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of an n by n band matrix A
- DLANGE - returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a real matrix A
- DLANGT - returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a real tridiagonal matrix A
- DLANH5 - returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a Hessenberg matrix A
- DLANSB - returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of an n by n symmetric band matrix A, with k super-diagonals
- DLANSP - returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a real symmetric matrix A, supplied in packed form
- DLANST - returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a real symmetric tridiagonal matrix A
- DLANSY - returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a real symmetric matrix A

- DLANTB - returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of an n by n triangular band matrix A , with $(k + 1)$ diagonals
- DLANTP - returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a triangular matrix A , supplied in packed form
- DLANTR - returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a trapezoidal or triangular matrix A
- DLANV2 - computes the Schur factorization of a real 2-by-2 nonsymmetric matrix in standard form
- DLAPLL - computes the QR factorization of $A=QR$
- DLAPMT - rearranges the columns of the M by N matrix X
- DLAPY2 - returns $\sqrt{x^2+y^2}$ without causing unnecessary overflow
- DLAPY3 - returns $\sqrt{x^2+y^2+z^2}$ without causing unnecessary overflow
- DLAQGB - equilibrates a general M by N band matrix A with KL subdiagonals and KU superdiagonals using the row and scaling factors in the vectors R and C
- DLAQGE - equilibrates a general M by N matrix A using the row and scaling factors in the vectors R and C
- DLAQP2 - computes a QR factorization with column pivoting of the block $A(\text{OFFSET}+1:M,1:N)$
- DLAQPS - computes a step of QR factorization with column pivoting of a real M -by- N matrix A by using Blas-3
- DLAQSB - equilibrates a symmetric band matrix A using the scaling factors in the vector S
- DLAQSP - equilibrates a symmetric matrix A using the scaling factors in the vector S
- DLAQSY - equilibrates a symmetric matrix A using the scaling factors in the vector S
- DLAQTR - solves a real quasi-triangular system

- DLAR1V - computes the (scaled) r^{th} column of the inverse of a submatrix
- DLAR2V - applies a vector of real plane rotations from both sides to a sequence of 2-by-2 real symmetric matrices, defined by the elements of the vectors x , y and z
- DLARF - applies a real elementary reflector H to a real m by n matrix C , from either the left or the right
- DLARFB - applies a real block reflector H or its transpose H' to a real m by n matrix C , from either the left or the right
- DLARFG - generates a real elementary reflector H of order n
- DLARFT - forms the triangular factor T of a real block reflector H of order n , which is defined as a product of k elementary reflectors
- DLARFX - applies a real elementary reflector H to a real m by n matrix C , from either the left or the right
- DLARGV - generates a vector of real plane rotations, determined by elements of the real vectors x and y
- DLARNV - returns a vector of n random real numbers from a uniform or normal distribution
- DLARRB - does limited bisection to locate eigenvalues
- DLARRE - sets "small" off-diagonal elements to zero
- DLARRF - finds a robust representation of input values
- DLARRV - computes the eigenvectors of the tridiagonal matrix
- DLARTG - generates a plane rotation
- DLARTV - applies a vector of real plane rotations to elements of the real vectors x and y
- DLARUV - returns a vector of n random real numbers from a uniform (0,1)
- DLARZ - applies a real elementary reflector H to a real M -by- N matrix C , from either the left or the right
- DLARZB - applies a real block reflector H or its transpose to a real distributed M -by- N C from the left or the right

- DLARZT - forms the triangular factor T of a real block reflector H of order $> n$, which is defined as a product of k elementary reflectors
- DLAS2 - computes the singular values of the 2-by-2 matrix
- DLASCL - multiplies the M by N real matrix A by the real scalar CTO/CFROM
- DLASD0 - computes the singular value decomposition (SVD) of a real upper bidiagonal N-by-M matrix B
- DLASD1 - computes the SVD of an upper bidiagonal N-by-M matrix B
- DLASD2 - merges the two sets of singular values together into a single sorted set
- DLASD3 - finds all the square roots of the roots of the secular equation, as defined by the values in D and Z
- DLASD4 - computes the square root of the I^{th} updated eigenvalue of a positive symmetric rank-one modification to a positive diagonal matrix
- DLASD5 - computes the square root of the I^{th} eigenvalue of a positive symmetric rank-one modification of a 2-by-2 diagonal matrix
- DLASD6 - computes the SVD of an updated upper bidiagonal matrix B obtained by merging two smaller ones by appending a row
- DLASD7 - merges the two sets of singular values together into a single sorted set
- DLASD8 - finds the square roots of the roots of the secular equation,
- DLASD9 - finds the square roots of the roots of the secular equation,
- DLASDA - computes the singular value decomposition (SVD) of a real upper bidiagonal N-by-M matrix B with diagonal D and offdiagonal E
- DLASDQ - computes the singular value decomposition (SVD) of a real (upper or lower) bidiagonal matrix with diagonal D and offdiagonal E, accumulating the transformations if desired
- DLASDT - creates a tree of subproblems for bidiagonal divide and conquer
- DLASET - initializes an m-by-n matrix A to BETA on the diagonal and ALPHA on the offdiagonals
- DLASQ1 - computes the singular values of a real N-by-N bidiagonal matrix with diagonal D and off-diagonal E

- DLASQ2 - computes all the eigenvalues of the symmetric positive definite tridiagonal matrix
- DLASQ3 - computes a shift (TAU)
- DLASQ4 - computes an approximation TAU to the smallest eigenvalue using values of d from the previous transform
- DLASQ5 - computes one dqds transform in ping-pong form, one version for IEEE machines another for non IEEE machines
- DLASQ6 - computes one dqd (shift equal to zero) transform in ping-pong form, with protection against underflow and overflow
- DLASR - perform a transformation where A is an m by n real matrix and P is an orthogonal matrix,
- DLASRT - sorts numbers
- DLASSQ - returns the values scl and smsq
- DLASV2 - computes the singular value decomposition of a 2-by-2 triangular matrix
- DLASWP - performs a series of row interchanges on the matrix A
- DLASY2 - solves for the N1 by N2 matrix X
- DLASYF - computes a partial factorization of a real symmetric matrix A using the Bunch-Kaufman diagonal pivoting method
- DLATBS - solves one of two triangular systems with scaling to prevent overflow, where A is an upper or lower triangular band matrix
- DLATDF - uses the LU factorization of the n-by-n matrix Z computed by DGETC2
- DLATPS - solves a triangular system with scaling to prevent overflow
- DLATRD - reduces NB rows and columns of a real symmetric matrix A to symmetric tridiagonal form
- DLATRS - solves a triangular system with scaling to prevent overflow
- DLATRZ - factors the M-by-(M+L) real upper trapezoidal matrix by means of orthogonal transformations
- DLATZM - routine is deprecated and has been replaced by routine DORMRZ

- DLAUU2 - computes the product $U \times U'$ or $L' \times L$, where the triangular factor U or L is stored in the upper or lower triangular part of the array A
- DLAUUM - computes the product $U \times U'$ or $L' \times L$, where the triangular factor U or L is stored in the upper or lower triangular part of the array A
- DOPGTR - generates a real orthogonal matrix Q which is defined as the product of $n-1$ elementary reflectors $H(i)$ of order n , as returned by DSPTRD using packed storage
- DOPMTR - overwrites the general real M -by- N matrix C with $SIDE = 'L'$ $SIDE = 'R'$ $TRANS = 'N'$
- DORG2L - generates an m by n real matrix Q with orthonormal columns
- DORG2R - generates an m by n real matrix Q with orthonormal columns
- DORGBR - generates one of the real orthogonal matrices Q or P^T determined by DGEBRD when reducing a real matrix A to bidiagonal form
- DORGHR - generates a real orthogonal matrix Q which is defined as the product of IHI-ILO elementary reflectors of order N , as returned by DGEHRD
- DORGL2 - generates an m by n real matrix Q with orthonormal rows
- DORGLQ - generates an M -by- N real matrix Q with orthonormal rows
- DORGQL - generates an M -by- N real matrix Q with orthonormal columns
- DORGQR - generates an M -by- N real matrix Q with orthonormal columns
- DORGR2 - generates an m by n real matrix Q with orthonormal rows
- DORGRQ - generates an M -by- N real matrix Q with orthonormal rows
- DORGTR - generates a real orthogonal matrix Q as returned by DSYTRD
- DORM2L - overwrites the general real m by n matrix C
- DORM2R - overwrites the general real m by n matrix C
- DORMBR - overwrites the general real M -by- N matrix C
- DORMHR - overwrites the general real M -by- N matrix C
- DORML2 - overwrites the general real m by n matrix C
- DORMLQ - overwrites the general real M -by- N matrix C

- DORMQL - overwrites the general real M-by-N matrix C
- DORMQR - overwrites the general real M-by-N matrix C
- DORMR2 - overwrites the general real m by n matrix C
- DORMR3 - overwrites the general real m by n matrix C
- DORMRQ - overwrites the general real M-by-N matrix C
- DORMRZ - overwrites the general real M-by-N matrix C
- DORMTR - overwrites the general real M-by-N matrix C
- DPBCON - estimates the reciprocal of the condition number (in the 1-norm) of a real symmetric positive definite band matrix using the Cholesky factorization computed by DPBTRF
- DPBEQU - computes row and column scalings intended to equilibrate a symmetric positive definite band matrix A and reduce its condition number (with respect to the two-norm)
- DPBRFS - improves the computed solution to a system of linear equations when the coefficient matrix is symmetric positive definite and banded, and provides error bounds and backward error estimates for the solution
- DPBSTF - computes a split Cholesky factorization of a real symmetric positive definite band matrix A
- DPBSV - computes the solution to a real system of linear equations
- DPBSVX - uses the Cholesky factorization to compute the solution to a real system of linear equations
- DPBTF2 - computes the Cholesky factorization of a real symmetric positive definite band matrix A
- DPBTRF - computes the Cholesky factorization of a real symmetric positive definite band matrix A
- DPBTRS - solves a system of linear equations with a symmetric positive definite band matrix A using the Cholesky factorization computed by DPBTRF
- DPOCON - estimates the reciprocal of the condition number (in the 1-norm) of a real symmetric positive definite matrix using the Cholesky factorization computed by DPOTRF

- DPOEQU - computes row and column scalings intended to equilibrate a symmetric positive definite matrix A and reduce its condition number (with respect to the two-norm)
- DPORFS - improves the computed solution to a system of linear equations when the coefficient matrix is symmetric positive definite
- DPOSV - computes the solution to a real system of linear equations
- DPOSVX - uses the Cholesky factorization to compute the solution to a real system of linear equations
- DPOTF2 - computes the Cholesky factorization of a real symmetric positive definite matrix A
- DPOTRF - computes the Cholesky factorization of a real symmetric positive definite matrix A
- DPOTRI - computes the inverse of a real symmetric positive definite matrix A using the Cholesky factorization computed by DPOTRF
- DPOTRS - solves a system of linear equations with a symmetric positive definite matrix A using the Cholesky factorization computed by DPOTRF
- DPPCON - estimates the reciprocal of the condition number (in the 1-norm) of a real symmetric positive definite packed matrix using the Cholesky factorization computed by DPPTRF
- DPPEQU - computes row and column scalings intended to equilibrate a symmetric positive definite matrix A in packed storage and reduce its condition number (with respect to the two-norm)
- DPPRFS - improves the computed solution to a system of linear equations when the coefficient matrix is symmetric positive definite and packed
- DPPSV - computes the solution to a real system of linear equations
- DPPSVX - uses the Cholesky factorization to compute the solution to a real system of linear equations
- DPPTRF - computes the Cholesky factorization of a real symmetric positive definite matrix A stored in packed format
- DPPTRI - computes the inverse of a real symmetric positive definite matrix A using the Cholesky factorization computed by DPPTRF

- DPPTRS - solves a system of linear equations with a symmetric positive definite matrix A in packed storage using the Cholesky factorization computed by DPPTRF
- DPTCON - computes the reciprocal of the condition number (in the 1-norm) of a real symmetric positive definite tridiagonal matrix using the factorization computed by DPTTRF
- DPTEQR - computes all eigenvalues and, optionally, eigenvectors of a symmetric positive definite tridiagonal matrix
- DPTRFS - improves the computed solution to a system of linear equations when the coefficient matrix is symmetric positive definite and tridiagonal
- DPTSV - computes the solution to a real system of linear equations
- DPTSVX - computes the solution to a real system of linear equations where A is an N-by-N symmetric positive definite tridiagonal matrix and X and B are N-by-NRHS matrices
- DPTTRF - computes the factorization of a real symmetric positive definite tridiagonal matrix A
- DPTTRS - solves a tridiagonal system using the factorization of A computed by DPTTRF
- DPTTS2 - solves a tridiagonal system using the factorization of A computed by DPTTRF
- DRSCL - multiplies an n-element real vector x by the real scalar 1/a
- DSBEV - computes all the eigenvalues and, optionally, eigenvectors of a real symmetric band matrix A
- DSBEVD - computes all the eigenvalues and, optionally, eigenvectors of a real symmetric band matrix A
- DSBEVX - computes selected eigenvalues and, optionally, eigenvectors of a real symmetric band matrix A
- DSBGST - reduces a real symmetric-definite banded generalized eigenproblem
- DSBGV - computes all the eigenvalues, and optionally, the eigenvectors of a real generalized symmetric-definite banded eigenproblem
- DSBGVD - computes all the eigenvalues, and optionally, the eigenvectors of a real generalized symmetric-definite banded eigenproblem

- DSBGVX - computes selected eigenvalues, and optionally, eigenvectors of a real generalized symmetric-definite banded eigenproblem
- DSBTRD - reduces a real symmetric band matrix A to symmetric tridiagonal form T by an orthogonal similarity transformation
- DSECND - returns the user time for a process in seconds
- DSPCON - estimates the reciprocal of the condition number (in the 1-norm) of a real symmetric packed matrix A
- DSPEV - computes all the eigenvalues and, optionally, eigenvectors of a real symmetric matrix A in packed storage
- DSPEVD - computes all the eigenvalues and, optionally, eigenvectors of a real symmetric matrix A in packed storage
- DSPEVX - computes selected eigenvalues and, optionally, eigenvectors of a real symmetric matrix A in packed storage
- DSPGST - reduces a real symmetric-definite generalized eigenproblem to standard form, using packed storage
- DSPGV - computes all the eigenvalues and, optionally, the eigenvectors of a real generalized symmetric-definite eigenproblem
- DSPGVD - computes all the eigenvalues, and optionally, the eigenvectors of a real generalized symmetric-definite eigenproblem
- DSPGVX - computes selected eigenvalues, and optionally, eigenvectors of a real generalized symmetric-definite eigenproblem
- DSPRFS - improves the computed solution to a system of linear equations when the coefficient matrix is symmetric indefinite and packed
- DSPSV - computes the solution to a real system of linear equations
- DSPSVX - uses the diagonal pivoting factorization to compute the solution to a real system of linear equations where A is an N -by- N symmetric matrix stored in packed format and X and B are N -by- $NRHS$ matrices
- DSPTRD - reduces a real symmetric matrix A stored in packed form to symmetric tridiagonal form T by an orthogonal similarity transformation
- DSPTRF - computes the factorization of a real symmetric matrix A stored in packed format using the Bunch-Kaufman diagonal pivoting method

- DSPTRI - computes the inverse of a real symmetric indefinite matrix A in packed storage using a factorization computed by DSPTRF
- DSPTRS - solves a system of linear equations with a real symmetric matrix A stored in packed format using a factorization computed by DSPTRF
- DSTEBZ - computes the eigenvalues of a symmetric tridiagonal matrix T
- DSTEDC - computes all eigenvalues and, optionally, eigenvectors of a symmetric tridiagonal matrix using the divide and conquer method
- DSTEGR - computes selected eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix T
- DSTEIN - computes the eigenvectors of a real symmetric tridiagonal matrix T corresponding to specified eigenvalues, using inverse iteration
- DSTEQR - computes all eigenvalues and, optionally, eigenvectors of a symmetric tridiagonal matrix using the implicit QL or QR method
- DSTERF - computes all eigenvalues of a symmetric tridiagonal matrix using the Pal-Walker-Kahan variant of the QL or QR algorithm
- DSTEV - computes all eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix A
- DSTEVD - computes all eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix
- DSTEVR - computes selected eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix T
- DSTEVM - computes selected eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix A
- DSYCON - estimates the reciprocal of the condition number (in the 1-norm) of a real symmetric matrix A using a factorization computed by DSYTRF
- DSYEV - computes all eigenvalues and, optionally, eigenvectors of a real symmetric matrix A
- DSYEVD - computes all eigenvalues and, optionally, eigenvectors of a real symmetric matrix A
- DSYEVR - computes selected eigenvalues and, optionally, eigenvectors of a real symmetric matrix T

- DSYEVX - computes selected eigenvalues and, optionally, eigenvectors of a real symmetric matrix A
- DSYGS2 - reduces a real symmetric-definite generalized eigenproblem to standard form
- DSYGST - reduces a real symmetric-definite generalized eigenproblem to standard form
- DSYGV - computes all the eigenvalues, and optionally, the eigenvectors of a real generalized symmetric-definite eigenproblem
- DSYGVD - computes all the eigenvalues, and optionally, the eigenvectors of a real generalized symmetric-definite eigenproblem
- DSYGVX - computes selected eigenvalues, and optionally, eigenvectors of a real generalized symmetric-definite eigenproblem
- DSYRFS - improves the computed solution to a system of linear equations when the coefficient matrix is symmetric indefinite, and provides error bounds and backward error estimates for the solution
- DSYSV - computes the solution to a real system of linear equations
- DSYSVX - uses the diagonal pivoting factorization to compute the solution to a real system of linear equations
- DSYTD2 - reduces a real symmetric matrix A to symmetric tridiagonal form T by an orthogonal similarity transformation
- DSYTF2 - computes the factorization of a real symmetric matrix A using the Bunch-Kaufman diagonal pivoting method
- DSYTRD - reduces a real symmetric matrix A to real symmetric tridiagonal form T by an orthogonal similarity transformation
- DSYTRF - computes the factorization of a real symmetric matrix A using the Bunch-Kaufman diagonal pivoting method
- DSYTRI - computes the inverse of a real symmetric indefinite matrix A using a factorization computed by DSYTRF
- DSYTRS - solves a system of linear equations with a real symmetric matrix A using a factorization computed by DSYTRF

- DTBCON - estimates the reciprocal of the condition number of a triangular band matrix A , in either the 1-norm or the infinity-norm
- DTBRFS - provides error bounds and backward error estimates for the solution to a system of linear equations with a triangular band coefficient matrix
- DTBTRS - solves a triangular system
- DTGEVC - computes some or all of the right and/or left generalized eigenvectors of a pair of real upper triangular matrices (A,B)
- DTGEX2 - swaps adjacent diagonal blocks (A_{11}, B_{11}) and (A_{22}, B_{22})
- DTGEXC - reorders the generalized real Schur decomposition of a real matrix pair (A,B)
- DTGSEN - reorders the generalized real Schur decomposition of a real matrix pair (A, B)
- DTGSJA - computes the generalized singular value decomposition (GSVD) of two real upper triangular (or trapezoidal) matrices A and B
- DTGSNA - estimates reciprocal condition numbers for specified eigenvalues and/or eigenvectors of a matrix pair (A, B) in generalized real Schur canonical form
- DTGSY2 - solves the generalized Sylvester equation
- DTGSYL - solves the generalized Sylvester equation
- DTPCON - estimates the reciprocal of the condition number of a packed triangular matrix A , in either the 1-norm or the infinity-norm
- DTPRFS - provides error bounds and backward error estimates for the solution to a system of linear equations with a triangular packed coefficient matrix
- DTPTRI - computes the inverse of a real upper or lower triangular matrix A stored in packed format
- DTPTRS - solves a triangular system
- DTRCON - estimates the reciprocal of the condition number of a triangular matrix A , in either the 1-norm or the infinity-norm
- DTREVC - computes some or all of the right and/or left eigenvectors of a real upper quasi-triangular matrix T

- DTREXC - reorders the real Schur factorization of a real matrix so that the diagonal block of T with row index IFST is moved to row ILST
- DTRID - computes the solution to a real system of linear equations where A is an N-by-N tridiagonal matrix, and x and b are vectors of length N
- DTRRFS - provide error bounds and backward error estimates for the solution to a system of linear equations with a triangular coefficient matrix
- DTRSEN - reorders the real Schur factorization of a real matrix so that a selected cluster of eigenvalues appears in the leading diagonal blocks of the upper quasi-triangular matrix T
- DTRSNA - estimates reciprocal condition numbers for specified eigenvalues and/or right eigenvectors of a real upper quasi-triangular matrix T
- DTRSYL - solves the real Sylvester matrix equation
- DTRTI2 - computes the inverse of a real upper or lower triangular matrix
- DTRTRI - computes the inverse of a real upper or lower triangular matrix A
- DTRTRS - solves a triangular system
- DTZRQF - routine is deprecated and has been replaced by routine DTZRZF
- DTZRZF - reduces the M-by-N real upper trapezoidal matrix A to upper triangular form by means of orthogonal transformations
- DZSUM1 - takes the sum of the absolute values of a complex vector and returns a double precision result
- ICMAX1 - finds the index of the element whose real part has maximum absolute value
- ILAENV - called from the LAPACK routines to choose problem-dependent parameters for the local environment
- IZMAX1 - finds the index of the element whose real part has maximum absolute value
- LSAME - return .TRUE
- LSAMEN - tests if the first N letters of CA are the same as the first N letters of CB, regardless of case

- SBDSDC - computes the singular value decomposition (SVD) of a real N-by-N (upper or lower) bidiagonal matrix B
- SBDSQR - computes the singular value decomposition (SVD) of a real N-by-N (upper or lower) bidiagonal matrix B
- SCSUM1 - take the sum of the absolute values of a complex vector and returns a single precision result
- SDISNA - computes the reciprocal condition numbers for the eigenvectors of a real symmetric or complex Hermitian matrix or for the left or right singular vectors of a general m-by-n matrix
- SECOND - returns the user time for a process in seconds
- SGBBRD - reduces a real general m-by-n band matrix A to upper bidiagonal form B by an orthogonal transformation
- SGBCON - estimates the reciprocal of the condition number of a real general band matrix A, in either the 1-norm or the infinity-norm,
- SGBEQU - computes row and column scalings intended to equilibrate an M-by-N band matrix A and reduce its condition number
- SGBRFS - improves the computed solution to a system of linear equations when the coefficient matrix is banded
- SGBSV - computes the solution to a real system of linear equations where A is a band matrix of order N with KL subdiagonals and KU superdiagonals, and X and B are N-by-NRHS matrices
- SGBSVX - uses the LU factorization to compute the solution to a real system of linear equations
- SGBTF2 - computes an LU factorization of a real m-by-n band matrix A using partial pivoting with row interchanges
- SGBTRF - computes an LU factorization of a real m-by-n band matrix A using partial pivoting with row interchanges
- SGBTRS - solves a system of linear equations with a general band matrix A using the LU factorization computed by SGBTRF
- SGEBAK - forms the right or left eigenvectors of a real general matrix by backward transformation on the computed eigenvectors of the balanced matrix output by SGEBAL

- SGEBAL - balances a general real matrix A
- SGEBD2 - reduces a real general m by n matrix A to upper or lower bidiagonal form B by an orthogonal transformation
- SGEBRD - reduces a general real M-by-N matrix A to upper or lower bidiagonal form B by an orthogonal transformation
- SGECON - estimates the reciprocal of the condition number of a general real matrix A, in either the 1-norm or the infinity-norm, using the LU factorization computed by SGETRF
- SGEEQU - computes row and column scalings intended to equilibrate an M-by-N matrix A and reduce its condition number
- SGEES - computes for an N-by-N real nonsymmetric matrix A, the eigenvalues, the real Schur form T, and, optionally, the matrix of Schur vectors Z
- SGEESX - computes for an N-by-N real nonsymmetric matrix A, the eigenvalues, the real Schur form T, and, optionally, the matrix of Schur vectors Z
- SGEEV - computes for an N-by-N real nonsymmetric matrix A, the eigenvalues and, optionally, the left and/or right eigenvectors
- SGEEVX - computes for an N-by-N real nonsymmetric matrix A, the eigenvalues and, optionally, the left and/or right eigenvectors
- SGEES - routine is deprecated and has been replaced by routine SGEES
- SGEGV - routine is deprecated and has been replaced by routine SGGEV
- SGEHD2 - reduces a real general matrix A to upper Hessenberg form H by an orthogonal similarity transformation
- SGEHRD - reduces a real general matrix A to upper Hessenberg form H by an orthogonal similarity transformation
- SGELQ2 - computes an LQ factorization of a real m by n matrix A
- SGELQF - computes an LQ factorization of a real M-by-N matrix A
- SGELS - solves overdetermined or underdetermined real linear systems involving an M-by-N matrix A, or its transpose, using a QR or LQ factorization of A
- SGELSD - computes the minimum-norm solution to a real linear least squares problem

- SGELSS - computes the minimum norm solution to a real linear least squares problem
- SGELSX - routine is deprecated and has been replaced by routine SGELSY
- SGELSY - computes the minimum-norm solution to a real linear least squares problem
- SGEQL2 - computes a QL factorization of a real m by n matrix A
- SGEQLF - computes a QL factorization of a real M -by- N matrix A
- SGEQP3 - computes a QR factorization with column pivoting of a matrix A
- SGEQPF - routine is deprecated and has been replaced by routine SGEQP3
- SGEQR2 - computes a QR factorization of a real m by n matrix A
- SGEQRF - computes a QR factorization of a real M -by- N matrix A
- SGERFS - improves the computed solution to a system of linear equations and provides error bounds and backward error estimates for the solution
- SGERQ2 - computes an RQ factorization of a real m by n matrix A
- SGERQF - computes an RQ factorization of a real M -by- N matrix A
- SGESC2 - solves a system of linear equations with a general N -by- N matrix A using the LU factorization with complete pivoting computed by SGETC2
- SGESDD - computes the singular value decomposition (SVD) of a real M -by- N matrix A
- SGESV - computes the solution to a real system of linear equations
- SGESVD - computes the singular value decomposition (SVD) of a real M -by- N matrix A
- SGESVX - uses the LU factorization to compute the solution to a real system of linear equations
- SGETC2 - computes an LU factorization with complete pivoting of the n -by- n matrix A
- SGETF2 - computes an LU factorization of a general m -by- n matrix A using partial pivoting with row interchanges

- SGETRF - computes an LU factorization of a general M-by-N matrix A using partial pivoting with row interchanges
- SGETRI - computes the inverse of a matrix using the LU factorization computed by SGETRF
- SGETRS - solves a system of linear equations with a general N-by-N matrix A using the LU factorization computed by SGETRF
- SGGBAK - forms the right or left eigenvectors of a real generalized eigenvalue problem by backward transformation on the computed eigenvectors of the balanced pair of matrices output by SGGBAL
- SGGBAL - balances a pair of general real matrices (A,B)
- SGGES - computes for a pair of N-by-N real nonsymmetric matrices (A,B),
- SGGESX - computes for a pair of N-by-N real nonsymmetric matrices (A,B), the generalized eigenvalues, the real Schur form (S,T), and,
- SGGEV - computes for a pair of N-by-N real nonsymmetric matrices (A,B)
- SGGEVX - computes for a pair of N-by-N real nonsymmetric matrices (A,B)
- SGGGLM - solves a general Gauss-Markov linear model (GLM) problem
- SGGHRD - reduces a pair of real matrices (A,B) to generalized upper Hessenberg form using orthogonal transformations, where A is a general matrix and B is upper triangular
- SGGLSE - solves the linear equality-constrained least squares (LSE) problem
- SGGQRF - computes a generalized QR factorization of an N-by-M matrix A and an N-by-P matrix B
- SGGRQF - computes a generalized RQ factorization of an M-by-N matrix A and a P-by-N matrix B
- SGGSD - computes the generalized singular value decomposition (GSVD) of an M-by-N real matrix A and P-by-N real matrix B
- SGGSP - computes orthogonal matrices U, V and Q
- SGTCON - estimates the reciprocal of the condition number of a real tridiagonal matrix A using the LU factorization as computed by SGTTRF

- SGTRFS - improves the computed solution to a system of linear equations when the coefficient matrix is tridiagonal
- SGTSV - solves the equation $AX = B$,
- SGTSVX - uses the LU factorization to compute the solution to a real system of linear equations
- SGTTRF - computes an LU factorization of a real tridiagonal matrix A using elimination with partial pivoting and row interchanges
- SGTTRS - solves one of two systems of equations
- SGTTS2 - solves one of two systems of equations
- SHGEQZ - implements a single-/double-shift version of the QZ method for finding generalized eigenvalues
- SHSEIN - uses inverse iteration to find specified right and/or left eigenvectors of a real upper Hessenberg matrix H
- SHSEQR - computes the eigenvalues of a real upper Hessenberg matrix H and, optionally, the matrices T and Z from the Schur decomposition
- SLABAD - takes as input the values computed by SLAMCH for underflow and overflow, and returns the square root of each of these values if the log of LARGE is sufficiently large
- SLABRD - reduces the first NB rows and columns of a real general m by n matrix A to upper or lower bidiagonal form by an orthogonal transformation
- SLACON - estimates the 1-norm of a square, real matrix A
- SLACPY - copies all or part of a two-dimensional matrix A to another matrix B
- SLADIV - performs complex division in real arithmetic
- SLAE2 - computes the eigenvalues of a 2-by-2 symmetric matrix
- SLAEBZ - contains the iteration loops which compute and use the function $N(w)$
- SLAED0 - computes all eigenvalues and corresponding eigenvectors of a symmetric tridiagonal matrix using the divide and conquer method
- SLAED1 - computes the updated eigensystem of a diagonal matrix after modification by a rank-one symmetric matrix

- SLAED2 - merges the two sets of eigenvalues together into a single sorted set
- SLAED3 - finds the roots of the secular equation, as defined by the values in D, W, and RHO, between 1 and K
- SLAED4 - computes the I^{th} updated eigenvalue of a symmetric rank-one modification to a diagonal matrix
- SLAED5 - computes the I^{th} eigenvalue of a symmetric rank-one modification of a 2-by-2 diagonal matrix
- SLAED6 - computes the positive or negative root (closest to the origin)
- SLAED7 - computes the updated eigensystem of a diagonal matrix after modification by a rank-one symmetric matrix
- SLAED8 - merges the two sets of eigenvalues together into a single sorted set
- SLAED9 - finds the roots of the secular equation, as defined by the values in D, Z, and RHO, between KSTART and KSTOP
- SLAEDA - computes the Z vector corresponding to the merge step in the CURLVLth step of the merge process with TLVLS steps for the CURPBMth problem
- SLAEIN - uses inverse iteration to find a right or left eigenvector corresponding to the eigenvalue (WR,WI) of a real upper Hessenberg matrix H
- SLAEV2 - computes the eigendecomposition of a 2-by-2 symmetric matrix
- SLAEXC - swaps adjacent diagonal blocks T11 and T22 of order 1 or 2 in an upper quasi-triangular matrix T by an orthogonal similarity transformation
- SLAG2 - computes the eigenvalues of a 2 x 2 generalized eigenvalue problem with scaling as necessary
- SLAGS2 - computes 2-by-2 orthogonal matrices
- SLAGTF - factorizes the matrix where T is an n by n tridiagonal matrix and lambda is a scalar
- SLAGTM - performs a matrix-vector product
- SLAGTS - solves one of two systems of equations
- SLAGV2 - computes the Generalized Schur factorization of a real 2-by-2 matrix pencil (A,B) where B is upper triangular

- SLAHQR - an auxiliary routine called by SHSEQR to update the eigenvalues and Schur decomposition already computed by SHSEQR
- SLAHRD - reduces the first NB columns of a real general n-by-(n-k+1) matrix A so that elements below the kth subdiagonal are zero
- SLAIC1 - applies one step of incremental condition estimation in its simplest version
- SLALN2 - solves a system with possible scaling ("s") and perturbation of A
- SLALS0 - applies back the multiplying factors of either the left or the right singular vector matrix of a diagonal matrix
- SLALSA - an intermediate step in solving the least squares problem by computing the SVD of the coefficient matrix in compact form
- SLALSD - uses the singular value decomposition of A to solve the least squares problem
- SLAMCH - determines single precision machine parameters
- SLAMRG - creates a permutation list that merges the elements of A (which is composed of two independently sorted sets) into a single set which is sorted in ascending order
- SLANGB - returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of an n by n band matrix A
- SLANGE - returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a real matrix A
- SLANGT - returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a real tridiagonal matrix A
- SLANH5 - returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a Hessenberg matrix A
- SLANSB - returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of an n by n symmetric band matrix A, with k super-diagonals
- SLANSP - returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a real symmetric matrix A, supplied in packed form

- SLANST - returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a real symmetric tridiagonal matrix A
- SLANSY - returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a real symmetric matrix A
- SLANTB - returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of an n by n triangular band matrix A
- SLANTP - returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a triangular matrix A
- SLANTR - returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a trapezoidal or triangular matrix A
- SLANV2 - computes the Schur factorization of a real 2-by-2 nonsymmetric matrix in standard form
- SLAPLL - computes the QR factorization of $A=QR$
- SLAPMT - rearranges the columns of the M by N matrix X
- SLAPY2 - returns $\sqrt{x^2+y^2}$ without causing unnecessary overflow
- SLAPY3 - returns $\sqrt{x^2+y^2+z^2}$ without causing unnecessary overflow
- SLAQGB - equilibrates a general M by N band matrix A with KL subdiagonals and KU superdiagonals
- SLAQGE - equilibrates a general M by N matrix A using the row and scaling factors in the vectors R and C
- SLAQP2 - computes a QR factorization with column pivoting of the block $A(\text{OFFSET}+1:M,1:N)$
- SLAQPS - computes a step of QR factorization with column pivoting of a real M-by-N matrix A by using Blas3
- SLAQSB - equilibrates a symmetric band matrix A using the scaling factors in the vector S
- SLAQSP - equilibrates a symmetric matrix A using the scaling factors in the vector S

- SLAQSY - equilibrates a symmetric matrix A using the scaling factors in the vector S
- SLAQTR - solves a real quasi-triangular system
- SLAR1V - computes the (scaled) r^{th} column of the inverse of the submatrix of a tridiagonal matrix
- SLAR2V - applies a vector of real plane rotations from both sides to a sequence of 2-by-2 real symmetric matrices, defined by the elements of the vectors x , y and z
- SLARF - applies a real elementary reflector H to a real m by n matrix C , from either the left or the right
- SLARFB - applies a real block reflector H or its transpose H' to a real m by n matrix C , from either the left or the right
- SLARFG - generates a real elementary reflector H of order n
- SLARFT - forms the triangular factor T of a real block reflector H of order n , which is defined as a product of k elementary reflectors
- SLARFX - applies a real elementary reflector H to a real m by n matrix C , from either the left or the right
- SLARGV - generates a vector of real plane rotations, determined by elements of the real vectors x and y
- SLARNV - returns a vector of n random real numbers from a uniform or normal distribution
- SLARRB - does limited bisection to locate eigenvalues
- SLARRE - sets "small" off-diagonal elements to zero
- SLARRF - finds a robust representation of input values.
- SLARRV - computes the eigenvectors of the tridiagonal matrix
- SLARTG - generates a plane rotation
- SLARTV - applies a vector of real plane rotations to elements of the real vectors x and y
- SLARUV - returns a vector of n random real numbers from a uniform (0,1)

- SLARZ - applies a real elementary reflector H to a real M-by-N matrix C, from either the left or the right
- SLARZB - applies a real block reflector H or its transpose to a real distributed M-by-N C from the left or the right
- SLARZT - forms the triangular factor T of a real block reflector H
- SLAS2 - computes the singular values of the 2-by-2 matrix
- SLASCL - multiplies the M by N real matrix A by the real scalar CTO/CFROM
- SLASD0 - computes the singular value decomposition (SVD) of a real upper bidiagonal N-by-M matrix B
- SLASD1 - computes the SVD of an upper bidiagonal N-by-M matrix B,
- SLASD2 - merges the two sets of singular values together into a single sorted set
- SLASD3 - finds all the square roots of the roots of the secular equation, as defined by the values in D and Z
- SLASD4 - computes the square root of the I^{th} updated eigenvalue of a positive symmetric rank-one modification to a positive diagonal matrix
- SLASD5 - computes the square root of the I^{th} eigenvalue of a positive symmetric rank-one modification of a 2-by-2 diagonal matrix
- SLASD6 - computes the SVD of an updated upper bidiagonal matrix B obtained by merging two smaller ones by appending a row
- SLASD7 - merges the two sets of singular values together into a single sorted set
- SLASD8 - finds the square roots of the roots of the secular equation,
- SLASD9 - finds the square roots of the roots of the secular equation,
- SLASDA - computes the singular value decomposition (SVD) of a real upper bidiagonal N-by-M matrix B with diagonal D and offdiagonal E
- SLASDQ - computes the singular value decomposition (SVD) of a real (upper or lower) bidiagonal matrix with diagonal D and offdiagonal E, accumulating the transformations if desired
- SLASDT - creates a tree of subproblems for bidiagonal divide and conquer

- SLASET - initializes an m-by-n matrix A to BETA on the diagonal and ALPHA on the offdiagonals
- SLASQ1 - computes the singular values of a real N-by-N bidiagonal matrix with diagonal D and off-diagonal E
- SLASQ2 - computes all the eigenvalues of the symmetric positive definite tridiagonal matrix associated with the qd array Z
- SLASQ3 - checks for deflation, computes a shift (TAU) and calls dqds
- SLASQ4 - computes an approximation TAU to the smallest eigenvalue using values of d from the previous transform
- SLASQ5 - computes some dqds transform in ping-pong form, one version for IEEE machines another for non IEEE machines
- SLASQ6 - computes one dqd (shift equal to zero) transform in ping-pong form, with protection against underflow and overflow
- SLASR - performs a transformation
- SLASRT - sorts numbers
- SLASSQ - returns the values scl and smsq
- SLASV2 - computes the singular value decomposition of a 2-by-2 triangular matrix
- SLASWP - performs a series of row interchanges on the matrix A
- SLASY2 - solves for the N1 by N2 matrix X
- SLASYF - computes a partial factorization of a real symmetric matrix A using the Bunch-Kaufman diagonal pivoting method
- SLATBS - solves one of two triangular systems with scaling to prevent overflow
- SLATDF - computes a contribution to the reciprocal Dif-estimate
- SLATPS - solves one of two triangular systems with scaling to prevent overflow
- SLATRD - reduces NB rows and columns of a real symmetric matrix A to symmetric tridiagonal form
- SLATRS - solves one of two triangular systems with scaling to prevent overflow

- SLATRZ - factors the M-by-(M+L) real upper trapezoidal matrix by means of orthogonal transformations
- SLATZM - routine is deprecated and has been replaced by routine SORMRZ
- SLAAU2 - computes the product $U \times U'$ or $L' \times L$
- SLAAUM - computes the product $U \times U'$ or $L' \times L$, where the triangular factor U or L is stored in the upper or lower triangular part of the array A
- SOPGTR - generates a real orthogonal matrix Q as returned by SSPTRD using packed storage
- SOPMTR - overwrites the general real M-by-N matrix C
- SORG2L - generates an m by n real matrix Q with orthonormal columns,
- SORG2R - generates an m by n real matrix Q with orthonormal columns,
- SORGBR - generates one of the real orthogonal matrices determined by SGEBRD when reducing a real matrix A to bidiagonal form
- SORGHR - generates a real orthogonal matrix Q as returned by SGEHRD
- SORGL2 - generates an m by n real matrix Q with orthonormal rows
- SORGLQ - generates an M-by-N real matrix Q with orthonormal rows
- SORGQL - generates an M-by-N real matrix Q with orthonormal columns
- SORGQR - generates an M-by-N real matrix Q with orthonormal columns
- SORGR2 - generates an m by n real matrix Q with orthonormal rows
- SORGRQ - generates an M-by-N real matrix Q with orthonormal rows
- SORGTR - generates a real orthogonal matrix Q as returned by SSYTRD
- SORM2L - overwrites the general real m by n matrix C
- SORM2R - overwrites the general real m by n matrix C with Q
- SORMBR - VECT = 'Q', SORMBR overwrites the general real M-by-N matrix C with SIDE = 'L' SIDE = 'R' TRANS = 'N'
- SORMHR - overwrites the general real M-by-N matrix C
- SORML2 - overwrites the general real m by n matrix C

- SORMLQ - overwrites the general real M-by-N matrix C
- SORMQL - overwrites the general real M-by-N matrix C
- SORMQR - overwrites the general real M-by-N matrix C
- SORMR2 - overwrites the general real m by n matrix C
- SORMR3 - overwrites the general real m by n matrix C
- SORMRQ - overwrites the general real M-by-N matrix C
- SORMRZ - overwrites the general real M-by-N matrix C
- SORMTR - overwrites the general real M-by-N matrix C
- SPBCON - estimates the reciprocal of the condition number (in the 1-norm) of a real symmetric positive definite band matrix using the Cholesky factorization computed by SPBTRF
- SPBEQU - computes row and column scalings intended to equilibrate a symmetric positive definite band matrix A and reduce its condition number (with respect to the two-norm)
- SPBRFS - improves the computed solution to a system of linear equations when the coefficient matrix is symmetric positive definite and banded
- SPBSTF - computes a split Cholesky factorization of a real symmetric positive definite band matrix A
- SPBSV - computes the solution to a real system of linear equations
- SPBSVX - uses the Cholesky factorization to compute the solution to a real system of linear equations
- SPBTF2 - computes the Cholesky factorization of a real symmetric positive definite band matrix A
- SPBTRF - computes the Cholesky factorization of a real symmetric positive definite band matrix A
- SPBTRS - solves a system of linear equations with a symmetric positive definite band matrix A using the Cholesky factorization computed by SPBTRF
- SPOCON - estimates the reciprocal of the condition number (in the 1-norm) of a real symmetric positive definite matrix using the Cholesky factorization computed by SPOTRF

- SPOEQU - computes row and column scalings intended to equilibrate a symmetric positive definite matrix A and reduce its condition number (with respect to the two-norm)
- SPORFS - improves the computed solution to a system of linear equations when the coefficient matrix is symmetric positive definite
- SPOSV - computes the solution to a real system of linear equations
- SPOSVX - uses the Cholesky factorization to compute the solution to a real system of linear equations
- SPOTF2 - computes the Cholesky factorization of a real symmetric positive definite matrix A
- SPOTRF - computes the Cholesky factorization of a real symmetric positive definite matrix A
- SPOTRI - computes the inverse of a real symmetric positive definite matrix A using the Cholesky factorization computed by SPOTRF
- SPOTRS - solves a system of linear equations with a symmetric positive definite matrix A using the Cholesky factorization computed by SPOTRF
- SPPCON - estimates the reciprocal of the condition number (in the 1-norm) of a real symmetric positive definite packed matrix using the Cholesky factorization computed by SPPTRF
- SPPEQU - computes row and column scalings intended to equilibrate a symmetric positive definite matrix A in packed storage and reduce its condition number (with respect to the two-norm)
- SPPRFS - improves the computed solution to a system of linear equations when the coefficient matrix is symmetric positive definite and packed, and provides error bounds and backward error estimates for the solution
- SPPSV - computes the solution to a real system of linear equations
- SPPSVX - uses the Cholesky factorization to compute the solution to a real system of linear equations
- SPPTRF - computes the Cholesky factorization of a real symmetric positive definite matrix A stored in packed format
- SPPTRI - computes the inverse of a real symmetric positive definite matrix A using the Cholesky factorization computed by SPPTRF

- SPPTRS - solves a system of linear equations with a symmetric positive definite matrix A in packed storage using the Cholesky factorization computed by SPPTRF
- SPTCON - computes the reciprocal of the condition number (in the 1-norm) of a real symmetric positive definite tridiagonal matrix using the factorization computed by SPTTRF
- SPTEQR - computes all eigenvalues and, optionally, eigenvectors of a symmetric positive definite tridiagonal matrix by first factoring the matrix using SPTTRF, and then calling SBDSQR to compute the singular values of the bidiagonal factor
- SPTRFS - improves the computed solution to a system of linear equations when the coefficient matrix is symmetric positive definite and tridiagonal, and provides error bounds and backward error estimates for the solution
- SPTSV - computes the solution to a real system of linear equations
- SPTSVX - uses a factorization to compute the solution to a real system of linear equations
- SPTTRF - computes the factorization of a real symmetric positive definite tridiagonal matrix A
- SPTTRS - solves a tridiagonal system
- SPTTS2 - solves a tridiagonal system using the factorization of A computed by SPTTRF
- SRSCCL - multiplies an n-element real vector x by the real scalar 1/a
- SSBEV - computes all the eigenvalues and, optionally, eigenvectors of a real symmetric band matrix A
- SSBEVD - computes all the eigenvalues and, optionally, eigenvectors of a real symmetric band matrix A
- SSBEVX - computes selected eigenvalues and, optionally, eigenvectors of a real symmetric band matrix A
- SSBGST - reduces a real symmetric-definite banded generalized eigenproblem
- SSBGV - computes all the eigenvalues, and optionally, the eigenvectors of a real generalized symmetric-definite banded eigenproblem
- SSBGVD - computes all the eigenvalues, and optionally, the eigenvectors of a real generalized symmetric-definite banded eigenproblem

- SSBGVX - computes selected eigenvalues, and optionally, eigenvectors of a real generalized symmetric-definite banded eigenproblem
- SSBTRD - reduces a real symmetric band matrix A to symmetric tridiagonal form T by an orthogonal similarity transformation
- SSPCON - estimates the reciprocal of the condition number (in the 1-norm) of a real symmetric packed matrix A
- SSPEV - computes all the eigenvalues and, optionally, eigenvectors of a real symmetric matrix A in packed storage
- SSPEVD - computes all the eigenvalues and, optionally, eigenvectors of a real symmetric matrix A in packed storage
- SSPEVX - computes selected eigenvalues and, optionally, eigenvectors of a real symmetric matrix A in packed storage
- SSPGST - reduces a real symmetric-definite generalized eigenproblem to standard form, using packed storage
- SSPGV - computes all the eigenvalues and, optionally, the eigenvectors of a real generalized symmetric-definite eigenproblem
- SSPGVD - computes all the eigenvalues, and optionally, the eigenvectors of a real generalized symmetric-definite eigenproblem
- SSPGVX - computes selected eigenvalues, and optionally, eigenvectors of a real generalized symmetric-definite eigenproblem
- SSPRFS - improves the computed solution to a system of linear equations when the coefficient matrix is symmetric indefinite and packed
- SSPSV - computes the solution to a real system of linear equations
- SSPSVX - uses the diagonal pivoting factorization to compute the solution to a real system of linear equations
- SSPTRD - reduces a real symmetric matrix A stored in packed form to symmetric tridiagonal form T by an orthogonal similarity transformation
- SSPTRF - computes the factorization of a real symmetric matrix A stored in packed format using the Bunch-Kaufman diagonal pivoting method
- SSPTRI - computes the inverse of a real symmetric indefinite matrix A in packed storage using the factorization computed by SSPTRF

- SSPTRS - solves a system of linear equations with a real symmetric matrix A stored in packed format using the factorization computed by SSPTRF
- SSTEBSZ - computes the eigenvalues of a symmetric tridiagonal matrix T
- SSTEBC - computes all eigenvalues and, optionally, eigenvectors of a symmetric tridiagonal matrix using the divide and conquer method
- SSTEGR - computes selected eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix T
- SSTEIN - computes the eigenvectors of a real symmetric tridiagonal matrix T corresponding to specified eigenvalues, using inverse iteration
- SSTEQR - computes all eigenvalues and, optionally, eigenvectors of a symmetric tridiagonal matrix using the implicit QL or QR method
- SSTERF - computes all eigenvalues of a symmetric tridiagonal matrix using the Pal-Walker-Kahan variant of the QL or QR algorithm
- SSTEVE - computes all eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix A
- SSTEVD - computes all eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix
- SSTEVR - computes selected eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix T
- SSTEVSX - computes selected eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix A
- SSYCON - estimates the reciprocal of the condition number (in the 1-norm) of a real symmetric matrix A using the factorization computed by SSYTRF
- SSYEVE - computes all eigenvalues and, optionally, eigenvectors of a real symmetric matrix A
- SSYEVD - computes all eigenvalues and, optionally, eigenvectors of a real symmetric matrix A
- SSYEVR - computes selected eigenvalues and, optionally, eigenvectors of a real symmetric matrix T
- SSYEVSX - computes selected eigenvalues and, optionally, eigenvectors of a real symmetric matrix A

- SSYGS2 - reduces a real symmetric-definite generalized eigenproblem to standard form
- SSYGST - reduces a real symmetric-definite generalized eigenproblem to standard form
- SSYGV - computes all the eigenvalues, and optionally, the eigenvectors of a real generalized symmetric-definite eigenproblem
- SSYGVD - computes all the eigenvalues, and optionally, the eigenvectors of a real generalized symmetric-definite eigenproblem
- SSYGVX - computes selected eigenvalues, and optionally, eigenvectors of a real generalized symmetric-definite eigenproblem
- SSYRFS - improves the computed solution to a system of linear equations when the coefficient matrix is symmetric indefinite
- SSVSV - computes the solution to a real system of linear equations
- SSVSVX - uses the diagonal pivoting factorization to compute the solution to a real system of linear equations
- SSYTD2 - reduces a real symmetric matrix A to symmetric tridiagonal form T by an orthogonal similarity transformation
- SSYTF2 - computes the factorization of a real symmetric matrix A using the Bunch-Kaufman diagonal pivoting method
- SSYTRD - reduces a real symmetric matrix A to real symmetric tridiagonal form T by an orthogonal similarity transformation
- SSYTRF - computes the factorization of a real symmetric matrix A using the Bunch-Kaufman diagonal pivoting method
- SSYTRI - computes the inverse of a real symmetric indefinite matrix A using the factorization computed by SSYTRF
- SSYTRS - solves a system of linear equations with a real symmetric matrix A using the factorization computed by SSYTRF
- STBCON - estimates the reciprocal of the condition number of a triangular band matrix A , in either the 1-norm or the infinity-norm
- STBRFS - provides error bounds and backward error estimates for the solution to a system of linear equations with a triangular band coefficient matrix

- STBTRS - solves a triangular system of the form
- STGEVC - computes some or all of the right and/or left generalized eigenvectors of a pair of real upper triangular matrices (A,B)
- STGEX2 - swaps adjacent diagonal blocks (A11, B11) and (A22, B22) of size 1-by-1 or 2-by-2 in an upper (quasi) triangular matrix pair (A, B) by an orthogonal equivalence transformation
- STGEXC - reorders the generalized real Schur decomposition of a real matrix pair (A,B) using an orthogonal equivalence transformation
- STGSEN - reorders the generalized real Schur decomposition of a real matrix pair (A, B)
- STGSJA - computes the generalized singular value decomposition (GSVD) of two real upper triangular (or trapezoidal) matrices A and B
- STGSNA - estimates reciprocal condition numbers for specified eigenvalues and/or eigenvectors of a matrix pair
- STGSY2 - solves the generalized Sylvester equation
- STGSYL - solves the generalized Sylvester equation
- STPCON - estimates the reciprocal of the condition number of a packed triangular matrix A, in either the 1-norm or the infinity-norm
- STPRFS - provides error bounds and backward error estimates for the solution to a system of linear equations with a triangular packed coefficient matrix
- STPTRI - computes the inverse of a real upper or lower triangular matrix A stored in packed format
- STPTRS - solves a triangular system
- STRCON - estimates the reciprocal of the condition number of a triangular matrix A, in either the 1-norm or the infinity-norm
- STREVC - computes some or all of the right and/or left eigenvectors of a real upper quasi-triangular matrix T
- STREXC - reorders the real Schur factorization of a real matrix
- STRID - computes the solution to a real system of linear equations

- STRRFS - provides error bounds and backward error estimates for the solution to a system of linear equations with a triangular coefficient matrix
- STRSEN - reorders the real Schur factorization of a real matrix
- STRSNA - estimates reciprocal condition numbers for specified eigenvalues and/or right eigenvectors of a real upper quasi-triangular matrix T
- STRSYL - solves the real Sylvester matrix equation
- STRTI2 - computes the inverse of a real upper or lower triangular matrix
- STRTRI - computes the inverse of a real upper or lower triangular matrix A
- STRTRS - solves a triangular system
- STZRQF - routine is deprecated and has been replaced by routine STZRZF
- STZRZF - reduces the M-by-N real upper trapezoidal matrix A to upper triangular form by means of orthogonal transformations
- XERBLA - error handler for the LAPACK routines
- ZBDSQR - computes the singular value decomposition (SVD) of a real N-by-N (upper or lower) bidiagonal matrix B
- ZDRSCL - multiplies an n-element complex vector x by the real scalar 1/a
- ZGBBRD - reduces a complex general m-by-n band matrix A to real upper bidiagonal form B by a unitary transformation
- ZGBCON - estimates the reciprocal of the condition number of a complex general band matrix A, in either the 1-norm or the infinity-norm,
- ZGBEQU - computes row and column scalings intended to equilibrate an M-by-N band matrix A and reduce its condition number
- ZGBRFS - improves the computed solution to a system of linear equations when the coefficient matrix is banded, and provides error bounds and backward error estimates for the solution
- ZGBSV - computes the solution to a complex system of linear equations where A is a band matrix of order N with KL subdiagonals and KU superdiagonals, and X and B are N-by-NRHS matrices
- ZGBSVX - uses the LU factorization to compute the solution to a complex system of linear equations

- ZGBTF2 - computes an LU factorization of a complex m-by-n band matrix A using partial pivoting with row interchanges
- ZGBTRF - computes an LU factorization of a complex m-by-n band matrix A using partial pivoting with row interchanges
- ZGBTRS - solves a system of linear equations with a general band matrix A using the LU factorization computed by ZGBTRF
- ZGEBAK - forms the right or left eigenvectors of a complex general matrix by backward transformation on the computed eigenvectors of the balanced matrix output by ZGEBAL
- ZGEBAL - balances a general complex matrix A
- ZGEBD2 - reduces a complex general m by n matrix A to upper or lower real bidiagonal form B by a unitary transformation
- ZGEBRD - reduces a general complex M-by-N matrix A to upper or lower bidiagonal form B by a unitary transformation
- ZGECON - estimates the reciprocal of the condition number of a general complex matrix A, in either the 1-norm or the infinity-norm, using the LU factorization computed by ZGETRF
- ZGEEQU - computes row and column scalings intended to equilibrate an M-by-N matrix A and reduce its condition number
- ZGEES - computes for an N-by-N complex nonsymmetric matrix A, the eigenvalues, the Schur form T, and, optionally, the matrix of Schur vectors Z
- ZGEESX - computes for an N-by-N complex nonsymmetric matrix A, the eigenvalues, the Schur form T, and, optionally, the matrix of Schur vectors Z
- ZGEEV - computes for an N-by-N complex nonsymmetric matrix A, the eigenvalues and, optionally, the left and/or right eigenvectors
- ZGEEVX - computes for an N-by-N complex nonsymmetric matrix A, the eigenvalues and, optionally, the left and/or right eigenvectors
- ZGEGS - routine is deprecated and has been replaced by routine ZGGES
- ZGEGV - routine is deprecated and has been replaced by routine ZGGEV
- ZGEHD2 - reduces a complex general matrix A to upper Hessenberg form H by a unitary similarity transformation

- ZGEHRD - reduces a complex general matrix A to upper Hessenberg form H by a unitary similarity transformation
- ZGELQ2 - computes an LQ factorization of a complex m by n matrix A
- ZGELQF - computes an LQ factorization of a complex M -by- N matrix A
- ZGELS - solves overdetermined or underdetermined complex linear systems involving an M -by- N matrix A , or its conjugate-transpose, using a QR or LQ factorization of A
- ZGELSD - computes the minimum-norm solution to a real linear least squares problem
- ZGELSS - computes the minimum norm solution to a complex linear least squares problem
- ZGELSX - routine is deprecated and has been replaced by routine ZGELSY
- ZGELSY - computes the minimum-norm solution to a complex linear least squares problem
- ZGEQL2 - computes a QL factorization of a complex m by n matrix A
- ZGEQLF - computes a QL factorization of a complex M -by- N matrix A
- ZGEQP3 - computes a QR factorization with column pivoting of a matrix A
- ZGEQPF - routine is deprecated and has been replaced by routine ZGEQP3
- ZGEQR2 - computes a QR factorization of a complex m by n matrix A
- ZGEQRF - computes a QR factorization of a complex M -by- N matrix A
- ZGERFS - improves the computed solution to a system of linear equations and provides error bounds and backward error estimates for the solution
- ZGERQ2 - computes an RQ factorization of a complex m by n matrix A
- ZGERQF - computes an RQ factorization of a complex M -by- N matrix A
- ZGESL2 - solves a system of linear equations with a general N -by- N matrix A using the LU factorization with complete pivoting computed by ZGETC2
- ZGESDD - computes the singular value decomposition (SVD) of a complex M -by- N matrix A

- ZGESV - computes the solution to a complex system of linear equations
- ZGESVD - computes the singular value decomposition (SVD) of a complex M-by-N matrix A
- ZGESVX - uses the LU factorization to compute the solution to a complex system of linear equations
- ZGETC2 - computes an LU factorization, using complete pivoting, of the n-by-n matrix A
- ZGETF2 - computes an LU factorization of a general m-by-n matrix A using partial pivoting with row interchanges
- ZGETRF - computes an LU factorization of a general M-by-N matrix A using partial pivoting with row interchanges
- ZGETRI - computes the inverse of a matrix using the LU factorization computed by ZGETRF
- ZGETRS - solves a system of linear equations with a general N-by-N matrix A using the LU factorization computed by ZGETRF
- ZGGBAK - forms the right or left eigenvectors of a complex generalized eigenvalue problem by backward transformation on the computed eigenvectors of the balanced pair of matrices output by ZGGBAL
- ZGGBAL - balances a pair of general complex matrices (A,B)
- ZGGES - computes for a pair of N-by-N complex nonsymmetric matrices (A,B), the generalized eigenvalues, the generalized complex Schur form (S, T), and optionally left and/or right Schur vectors (VSL and VSR)
- ZGGESX - computes for a pair of N-by-N complex nonsymmetric matrices (A,B), the generalized eigenvalues, the complex Schur form (S,T),
- ZGGEV - computes for a pair of N-by-N complex nonsymmetric matrices (A,B), the generalized eigenvalues, and optionally, the left and/or right generalized eigenvectors
- ZGGEVX - computes for a pair of N-by-N complex nonsymmetric matrices (A,B) the generalized eigenvalues, and optionally, the left and/or right generalized eigenvectors
- ZGGGLM - solves a general Gauss-Markov linear model (GLM) problem

- ZGGHRD - reduces a pair of complex matrices (A,B) to generalized upper Hessenberg form using unitary transformations, where A is a general matrix and B is upper triangular
- ZGGLSE - solves the linear equality-constrained least squares (LSE) problem
- ZGGQRF - computes a generalized QR factorization of an N-by-M matrix A and an N-by-P matrix B
- ZGGRQF - computes a generalized RQ factorization of an M-by-N matrix A and a P-by-N matrix B
- ZGGSVD - computes the generalized singular value decomposition (GSVD) of an M-by-N complex matrix A and P-by-N complex matrix B
- ZGGSVP - computes unitary matrices U, V and Q
- ZGTCON - estimates the reciprocal of the condition number of a complex tridiagonal matrix A using the LU factorization as computed by ZGTTRF
- ZGTRFS - improves the computed solution to a system of linear equations when the coefficient matrix is tridiagonal, and provides error bounds and backward error estimates for the solution
- ZGTSV - solves the equation $AX = B$
- ZGTSVX - uses the LU factorization to compute the solution to a complex system of linear equations
- ZGTTRF - computes an LU factorization of a complex tridiagonal matrix A using elimination with partial pivoting and row interchanges
- ZGTTRS - solves one of the systems of equations
- ZGTTS2 - solves one of the systems of equations
- ZHBEV - computes all the eigenvalues and, optionally, eigenvectors of a complex Hermitian band matrix A
- ZHBEVD - computes all the eigenvalues and, optionally, eigenvectors of a complex Hermitian band matrix A
- ZHBEVX - computes selected eigenvalues and, optionally, eigenvectors of a complex Hermitian band matrix A

- ZHBGST - reduces a complex Hermitian-definite banded generalized eigenproblem to standard form
- ZHBGV - computes all the eigenvalues, and optionally, the eigenvectors of a complex generalized Hermitian-definite banded eigenproblem
- ZHBGVD - computes all the eigenvalues, and optionally, the eigenvectors of a complex generalized Hermitian-definite banded eigenproblem
- ZHBGVX - computes all the eigenvalues, and optionally, the eigenvectors of a complex generalized Hermitian-definite banded eigenproblem
- ZHBTRD - reduces a complex Hermitian band matrix A to real symmetric tridiagonal form T by a unitary similarity transformation
- ZHECON - estimates the reciprocal of the condition number of a complex Hermitian matrix A using the factorization computed by ZHETRF
- ZHEEV - computes all eigenvalues and, optionally, eigenvectors of a complex Hermitian matrix A
- ZHEEVD - computes all eigenvalues and, optionally, eigenvectors of a complex Hermitian matrix A
- ZHEEVR - computes selected eigenvalues and, optionally, eigenvectors of a complex Hermitian matrix T
- ZHEEVX - computes selected eigenvalues and, optionally, eigenvectors of a complex Hermitian matrix A
- ZHEGS2 - reduces a complex Hermitian-definite generalized eigenproblem to standard form
- ZHEGST - reduces a complex Hermitian-definite generalized eigenproblem to standard form
- ZHEGV - computes all the eigenvalues, and optionally, the eigenvectors of a complex generalized Hermitian-definite eigenproblem
- ZHEGVD - computes all the eigenvalues, and optionally, the eigenvectors of a complex generalized Hermitian-definite eigenproblem
- ZHEGVX - computes selected eigenvalues, and optionally, eigenvectors of a complex generalized Hermitian-definite eigenproblem

- ZHERFS - improves the computed solution to a system of linear equations when the coefficient matrix is Hermitian indefinite, and provides error bounds and backward error estimates for the solution
- ZHESV - computes the solution to a complex system of linear equations
- ZHESVX - uses the diagonal pivoting factorization to compute the solution to a complex system of linear equations
- ZHETD2 - reduces a complex Hermitian matrix A to real symmetric tridiagonal form T by a unitary similarity transformation
- ZHETF2 - computes the factorization of a complex Hermitian matrix A using the Bunch-Kaufman diagonal pivoting method
- ZHETRD - reduces a complex Hermitian matrix A to real symmetric tridiagonal form T by a unitary similarity transformation
- ZHETRF - computes the factorization of a complex Hermitian matrix A using the Bunch-Kaufman diagonal pivoting method
- ZHETRI - computes the inverse of a complex Hermitian indefinite matrix A using the factorization computed by ZHETRF
- ZHETRS - solves a system of linear equations with a complex Hermitian matrix A using the factorization computed by ZHETRF
- ZHGEQZ - implements a single-shift version of the QZ method for finding generalized eigenvalues
- ZHPCON - estimates the reciprocal of the condition number of a complex Hermitian packed matrix A using the factorization computed by ZHPTRF
- ZHPEV - computes all the eigenvalues and, optionally, eigenvectors of a complex Hermitian matrix in packed storage
- ZHPEVD - computes all the eigenvalues and, optionally, eigenvectors of a complex Hermitian matrix A in packed storage
- ZHPEVX - computes selected eigenvalues and, optionally, eigenvectors of a complex Hermitian matrix A in packed storage
- ZHPGST - reduces a complex Hermitian-definite generalized eigenproblem to standard form, using packed storage

- ZHPGV - computes all the eigenvalues and, optionally, the eigenvectors of a complex generalized Hermitian-definite eigenproblem
- ZHPGVD - computes all the eigenvalues and, optionally, the eigenvectors of a complex generalized Hermitian-definite eigenproblem
- ZHPGVX - computes selected eigenvalues and, optionally, eigenvectors of a complex generalized Hermitian-definite eigenproblem
- ZHPRFS - improves the computed solution to a system of linear equations when the coefficient matrix is Hermitian indefinite and packed, and provides error bounds and backward error estimates for the solution
- ZHPSV - computes the solution to a complex system of linear equations
- ZHPSVX - uses the diagonal pivoting factorization to compute the solution to a complex system of linear equations
- ZHPTRD - reduces a complex Hermitian matrix A stored in packed form to real symmetric tridiagonal form T by a unitary similarity transformation
- ZHPTRF - computes the factorization of a complex Hermitian packed matrix A using the Bunch-Kaufman diagonal pivoting method
- ZHPTRI - computes the inverse of a complex Hermitian indefinite matrix A in packed storage using the factorization computed by ZHPTRF
- ZHPTRS - solves a system of linear equations with a complex Hermitian matrix A stored in packed format using the factorization computed by ZHPTRF
- ZHSEIN - uses inverse iteration to find specified right and/or left eigenvectors of a complex upper Hessenberg matrix H
- ZHSEQR - computes the eigenvalues of a complex upper Hessenberg matrix H, and, optionally, the matrices T and Z from the Schur decomposition
- ZLABRD - reduces the first NB rows and columns of a complex general m by n matrix A to upper or lower real bidiagonal form by a unitary transformation
- ZLACGV - conjugates a complex vector of length N
- ZLACON - estimates the 1-norm of a square, complex matrix A
- ZLACP2 - copies all or part of a real two-dimensional matrix A to a complex matrix B

- ZLACPY - copies all or part of a two-dimensional matrix A to another matrix B
- ZLACRM - performs a very simple matrix-matrix multiplication
- ZLACRT - performs the operation $(c\ s)(x) \Rightarrow (x)(-s\ c)(y)(y)$ where c and s are complex and the vectors x and y are complex
- ZLADIV - $:= X / Y$, where X and Y are complex
- ZLAED0 - computes all eigenvalues of a symmetric tridiagonal matrix
- ZLAED7 - computes the updated eigensystem of a diagonal matrix after modification by a rank-one symmetric matrix
- ZLAED8 - merges the two sets of eigenvalues together into a single sorted set
- ZLAEIN - uses inverse iteration to find a right or left eigenvector corresponding to the eigenvalue W of a complex upper Hessenberg matrix H
- ZLAESY - computes the eigendecomposition of a 2-by-2 symmetric matrix
- ZLAEV2 - computes the eigendecomposition of a 2-by-2 Hermitian matrix
- ZLAGS2 - computes 2-by-2 unitary matrices U, V and Q
- ZLAGTM - performs a matrix-vector product
- ZLAHEF - computes a partial factorization of a complex Hermitian matrix A using the Bunch-Kaufman diagonal pivoting method
- ZLAHQR - called by ZHSEQR to update the eigenvalues and Schur decomposition already computed by ZHSEQR
- ZLAHRD - reduces the first NB columns of a complex general n-by-(n-k+1) matrix A so that elements below the kth subdiagonal are zero
- ZLAIC1 - applies one step of incremental condition estimation in its simplest version
- ZLALS0 - applies back the multiplying factors of either the left or the right singular vector matrix of a diagonal matrix appended by a row to the right hand side matrix B in solving the least squares problem using the divide-and-conquer SVD approach
- ZLALSA - an intermediate step in solving the least squares problem by computing the SVD of the coefficient matrix in compact form

- ZLALSD - uses the singular value decomposition of A to solve the least squares problem of finding X to minimize the Euclidean norm
- ZLANGB - returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of an n by n band matrix A,
- ZLANGE - returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a complex matrix A
- ZLANGT - returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a complex tridiagonal matrix A
- ZLANHB - returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of an n by n hermitian band matrix A, with k super-diagonals
- ZLANHE - returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a complex hermitian matrix A
- ZLANHP - returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a complex hermitian matrix A, supplied in packed form
- ZLANHS - returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a Hessenberg matrix A
- ZLANHT - returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a complex Hermitian tridiagonal matrix A
- ZLANSB - returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of an n by n symmetric band matrix A, with k super-diagonals
- ZLANSP - returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a complex symmetric matrix A, supplied in packed form
- ZLANSY - returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a complex symmetric matrix A
- ZLANTB - returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of an n by n triangular band matrix A, with (k + 1) diagonals

- ZLANTP - returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a triangular matrix A , supplied in packed form
- ZLANTR - returns the value of the one norm, or the Frobenius norm, or the infinity norm, or the element of largest absolute value of a trapezoidal or triangular matrix A
- ZLAPLL - computes the QR factorization of $A=QR$
- ZLAPMT - rearranges the columns of the M by N matrix X
- ZLAQGB - equilibrates a general M by N band matrix A with KL subdiagonals and KU superdiagonals using the row and scaling factors in the vectors R and C
- ZLAQGE - equilibrates a general M by N matrix A using the row and scaling factors in the vectors R and C
- ZLAQHB - equilibrates a symmetric band matrix A using the scaling factors in the vector S
- ZLAQHE - equilibrates a Hermitian matrix A using the scaling factors in the vector S
- ZLAQHP - equilibrates a Hermitian matrix A using the scaling factors in the vector S
- ZLAQP2 - computes a QR factorization with column pivoting
- ZLAQPS - computes a step of QR factorization with column pivoting of a complex M -by- N matrix A by using Blas-3
- ZLAQSB - equilibrates a symmetric band matrix A using the scaling factors in the vector S
- ZLAQSP - equilibrates a symmetric matrix A using the scaling factors in the vector S
- ZLAQSY - equilibrates a symmetric matrix A using the scaling factors in the vector S
- ZLAR1V - computes the (scaled) r^{th} column of the inverse of the submatrix
- ZLAR2V - applies a vector of complex plane rotations with real cosines from both sides to a sequence of 2-by-2 complex Hermitian matrices,
- ZLARCM - performs a very simple matrix-matrix multiplication

- ZLARF - applies a complex elementary reflector H to a complex M-by-N matrix C, from either the left or the right
- ZLARFB - applies a complex block reflector H or its transpose H' to a complex M-by-N matrix C, from either the left or the right
- ZLARFG - generates a complex elementary reflector H
- ZLARFT - forms the triangular factor T of a complex block reflector H of order n, which is defined as a product of k elementary reflectors
- ZLARFX - applies a complex elementary reflector H to a complex m by n matrix C, from either the left or the right
- ZLARGV - generates a vector of complex plane rotations with real cosines, determined by elements of the complex vectors x and y
- ZLARNV - returns a vector of n random complex numbers from a uniform or normal distribution
- ZLARRV - computes the eigenvectors of a tridiagonal matrix
- ZLARTG - generates a plane rotation
- ZLARTV - applies a vector of complex plane rotations with real cosines to elements of the complex vectors x and y
- ZLARZ - applies a complex elementary reflector H to a complex M-by-N matrix C, from either the left or the right
- ZLARZB - applies a complex block reflector H or its transpose to a complex distributed M-by-N C from the left or the right
- ZLARZT - forms the triangular factor T of a complex block reflector which is defined as a product of k elementary reflectors
- ZLASCL - multiplies the M by N complex matrix A by the real scalar CTO/CFROM
- ZLASET - initializes a 2-D array A to BETA on the diagonal and ALPHA on the offdiagonals
- ZLASR - performs a transformation where A is an m by n complex matrix and P is an orthogonal matrix
- ZLASSQ - returns the values scl and ssq
- ZLASWP - performs a series of row interchanges on the matrix A

- ZLASZF - computes a partial factorization of a complex symmetric matrix A using the Bunch-Kaufman diagonal pivoting method
- ZLATBS - solves triangular systems
- ZLATDF - computes the contribution to the reciprocal Dif-estimate
- ZLATPS - solves triangular systems
- ZLATRD - reduces NB rows and columns of a complex Hermitian matrix A to Hermitian tridiagonal form
- ZLATRS - solves triangular systems
- ZLATRZ - factors the M -by- $(M+L)$ complex upper trapezoidal matrix
- ZLATZM - routine is deprecated and has been replaced by routine ZUNMRZ
- ZLAUU2 - computes the product $U \times U'$ or $L' \times L$
- ZLAUUM - computes the product $U \times U'$ or $L' \times L$
- ZPBCON - estimates the reciprocal of the condition number (in the 1-norm) of a complex Hermitian positive definite band matrix
- ZPBEQU - computes row and column scalings intended to equilibrate a Hermitian positive definite band matrix A
- ZPBRFS - improves the computed solution to a system of linear equations when the coefficient matrix is Hermitian positive definite and banded
- ZPBSTF - computes a split Cholesky factorization of a complex Hermitian positive definite band matrix A
- ZPBSV - computes the solution to a complex system of linear equations
- ZPBSVX - uses the Cholesky factorization to compute the solution to a complex system of linear equations
- ZPBTF2 - computes the Cholesky factorization of a complex Hermitian positive definite band matrix A
- ZPBTRF - computes the Cholesky factorization of a complex Hermitian positive definite band matrix A
- ZPBTRS - solves a system of linear equations with a Hermitian positive definite band matrix A using the Cholesky factorization computed by ZPBTRF

- ZPOCON - estimates the reciprocal of the condition number (in the 1-norm) of a complex Hermitian positive definite matrix using the Cholesky factorization computed by ZPOTRF
- ZPOEQU - computes row and column scalings intended to equilibrate a Hermitian positive definite matrix A and reduce its condition number (with respect to the two-norm)
- ZPORFS - improves the computed solution to a system of linear equations when the coefficient matrix is Hermitian positive definite,
- ZPOSV - computes the solution to a complex system of linear equations
- ZPOSVX - uses the Cholesky factorization to compute the solution to a complex system of linear equations
- ZPOTF2 - computes the Cholesky factorization of a complex Hermitian positive definite matrix A
- ZPOTRF - computes the Cholesky factorization of a complex Hermitian positive definite matrix A
- ZPOTRI - computes the inverse of a complex Hermitian positive definite matrix A using the Cholesky factorization computed by ZPOTRF
- ZPOTRS - solves a system of linear equations with a Hermitian positive definite matrix A using the Cholesky factorization computed by ZPOTRF
- ZPPCON - estimates the reciprocal of the condition number (in the 1-norm) of a complex Hermitian positive definite packed matrix using the Cholesky factorization computed by ZPPTRF
- ZPPEQU - computes row and column scalings intended to equilibrate a Hermitian positive definite matrix A in packed storage and reduce its condition number (with respect to the two-norm)
- ZPPRFS - improves the computed solution to a system of linear equations when the coefficient matrix is Hermitian positive definite and packed, and provides error bounds and backward error estimates for the solution
- ZPPSV - computes the solution to a complex system of linear equations
- ZPPSVX - use the Cholesky factorization to compute the solution to a complex system of linear equations

- ZPPTRF - computes the Cholesky factorization of a complex Hermitian positive definite matrix A stored in packed format
- ZPPTRI - computes the inverse of a complex Hermitian positive definite matrix A using the Cholesky factorization computed by ZPPTRF
- ZPPTRS - solves a system of linear equations with a Hermitian positive definite matrix A in packed storage using the Cholesky factorization computed by ZPPTRF
- ZPTCON - computes the reciprocal of the condition number (in the 1-norm) of a complex Hermitian positive definite tridiagonal matrix using the factorization computed by ZPTTRF
- ZPTEQR - computes all eigenvalues and, optionally, eigenvectors of a symmetric positive definite tridiagonal matrix
- ZPTRFS - improves the computed solution to a system of linear equations when the coefficient matrix is Hermitian positive definite and tridiagonal, and provides error bounds and backward error estimates for the solution
- ZPTSV - computes the solution to a complex system of linear equations where A is an N-by-N Hermitian positive definite tridiagonal matrix, and X and B are N-by-NRHS matrices
- ZPTSVX - uses the factorization to compute the solution to a complex system of linear equations where A is an N-by-N Hermitian positive definite tridiagonal matrix and X and B are N-by-NRHS matrices
- ZPTTRF - computes the factorization of a complex Hermitian positive definite tridiagonal matrix A
- ZPTTRS - solves a tridiagonal system of the form using the factorization computed by ZPTTRF
- ZPTTS2 - solves a tridiagonal system of the form using the factorization computed by ZPTTRF
- ZSPCON - estimates the reciprocal of the condition number (in the 1-norm) of a complex symmetric packed matrix A using the factorization computed by ZSPTRF
- ZSPRFS - improves the computed solution to a system of linear equations when the coefficient matrix is symmetric indefinite and packed, and provides error bounds and backward error estimates for the solution
- ZSPSV - computes the solution to a complex system of linear equations

- ZSPSVX - uses the diagonal pivoting factorization to compute the solution to a complex system of linear equations
- ZSPTRF - computes the factorization of a complex symmetric matrix A stored in packed format using the Bunch-Kaufman diagonal pivoting method
- ZSPTRI - computes the inverse of a complex symmetric indefinite matrix A in packed storage using the factorization computed by ZSPTRF
- ZSPTRS - solves a system of linear equations with a complex symmetric matrix A stored in packed format using the factorization computed by ZSPTRF
- ZSTEDC - computes all eigenvalues and, optionally, eigenvectors of a symmetric tridiagonal matrix using the divide and conquer method
- ZSTEGR - computes selected eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix T
- ZSTEIN - computes the eigenvectors of a real symmetric tridiagonal matrix T corresponding to specified eigenvalues, using inverse iteration
- ZSTEQR - computes all eigenvalues and, optionally, eigenvectors of a symmetric tridiagonal matrix using the implicit QL or QR method
- ZSYCON - estimates the reciprocal of the condition number (in the 1-norm) of a complex symmetric matrix A using the factorization computed by ZSYTRF
- ZSYRFS - improves the computed solution to a system of linear equations when the coefficient matrix is symmetric indefinite, and provides error bounds and backward error estimates for the solution
- ZSYSV - computes the solution to a complex system of linear equations
- ZSYSVX - uses the diagonal pivoting factorization to compute the solution to a complex system of linear equations
- ZSYTF2 - computes the factorization of a complex symmetric matrix A using the Bunch-Kaufman diagonal pivoting method
- ZSYTRF - computes the factorization of a complex symmetric matrix A using the Bunch-Kaufman diagonal pivoting method
- ZSYTRI - computes the inverse of a complex symmetric indefinite matrix A using the factorization computed by ZSYTRF

- ZSYTRS - solves a system of linear equations with a complex symmetric matrix A using the factorization computed by ZSYTRF
- ZTBCON - estimates the reciprocal of the condition number of a triangular band matrix A , in either the 1-norm or the infinity-norm
- ZTBRFS - provides error bounds and backward error estimates for the solution to a system of linear equations with a triangular band coefficient matrix
- ZTBTRS - solves a triangular system
- ZTGEVC - computes some or all of the right and/or left generalized eigenvectors of a pair of complex upper triangular matrices (A,B)
- ZTGEX2 - swaps adjacent diagonal 1 by 1 blocks (A_{11},B_{11}) and (A_{22},B_{22})
- ZTGEXC - reorders the generalized Schur decomposition of a complex matrix pair (A,B)
- ZTGSEN - reorders the generalized Schur decomposition of a complex matrix pair (A, B)
- ZTGSJA - computes the generalized singular value decomposition (GSVD) of two complex upper triangular (or trapezoidal) matrices A and B
- ZTGSNA - estimates reciprocal condition numbers for specified eigenvalues and/or eigenvectors of a matrix pair (A, B)
- ZTGSY2 - solves the generalized Sylvester equation
- ZTGSYL - solves the generalized Sylvester equation
- ZTPCON - estimates the reciprocal of the condition number of a packed triangular matrix A , in either the 1-norm or the infinity-norm
- ZTPRFS - provides error bounds and backward error estimates for the solution to a system of linear equations with a triangular packed coefficient matrix
- ZTPTRI - computes the inverse of a complex upper or lower triangular matrix A stored in packed format
- ZTPTRS - solves a triangular system
- ZTRCON - estimates the reciprocal of the condition number of a triangular matrix A , in either the 1-norm or the infinity-norm

- ZTREVC - computes some or all of the right and/or left eigenvectors of a complex upper triangular matrix T
- ZTREXC - reorders the Schur factorization of a complex matrix so that the diagonal element of T with row index IFST is moved to row ILST
- ZTRID - computes the solution to a complex system of linear equations where A is an N-by-N tridiagonal matrix, and x and b are vectors of length N
- ZTRRFS - provides error bounds and backward error estimates for the solution to a system of linear equations with a triangular coefficient matrix
- ZTRSEN - reorders the Schur factorization of a complex matrix
- ZTRSNA - estimates reciprocal condition numbers for specified eigenvalues and/or right eigenvectors of a complex upper triangular matrix T
- ZTRSYL - solves the complex Sylvester matrix equation
- ZTRTI2 - computes the inverse of a complex upper or lower triangular matrix
- ZTRTRI - computes the inverse of a complex upper or lower triangular matrix A
- ZTRTRS - solves a triangular system
- ZTZRQF - routine is deprecated and has been replaced by routine ZTZRZF
- ZTZRZF - reduces the M-by-N complex upper trapezoidal matrix A to upper triangular form by means of unitary transformations
- ZUNG2L - generates an m by n complex matrix Q with orthonormal columns,
- ZUNG2R - generates an m by n complex matrix Q with orthonormal columns,
- ZUNGBR - generates one of the complex unitary matrices determined by ZGEBRD when reducing a complex matrix A to bidiagonal form
- ZUNGHR - generates a complex unitary matrix Q
- ZUNGL2 - generates an m-by-n complex matrix Q with orthonormal rows,
- ZUNGLQ - generates an M-by-N complex matrix Q with orthonormal rows,
- ZUNGQL - generates an M-by-N complex matrix Q with orthonormal columns,
- ZUNGQR - generates an M-by-N complex matrix Q with orthonormal columns,
- ZUNGR2 - generates an m by n complex matrix Q with orthonormal rows,

- ZUNGRQ - generates an M-by-N complex matrix Q with orthonormal rows,
- ZUNGTR - generates a complex unitary matrix Q which is defined as the product of n-1 elementary reflectors of order N, as returned by ZHETRD
- ZUNM2L - overwrites the general complex m-by-n matrix C
- ZUNM2R - overwrites the general complex m-by-n matrix C
- ZUNMBR - overwrites the general complex M-by-N matrix C
- ZUNMHR - overwrites the general complex M-by-N matrix C
- ZUNML2 - overwrites the general complex m-by-n matrix C
- ZUNMLQ - overwrites the general complex M-by-N matrix C
- ZUNMLQ - overwrites the general complex M-by-N matrix C
- ZUNMQL - overwrites the general complex M-by-N matrix C
- ZUNMQR - overwrites the general complex M-by-N matrix C
- ZUNMR2 - overwrites the general complex m-by-n matrix C
- ZUNMR3 - overwrites the general complex m by n matrix C
- ZUNMRQ - overwrites the general complex M-by-N matrix C
- ZUNMRZ - overwrites the general complex M-by-N matrix C
- ZUPGTR - generates a complex unitary matrix Q
- ZUPMTR - overwrites the general complex M-by-N matrix C

Glossary

Basic Linear Algebra Subprogram

A set of commonly used algebraic equations defined by C. L. Lawson, and J. J. Dongarra, in a series of papers (see bibliography of *LAPACK User's Guide*, publication TPD-0003, pp. 112–115, entries [16], [17], [18], [19], and [38]).

BCG

See Bi-Conjugate Gradient Method.

Bi-Conjugate Gradient Method

One of the iterative methods provided through the `DITERATIVE` package of optimized preconditioned iterative methods.

BLAS

See Basic Linear Algebra Subprogram.

CGM

See Conjugate Gradient Method.

CGS

See Bi-Conjugate Gradient Squared Method.

computational routines

Term used to define LAPACK routines that perform a distinct computational task.

Conjugate Gradient Method

One of the iterative methods provided through the `DITERATIVE` package of optimized preconditioned iterative methods.

dedicated environment

A parallel processing environment in which the `NCPUS` environment variable is equal to the number of available processors.

direct solution methods

Direct solution methods for sparse linear systems transform the matrix A into a product of several other operators so that each of the resulting operators is easy to invert for a given right-hand side b .

driver routines

Term used to define LAPACK routines used for solving standard types of problems.

equilibration

The process of scaling a problem before computing its solution.

Fourier analysis

The mathematical process of resolving a given function, $f(x)$, into its frequency components, which means finding the sequence of constant amplitudes to plug into a Fourier series to reconstruct the original function.

Hermitian matrix

A complex matrix which is equal to the conjugate of its transpose, with either the lower or upper triangle being stored.

iterative solution methods

Iterative solution methods for sparse linear systems attempt to solve $Ax = b$ by solving an equivalent system $M^{-1}Ax = M^{-1}b$, where M is some approximation to A which is inexpensive to construct and can be easily used to compute z . Unlike direct methods, iterative methods are more special-purpose. There are no general, effective iterative algorithms for an arbitrary sparse linear system.

LAPACK

A public domain library of subroutines for solving dense linear algebra problems, including systems of linear equations, linear least squares problems, eigenvalue problems, and singular value problems. It has been designed for efficiency on high-performance computers.

linear system

A set of simultaneous linear algebraic equations.

load balancing

The process of dividing work done by each available processor into approximately equal amounts.

multiuser environment

A parallel processing environment in which users do not know how many processors will be available to a job during run time.

out-of-core technique

A term that refers to algorithms that combine input and output with computation to solve problems in which the data resides on disk or some other secondary random-access storage device.

packed storage

A triangular or symmetric matrix in which the full matrix representation is retained while storing only half the matrix elements.

parallel instruction execution

The execution of one instruction per clock period, even those instructions that take several clock periods to complete execution.

Pipelining

A method of execution which allows each step of an operation to pass its result to the next step after only one clock period.

single-threaded code segments

A section of a program that must use a single processor.

small parallel/vector problem

A class of problem size in which problems are large enough for vector and parallel processing, but for which parallel processing degrades vector performance.

sparse matrix

A linear system which can be described as $Ax = b$, where A is an n -by- n matrix, and x and b are n dimensional vectors. A system of this kind is considered *sparse* if the matrix A has a small percentage of nonzero terms (less than 10%, often less than 1%).

SPD

See Symmetric Positive Definite Matrix.

SPVP

See Small Paralell/Vector Problem.

Strassen's algorithm

A recursive algorithm that is slightly faster than the ordinary inner product algorithm. Strassen's algorithm performs the floating-point operations for matrix multiplication in an order differently from the vector method; this can cause round-off problems.

supernodes

A collection of columns that have the same nonzero pattern.

time slicing

A method of execution in which the system works on several jobs or processes simultaneously.

vectorization

A form of parallel processing that uses instruction segmenting and vector registers.

virtual matrices

A virtual matrix is similar to a Fortran array, but it cannot be accessed directly from a program. It can only be accessed with calls to specific subroutines. Users do not do any explicit input or output to read from or write to a virtual matrix.

VP

See Vector Problem.

well-conditioned matrix

The condition number of a matrix is defined as $\kappa(A) = \|A\| \cdot \|A^{-1}\|$. A *well-conditioned* matrix is one for which $\kappa(A)$ is small. Although small is relative, if $\kappa(A) < 10^3$, A can be considered well-conditioned.

Index

B

- banded matrix, 42
- Basic Linear Algebra Subprogram (BLAS), 5
- BLAS routines
 - array storage, 8
 - C interface, 6
 - C/C++ function prototypes, 7
 - casts, 7
 - data types, 5
 - increment arguments, 8
 - integer argument defaults, 6
 - Level 1, 8
 - Level 2, 10
 - Level 3, 12
 - levels, 5
 - list of, 57
 - man page names, 6
 - overview, 5
 - user-defined complex types, 7

C

- C interface
 - in BLAS routines, 6
- compiler options, 1
- complex<double> data type, 6
- computational routines, 16
- computing a simple bound, 27
- condition estimation, 26
- condition number, 26
- convolution routines, 54
- correlation routines, 54

D

- data types
 - BLAS routines, 5
- diagonally dominant matrix, 42
- direct solvers, 41
 - solution techniques, 43
- DITERATIVE, 44
- double precision complex data type, 6
- double precision data type, 6
- driver routines, 16, 25

E

- EISPACK, 13
- equilibration, 29
- error bounds, 27
- error bounds computations, 32
- error codes, 24
- error conditions, 24
- examples
 - error conditions, 24
 - LU factorization, 21
 - orthogonal factorization, 35
 - roundoff errors, 26
 - symmetric indefinite matrix factorization, 22
- explicit form, 20

F

- factored form, 20
- factoring a matrix, 20
- factorization forms, 21
- Fast Fourier Transforms, 47
 - casts, 49

- data types, 47
 - implementation details, 48
 - C/C++ function prototypes, 48
 - data types for variables, 48
 - integer argument defaults, 48
 - isys array, 53
 - scratch space, 53
 - work and table arrays, 51
 - include files, 48
 - supported routines, 49
 - user-defined complex types, 49
 - FFT routines, 47
 - list of, 61
 - Fortran type declarations
 - Level 1 BLAS, 9
- H**
- Hilbert matrix, 31
 - Householder transformation, 37
- I**
- ILAENV, 13, 14
 - increment arguments
 - BLAS routines, 8
 - introductory man pages, 57
 - inverse of dense matrix, 33
 - isys array
 - in FFT, 53
 - iterative refinement, 31
 - iterative solvers, 44
- L**
- LAPACK
 - and tuning parameters, 13
 - computation types, 15
 - data types supported, 13
 - error codes, 24
 - factoring a matrix, 20
 - iterative refinement, 31
 - matrix types, 14
 - naming scheme, 14
 - orthogonal factorizations, 34
 - overview, 13
 - result comparisons, 39
 - simple driver routines, 26
 - solving from the factored form, 25
 - solving linear systems, 16
 - types of problems solved, 15
 - types of routines, 16
 - LAPACK routines
 - list of, 62
 - least squares problem, 15
 - least squares problems
 - solving, 34
 - Level 1 BLAS, 8
 - Fortran type declarations, 9
 - Level 2 BLAS, 10
 - Level 3 BLAS, 12
 - levels of BLAS routines, 5
 - linear system, 41
 - linear system solutions, 15
 - linkage defaults, 1
 - LINPACK, 13
 - list of supported routines, 57
 - LU factorization, 21
- M**
- man pages
 - introductory, 57
 - matrix inversion, 33
- N**
- naming

LAPACK routines, 14

O

orthogonal factorizations, 34
 orthogonal matrix
 generating, 38
 multiplying by, 37
 overdetermined linear system, 15

P

parallel processing
 benefits, 2
 common problems, 2
 costs/benefits discussion, 2
 discussions of, 3
 overhead, 3

Q

QR factorization, 35

R

reciprocal condition number, 26
 roundoff errors, 26

S

scratch space
 in FFT, 53
 SCSL
 compiler options, 1
 linkage defaults, 1
 overview, 1
 sctl_complex data type, 6

signal processing routines, 47
 convolution, 54
 correlation, 54
 FFT, 47
 single precision complex data type, 5
 single precision data type, 5
 solution techniques
 direct methods, 43, 140
 direct solvers, 43
 iterative methods, 45
 sparse linear systems, 43
 solving dense linear systems, 25
 solving linear systems, 16
 sparse linear solvers, 41
 sparse linear systems
 solution techniques, 43
 direct methods, 43, 140
 iterative methods, 45
 sparse matrices
 banded matrix, 42
 diagonally dominant matrix, 42
 overview, 41
 structurally symmetric matrix, 42
 Symmetric Positive Definite matrix, 41
 tridiagonal matrix, 42
 types of, 41
 structurally symmetric matrix, 42
 supported routines, 57
 BLAS routines, 57
 FFT routines, 61
 LAPACK routines, 62
 symmetric indefinite matrix factorization, 22
 Symmetric Positive Definite matrix, 41

T

table array
 FFT, 51
 throughput, 2
 tridiagonal matrix, 42

Tuning parameters, 13

in FFT, 51

U

underdetermined linear system, 15, 34
user-defined complex types, 8, 49

X

XERBLA, 24

W

work array